ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΥΠΟΥΡΓΕΙΟ ΕΣΩΤΕΡΙΚΩΝ

ΕΘΝΙΚΟ ΚΕΝΤΡΟ ΔΗΜΟΣΙΑΣ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΑΥΤΟΔΙΟΙΚΗΣΗΣ

ΥΠΟΕΡΓΟ: ΥΠΟΕΡΓΟ 2 «ΠΡΟΓΡΑΜΜΑΤΑ ΚΑΤΑΡΤΙΣΗΣ, ΑΝΑΠΤΥΞΗΣ ΔΕΞΙΟΤΗΤΩΝ, ΕΝΔΥΝΑΜΩΣΗΣ, ΠΙΣΤΟΠΟΙΗΣΗΣ - ΥΛΟΠΟΙΗΣΗ ΜΕ ΙΔΙΑ ΜΕΣΑ, ΕΠΙΜΟΡΦΩΣΗ ΑΠΟ ΤΟ ΕΚΔΔΑ» του Έργου «SUB4. Αναβάθμιση των δεξιοτήτων του ανθρώπινου δυναμικού του Δημόσιου Τομέα» με κωδικό ΟΠΣ ΤΑ 5150174
της Δράσης 16972 ΤΑΑ

ΤΙΤΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ:
ΑΝΑΛΥΣΗ ΔΕΔΟΜΕΝΩΝ ΜΕ ΤΗ ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΡΥΤΗΟΝ

ΕΚΠΑΙΔΕΥΤΙΚΟ ΥΛΙΚΟ

Κωδικός εκπαιδευτικού υλικού:

Κωδικός Πιστοποίησης προγράμματος: 994

Ελλάδα 2.0
Με τη χρηματοδότηση της Ευρωπαϊκής Ένωσης
NextGenerationEU

ΥΠΟΕΡΓΟ: ΥΠΟΕΡΓΟ 2 «ΠΡΟΓΡΑΜΜΑΤΑ ΚΑΤΑΡΤΙΣΗΣ, ΑΝΑΠΤΥΞΗΣ ΔΕΞΙΟΤΗΤΩΝ, ΕΝΔΥΝΑΜΩΣΗΣ, ΠΙΣΤΟΠΟΙΗΣΗΣ - ΥΛΟΠΟΙΗΣΗ ΜΕ ΙΔΙΑ ΜΕΣΑ, ΕΠΙΜΟΡΦΩΣΗ ΑΠΟ ΤΟ ΕΚΔΔΑ» του Έργου «SUB4. Αναβάθμιση των δεξιοτήτων του ανθρώπινου δυναμικού του Δημόσιου Τομέα» με κωδικό ΟΠΣ ΤΑ 5150174

της Δράσης 16972 ΤΑΑ

ΤΙΤΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ:
ΑΝΑΛΥΣΗ ΔΕΔΟΜΕΝΩΝ ΜΕ ΤΗ ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΡΥΤΗΟΝ

ΟΜΑΔΑ ΕΡΓΑΣΙΑΣ

Μέλη Ομάδας

Συντονιστής:
Ιωάννης Ματζαβάκης

Συγγραφείς:
Δρ. Χρήστος Γκόγκος
Δρ. Γεώργιος Χρ. Μακρής
Δρ. Εμμανουήλ Ζούλιας
Δρ. Δημήτριος Καραπιπέρης

Αξιολογητές:
Δρ. Γεώργιος Παπαμιχαήλ
Λάμπρος Κωσταπαππάς

Ελλάδα 2.0
Με τη χρηματοδότηση της Ευρωπαϊκής Ένωσης
NextGenerationEU

ΑΝΑΛΥΣΗ ΔΕΔΟΜΕΝΩΝ ΜΕ ΤΗ ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ PYTHON

Χρήστος Γκόγκος, Γεώργιος Χρ. Μακρής, Εμμανουήλ Ζούλιας, Δημήτριος Καραπιπέρης

Περιεχόμενα

Λίγα λόγια για τους συγγραφείς

Δρ. Χρήστος Γκόγκος

Ο Χρήστος Γκόγκος είναι Μηχανικός Ηλεκτρονικών Υπολογιστών και Πληροφορικής του Πανεπιστημίου Πατρών, κάτοχος μεταπτυχιακού σε «Αλγόριθμους Υψηλής Απόδοσης» από το Τμήμα Πληροφορικής και Τηλεπικοινωνιών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών, και διδακτορικού με τίτλο «Αλγόριθμοι συνδυαστικής βελτιστοποίησης με έμφαση σε μεταευρετικές τεχνικές» από το Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών του Πανεπιστημίου Πατρών. Εργάστηκε σε διάφορες εταιρείες πληροφορικής ως αναλυτής και σχεδιαστής συστημάτων από το 1993 μέχρι και το 2004. Από τον Ιανουάριο του 2004 εργάζεται ως μέλος ΔΕΠ, αρχικά στο ΤΕΙ Ηπείρου και από τον Οκτώβριο του 2018 στο Πανεπιστήμιο Ιωαννίνων. Από τον Μάϊο του 2024, κατέχει θέση Καθηγητή Α' Βαθμίδας στο Τμήμα Πληροφορικής και Τηλεπικοινωνιών του Πανεπιστημίου Ιωαννίνων στην Άρτα με γνωστικό αντικείμενο «Συνδυαστικοί Αλγόριθμοι και Προγραμματισμός με έμφαση στον Χρονοπρογραμματισμό». Τα μαθήματα που διδάσκει είναι «Αντικειμενοστραφής Προγραμματισμός με τη C++», «Αλγόριθμοι και Πολυπλοκότητα», «Αρχές Γλωσσών Προγραμματισμού» και «Επιχειρησιακή Έρευνα». Παράλληλα είναι Συνεργαζόμενο Εκπαιδευτικό Προσωπικό (ΣΕΠ) του Ελληνικού Ανοικτού Πανεπιστημίου όπου έχει διδάξει τη γλώσσα προγραμματισμού Python και έχει το ρόλο του συντονιστή και μέλους ΣΕΠ στη Θεματική Ενότητα «Πληροφοριακά Συστήματα Ηλεκτρονικής Διακυβέρνησης». Επιπλέον, είναι επιμορφωτής του Εθνικού Κέντρου Δημόσια Διοίκησης και Αυτοδιοίκησης όπου και διδάσκει αντικείμενα σχετικά με Python, Ανάλυση Δεδομένων, Βάσεις Δεδομένων και SQL. Ο κ. Γκόγκος ειδικεύεται στην αντιμετώπιση δισεπίλυτων προβλημάτων συνδυαστικής βελτιστοποίησης και στην επίλυσή τους με τεχνικές μαθηματικού προγραμματισμού, ευρετικών και μεταευρετικών προσεγγίσεων. Επιπλέον, διαθέτει σημαντική εμπειρία στην αντιμετώπιση διαφόρων προβλημάτων χρονοπρογραμματισμού (προσωπικού, μηχανών, γραμμών παραγωγής κ.α.) με δυναμικά χαρακτηριστικά και έχει διακριθεί σε διεθνείς διαγωνισμούς χρονοπρογραμματισμού. Έχει δημοσιεύσει εργασίες σε διεθνή επιστημονικά περιοδικά υψηλής αναγνώρισης όπως το European Journal Of Operational Research (Elsevier), το Annals of Operations Research (Springer), το Future Generation Computer Systems (Elsevier), το Journal of Scheduling (Springer), το Cryptography and Communications (Springer) και άλλα. Το ερευνητικό του έργο σε επιστημονικά περιοδικά, κεφάλαια βιβλίων και επιστημονικά συνέδρια έχει λάβει σημαντικό αριθμό ετεροαναφορών (https://scholar.google.gr/citations?user=iNX_7IsAAAAJ). Είναι τακτικός κριτής σε επιστημονικά περιοδικά, συνέδρια και σε αξιολογήσεις έργων. Είναι μέλος επιστημονικών επιτροπών συνεδρίων και περιοδικών. Έχει συμμετάσχει σε μεγάλο αριθμό Ευρωπαϊκών και Εθνικών έργων. Είναι συν-

συγγραφέας του βιβλίου «Μια σύγχρονη προσέγγιση στη γλώσσα C» που είναι ελεύθερα διαθέσιμο μέσω της δράσης Ακαδημαϊκές εκδόσεις ΚΑΛΛΙΠΟΣ.

Δρ. Γεώργιος Μακρής

Ο κος Γεώργιος Μακρής γεννήθηκε το 1971 στο Χέρφορντ της Γερμανίας μεγάλωσε και ζει στην Κατερίνη. Από τον Ιούλιο του 2022 είναι Διευθυντής της Διεύθυνσης Δευτεροβάθμιας Εκπαίδευσης Πιερίας.

Έχει σπουδάσει Μαθηματικά, Πληροφορική και Οικονομικές Επιστήμες. Έχει μεταπτυχιακές σπουδές: α) στην Εκπαίδευση, β) στην Θεωρητική Πληροφορική και Συστήματα Αυτομάτου Ελέγχου, γ) στα Πληροφοριακά Συστήματα και δ) στην Επιστήμη του Διαδικτύου. Η διδακτορική του διατριβή έχει θέμα: «Κρυπτογραφία με Χάος» στο τμήμα Μαθηματικών του ΑΠΘ. Έχει ολοκληρώσει δύο μεταδιδακτορικές έρευνες α) στο τμήμα Μαθηματικών του ΑΠΘ με θέμα: «Ανάπτυξη Νέων Στατιστικών και Υπολογιστικών Εργαλείων για την Ανάλυση Δικτύων» (Development of new statistical and computational tools for the analysis of networks) και β) στο τμήμα Πληροφορικής του ΑΠΘ με θέμα: «Μοντελοποίηση του Οικοσυστήματος του Ανοιχτού Κώδικα».

Η διδακτική του εμπειρία περιλαμβάνει μαθήματα πληροφορικής και μαθηματικών τόσο στην δευτεροβάθμια όσο και στην τριτοβάθμια εκπαίδευση. Έχει διδάξει επί σειρά ετών μαθήματα προπτυχιακού και μεταπτυχιακού επιπέδου.

Τα ερευνητικά του ενδιαφέροντα σχετίζονται με: «Δίκτυα και Πολύπλοκα συστήματα», «Κρυπτογραφία και Στεγανογραφία», «Δυναμικά Συστήματα και Νευρωνικά Δίκτυα» και «Γλώσσες Προγραμματισμού». Είναι συγγραφέας 7 βιβλίων και ενός συλλογικού τόμου. Έχει πλήθος δημοσιεύσεων τόσο σε ξένα όσο και σε ελληνικά περιοδικά και συνέδρια. Ασχολείται με τον προγραμματισμό από το 1985 και έχει ευχέρεια προγραμματισμού σε πολλές γλώσσες όπως: C, C++, Java, Python, Visual Basic, Assembly κλπ.

Εργάστηκε στον ιδιωτικό τομέα για 10 χρόνια. Από το 2002 εργάζεται ως εκπαιδευτικός στο Δημόσιο Σχολείο, διετέλεσε για 6 χρόνια υποδιευθυντής στο 3ο ΓΕΛ Κατερίνης και για 5 χρόνια Διευθυντής στο 4ο ΓΕΛ.

Δρ. Δημήτριος Καραπιπέρης

Ο Δημήτριος Καραπιπέρης είναι διδάσκων στη Σχολή Θετικών Επιστημών και Τεχνολογίας του Διεθνούς Πανεπιστημίου της Ελλάδος (ΔΙ.ΠΑ.Ε.). Είναι επίσης μέλος του συνεργαζόμενου

εκπαιδευτικού προσωπικού στο Ελληνικό Ανοικτό Πανεπιστήμιο και στο Εθνικό Κέντρο Δημόσιας Διοίκησης και Αυτοδιοίκησης.

Η ερευνητική του δραστηριότητα επικεντρώνεται στα κάτωθι πεδία:

- στην Αντιστοίχιση Οντοτήτων (Entity Reoslution), με έμφαση στην ανάπτυξη αλγορίθμων ομοιότητας, δομών δεδομένων, σχημάτων προσεγγιστικού υπολογισμού και επεκτάσιμων (κατανεμημένων) λύσεων που αξιοποιούν διάφορες τεχνικές τυχαιοποίησης,
- στην ανίχνευση λεξικογραφικών αλλά και σημασιολογικών ομοιοτήτων, με χρήση μεγάλων γλωσσικών μοντέλων (LLMs),
- στην εφαρμογή μεθόδων μηχανικής μάθησης και τεχνητής νοημοσύνης σε ετερογενή πεδία, όπως η ιατρική διάγνωση, η πρόβλεψη στην ιατρική διαλογή ασθενών, η πρόβλεψη ποδοσφαιρικών γεγονότων και η μοντελοποίηση αλληλεπιδράσεων φοιτητών με LMSs (Learning Management Systems).

Είναι κάτοχος διδακτορικού διπλώματος από το Ελληνικό Ανοικτό Πανεπιστήμιο και μεταπτυχιακού τίτλου (MSc) από το University of York (Ηνωμένο Βασίλειο). Η διδακτορική του διατριβή παρουσιάστηκε στο **IEEE Intelligent Informatics Bulletin** τον Αύγουστο του 2017.

Έχει συμμετέχει ενεργά στην έρευνα στους τομείς της ολοκλήρωσης και ανάλυσης δεδομένων. Έχει δημοσιεύσει εκτενώς σε επιστημονικά περιοδικά υψηλής απήχησης, όπως τα **TKDE, DMKD, KAIS, TIFS**, **Inf. Systems**, **KBS** και **DPDB**, καθώς και σε πρακτικά μεγάλων διεθνών συνεδρίων, όπως τα **PVLDB, ICDE, ICDM** και **EDBT**.

Δρ. Εμμανουήλ Ζούλιας

Ο Εμμανουήλ Ζούλιας είναι Μέλος Ε.ΔΙ.Π. του τμήματος Νοσηλευτικής του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών με γνωστικό αντικείμενο Πληροφορική της Υγείας. Είναι Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών ΕΜΠ με Διδακτορικό Δίπλωμα στο αντικείμενο των Ευφυών συστημάτων και πιο συγκεκριμένα στη Μηχανική Μάθηση και Εξόρυξη Δεδομένων Υγείας, από το Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών (ΕΜΠ) και την Ιατρική Σχολή (Πανεπιστήμιο Πατρών). Η κύρια έρευνα του επικεντρώνεται στις τεχνολογίες Πληροφορικής και επικοινωνιών στην Υγεία και την εκπαίδευση. Έχει διατελέσει αναλυτής και προγραμματιστής, διαχειριστής πλήθους πληροφοριακών συστημάτων και πλατφορμών διαδικτύου για εφαρμογές στον τομέα εκπαίδευσης, κατάρτισης και υγείας, οι οποίες έχουν αναπτυχθεί με διάφορες τεχνολογίες όπως PHP, JSP, ASP, HTML5, XHTML, JavaScript, Joomla,

Wordpress. Η διδακτική εμπειρία του περιλαμβάνει, μεταξύ άλλων, μαθήματα βιοϊατρικής τεχνολογίας, πληροφορικής της υγείας, βιοστατιστικής, εκπαιδευτικής τεχνολογίας, ανάπτυξης ιστοσελίδων και τεχνολογιών διαδικτύου, δεξιότητες ΤΠΕ και για στην Εθνική Σχολή Δημόσιας Διοίκησης το αντικείμενο της διοίκησης έργων. Επιπλέον στα πλαίσια της δημόσιας διοίκησης έχει μεγάλη εμπειρία σε θέματα εργαλείων αναδιοργάνωσης λειτουργίας δημοσίων υπηρεσιών, ανοικτών δεδομένων, δημοσίων διαβουλεύσεων, λειτουργίας αποθετηρίων και πολιτικών ανοικτής διακυβέρνησης. Έχει διατελέσει προϊστάμενος στο τμήμα Πληροφορικής του Εθνικού Κέντρου Δημόσιας Διοίκησης, επιστημονικός συνεργάτης του ΤΕΙ Αθήνας, ενώ έχει συμμετάσχει σε πλήθος ευρωπαϊκών προγραμμάτων στον τομέα τεχνολογιών της υγείας, τηλε – ιατρικής, τεχνολογιών εκπαίδευσης, τηλε - εκπαίδευσης με διάφορους φορείς. Σήμερα είναι μέλος ΕΔΙΠ του εργαστηρίου Πληροφορικής της Υγείας στο Τμήμα νοσηλευτικής του ΕΚΠΑ.

ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ ΣΤΗΝ PYTHON

Η Python είναι μια ισχυρή γλώσσα προγραμματισμού, εύκολη για εισαγωγή στον προγραμματισμό για τον αρχάριο προγραμματιστή και με προχωρημένα χαρακτηριστικά για τον έμπειρο προγραμματιστή που μπορεί να κατασκευάσει με την Python επαγγελματικές εφαρμογές. Είναι μια γλώσσα προγραμματισμού που υποστηρίζει το αντικειμενοστραφές (object oriented) υπόδειγμα για την ανάπτυξη εφαρμογών, διαθέτει έτοιμη σημαντική λειτουργικότητα στην τυπική βιβλιοθήκη της (standard library) που έρχεται προεγκατεστημένη με τη γλώσσα, ενισχύεται περαιτέρω με ένα πλούσιο οικοσύστημα εξωτερικών βιβλιοθηκών, και μπορεί να χρησιμοποιηθεί διαδραστικά (interactively), γεγονός που επιτρέπει τη γρήγορη δοκιμή κώδικα.

Σε αυτό το κεφάλαιο θα πραγματοποιηθεί μια εισαγωγή σε βασικά χαρακτηριστικά της γλώσσας όπως η σύνταξη των εντολών της Python, οι τύποι δεδομένων, οι εντολές ελέγχου ροής εκτέλεσης, οι συναρτήσεις, ο αντικειμενοστραφής προγραμματισμός με την Python και άλλα. Στο χώρο της ανάλυσης δεδομένων, η Python διατηρεί κυρίαρχη θέση λόγω της απλότητάς της και των πολλών βιβλιοθηκών που διαθέτει και που επιτρέπουν την προχωρημένη ανάλυση δεδομένων στα επίπεδα της περιγραφικής αναλυτικής (descriptive analytics), της προγνωστικής αναλυτικής (predictive analytics) αλλά και της καθοδηγητικής αναλυτικής (prescriptive analytics).

1.1. Γενικά για την Python

Η αρχική ιδέα της γλώσσας προγραμματισμού Python συνελήφθη από τον Ολλανδό επιστήμονα υπολογιστών Guido van Rossum το 1989, ενώ η πρώτη έκδοσή της κυκλοφόρησε το 1991. Έκτοτε η Python έχει εξελιχθεί ραγδαία έτσι ώστε (στις αρχές του 2026 που γράφτηκε το παρόν σύγγραμμα) να αποτελεί κατά γενική ομολογία τη γλώσσα προγραμματισμού με την υψηλότερη δημοτικότητα, μια διάκριση που διατηρεί εδώ και αρκετά χρόνια. Χωρίς να μπορεί να υπολογιστεί με ακρίβεια, εκτιμάται ότι η Python ήταν ήδη κυρίαρχη από πλευράς αποδοχής για αρχάριους προγραμματιστές, για επιστήμονες δεδομένων (data scientists) και για ακαδημαϊκούς χρήστες από το 2018-2019. Στην Εικόνα 1 φαίνεται η κατάταξη της Python στην πρώτη θέση ανάμεσα στις γλώσσες προγραμματισμού σύμφωνα με τις μετρήσεις του TIOBE index τον φεβρουάριο του 2026. Αναζητώντας τους λόγους για τους οποίους η Python παρουσιάζει τόσο υψηλή δημοτικότητα, μπορούν να ανιχνευθούν οι ακόλουθοι:

- Είναι δωρεάν και εύκολα διαθέσιμη σε όλους (π.χ., Windows, MacOS, Linux, κ.λπ.).
- Διαθέτει διερμηνευτή εντολών και σημειωματάρια (jupyter notebooks) για άμεση εκτέλεση τμημάτων κώδικα.

- Αρκετή λειτουργικότητα περιλαμβάνεται στην ίδια την γλώσσα, είναι δηλαδή όπως αναφέρεται μια “batteries included” γλώσσα προγραμματισμού.
- Είναι ανοικτού κώδικα (open source).
- Διαθέτει δωρεάν μεγάλο πλήθος βιβλιοθηκών (π.χ., numpy, pandas, matplotlib, scikit-learn, pytorch κ.α.) που είναι εύκολο να εγκαθίστανται και να απεγκαθίστανται.
- Τα προγράμματα σε Python είναι σύντομα και εύκολα αναγνώσιμα, σε βαθμό μάλιστα που συχνά αναφέρονται ως «εκτελέσιμος ψευδοκώδικας».
- Η κοινότητα της Python είναι ιδιαίτερα υποστηρικτική και υπάρχει πληθώρα διαθέσιμου υλικού για εκμάθηση της Python.

Feb 2026	Feb 2025	Change	Programming Language	Ratings	Change
1	1		Python	21.81%	-2.08%
2	4	↑	C	11.05%	+1.22%
3	2	↓	C++	8.55%	-2.82%
4	3	↓	Java	8.12%	-2.54%
5	5		C#	6.83%	+2.71%
6	6		JavaScript	2.92%	-0.85%
7	10	↑	Visual Basic	2.85%	+0.81%
8	15	⇈	R	2.19%	+1.14%
9	7	↓	SQL	1.93%	-0.93%
10	9	↓	Delphi/Object Pascal	1.88%	-0.29%

Εικόνα 1 – Δείκτης δημοτικότητας γλωσσών προγραμματισμού, TIOBE index, για τον Φεβρουάριο του 2026, https://www.tiobe.com/tiobe-index/

Το όνομα της Python προέκυψε από την κωμική σειρά “Monty Python’s Flying Circus” του BBC της αγγλικής τηλεόρασης και αποτέλεσε εξέλιξη της γλώσσας προγραμματισμού ABC.

Η Python είναι μια γλώσσα προγραμματισμού που υποστηρίζει πολλαπλά προγραμματιστικά υποδείγματα (programming paradigms), όπως τον διαδικασιακό προγραμματισμό, τον αντικειμενοστραφή προγραμματισμό, αλλά ακόμα και σε κάποιο βαθμό τον συναρτησιακό

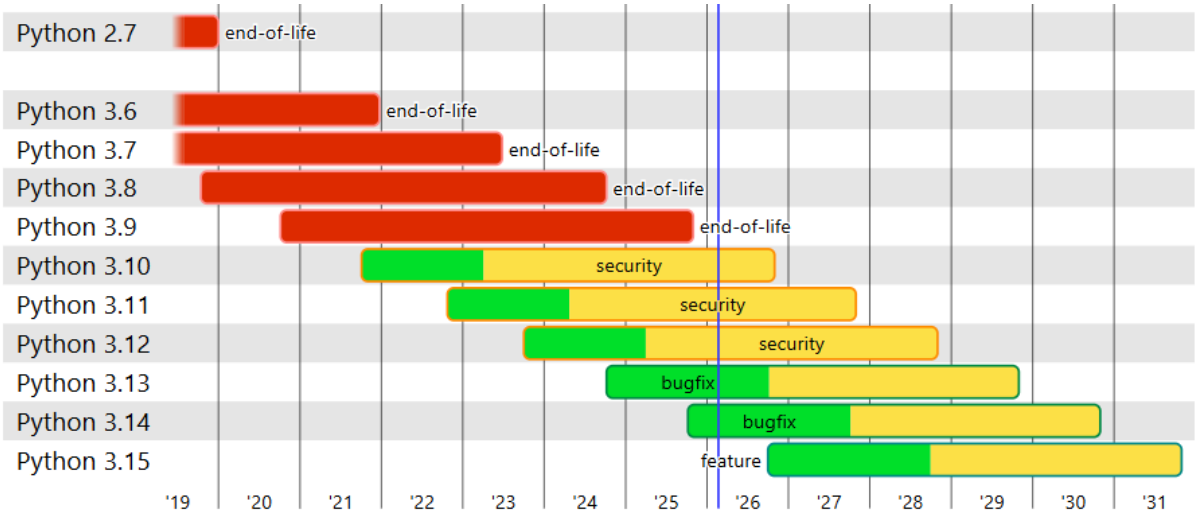
προγραμματισμό. Σε αντίθεση με γλώσσες προγραμματισμού όπως η C και η C++ που είναι μεταγλωττιζόμενες και παράγουν εγγενή (native) κώδικα μηχανής, η Python (στην κύρια υλοποίησή της, την CPython) ακολουθεί υβριδικό μοντέλο εκτέλεσης, μεταγλώττισης και διερμηνείας. Έτσι, ενώ στη C ο μεταγλωττιστής (π.χ., gcc) μετατρέπει τον πηγαίο κώδικα σε δυαδικό εκτελέσιμο κώδικα μέσω των φάσεων προεπεξεργασίας, μεταγλώττισης και σύνδεσης, στην Python ο πηγαίος κώδικας μεταγλωττίζεται αρχικά σε ενδιάμεση μορφή bytecode (αρχεία .pyc) που εκτελείται από την εικονική μηχανή της Python (Python Virtual Machine - PVM). Το χαρακτηριστικό αυτό προσφέρει αυξημένη φορητότητα και ευελιξία, αν και η εκτέλεση μέσω εικονικής μηχανής συνεπάγεται συνήθως μικρότερη απόδοση σε σχέση με γλώσσες που μεταγλωττίζονται απευθείας σε εγγενή κώδικα μηχανής. Επίσης, η Python είναι μια γλώσσα με δυναμικούς τύπους δεδομένων που σημαίνει ότι ο τύπος δεδομένων κάθε μεταβλητής δεν προσδιορίζεται μονοσήμαντα στο πρόγραμμα, αλλά αναγνωρίζεται με βάση την τιμή που κάθε φορά ανατίθεται στη μεταβλητή και μπορεί να αλλάζει κατά τη διάρκεια εκτέλεσης του προγράμματος. Ένα ακόμα χαρακτηριστικό της Python είναι ότι διαθέτει «συλλέκτη απορριμάτων» (garbage collector), όπως και άλλες γλώσσες προγραμματισμού (π.χ., Java), που αναλαμβάνει να καταστήσει ξανά διαθέσιμη τη μνήμη που είχε δεσμευθεί από μεταβλητές αλλά δεν χρησιμοποιείται πλέον. Η ύπαρξη του garbage collector απαλάσσει τον προγραμματιστή από την ευθύνη της χειροκίνητης δέσμευσης και αποδέσμευσης μνήμης και διευκολύνει τη συγγραφή προγραμμάτων.

1.1.1. Διανομές και εκδόσεις της Python

Η κύρια διανομή της Python είναι η CPython, αλλά υπάρχουν και άλλες διανομές όπως η PyPy η οποία διαθέτει Just-In-Time (JIT) μεταγλωττιστή για βελτιωμένη απόδοση, η Jython που εκτελείται πάνω στην πλατφόρμα της Java (JVM), και η IronPython που στοχεύει στο οικοσύστημα του .NET, καθεμία με διαφορετικά χαρακτηριστικά υλοποίησης και πεδία εφαρμογής.

Σχετικά με τις εκδόσεις της γλώσσας, στο παρελθόν υπήρχε η έκδοση 2, που όμως έφτασε στο λεγόμενο «τέλος ζωής» στην 1η Ιανουαρίου του 2020 με τελευταία επιμέρους έκδοση να είναι η 2.7.18. Αυτό σημαίνει ότι δεν πρόκειται να υπάρξει έκδοση 2.8 της Python. Η έκδοση της γλώσσας που εξελίσσεται και συνίσταται να χρησιμοποιείται είναι η έκδοση 3 που τον Φεβρουάριο του 2026 βρίσκεται στην έκδοση 3.14, όπως φαίνεται στην Εικόνα 2. Είναι σημαντικό να αναφερθεί ότι η Python 2 παρουσιάζει ασυμβατότητες συγκριτικά με την Python 3 οι οποίες καθιστούν προγράμματα που είναι γραμμένα σε μια από τις δύο εκδόσεις, μη εκτελέσιμα στην άλλη έκδοση. Από την άλλη μεριά οι επιμέρους εκδόσεις της Python 3, π.χ. 3.14, 3.13, 3.12 κ.ο.κ. έχουν συμβατότητα προς τα πίσω (backwards compatibility) που σημαίνει ότι κώδικας ο οποίος έχει αναπτυχθεί για μια παλαιότερη έκδοση της Python 3 μπορεί, κατά κανόνα, να εκτελεστεί και σε νεότερη έκδοση χωρίς

να απαιτούνται τροποποιήσεις, εκτός εάν χρησιμοποιεί λειτουργικότητες που έχουν ρητά αποσυρθεί ή χαρακτηριστεί ως deprecated (κατηργημένες).



Εικόνα 2 – Εκδόσεις της Python, https://devguide.python.org/versions/

1.1.2. Εγκατάσταση της Python

Υπάρχουν διάφοροι τρόποι με τους οποίους μπορεί να εγκατασταθεί η Python σε ένα υπολογιστικό σύστημα, ενώ σε πολλά συστήματα, όπως σε διάφορες διανομές Linux, η Python είναι ήδη προεγκατεστημένη. Ένας απλός τρόπος εγκατάστασης είναι η λήψη του επίσημου αρχείου εγκατάστασης από τον ιστότοπο του Python Software Foundation (python.org) και η εκτέλεσή του, ακολουθώντας τα βήματα του οδηγού εγκατάστασης για το αντίστοιχο λειτουργικό σύστημα (Windows, macOS ή Linux).

1.1.3. Μετά την εγκατάσταση της Python

Μετά την εγκατάσταση της Python μπορούν να χρησιμοποιηθούν:

- Το **REPL (Read Evaluate Print Loop)** της Python που μπορεί να κληθεί από τη γραμμή εντολών πληκτρολογώντας την εντολή python ή python3 ή py ανάλογα με την εγκατάσταση.
- Το **Python IDLE shell** που επιτρέπει τη αλληλοεπιδραστική εκτέλεση εντολών Python, την εμφάνιση αποτελεσμάτων, αλλά και τη συγγραφή εκτέλεση και αποσφαλμάτωση προγραμμάτων σε Python. Στα Windows δημιουργείται με την εγκατάσταση της Python εικονίδιο σχετικής εφαρμογής.

Ωστόσο, υπάρχουν και άλλοι τρόποι συγγραφής και εκτέλεσης κώδικα σε Python που μπορεί να είναι βολικότεροι για συγγραφή και εκτέλεση αποσπασμάτων κώδικα ή μεγαλύτερων εφαρμογών. Ορισμένοι τέτοιοι τρόποι είναι τα Jupyter Notebooks και τα IDEs (Integrated Development Environments), όπως το Visual Studio Code που θα αναφερθούν στη συνέχεια.

1.1.4. Το REPL της Python

Όπως ήδη αναφέρθηκε, εφόσον έχει προηγηθεί η εγκατάσταση της Python, το REPL μπορεί να ξεκινήσει απλά πληκτρολογώντας στην γραμμή εντολών την εντολή python. Τότε, θα εμφανιστεί ένα περιβάλλον αλληλεπίδρασης του χρήστη με την Python παρόμοιο με την Εικόνα 3. Στο περιβάλλον αυτό ο χρήστης μπορεί να εισάγει εντολές στην προτροπή >>> και να λάβει αποτελέσματα, όπως για παράδειγμα φαίνεται με τα παραδείγματα των εντολών που εμφανίζονται στην Εικόνα 3.

```
Python 3.14.3 (tags/v3.14.3:323c59a, Feb  3 2026, 16:04:56) [MSC v.1944 64
 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2 + 3**2
11
>>> "hello".upper()
'HELLO'
>>> import math
>>> math.sqrt(2)
1.4142135623730951
>>>
```

Εικόνα 3 – Το περιβάλλον REPL, παραδείγματα εντολών.

Τα αρχικά REPL σημαίνουν:

- Read – λήψη εισόδου από τον χρήστη, δηλαδή εισαγωγή python κώδικα και δεδομένων από τον χρήστη.
- Evaluate – αποτίμηση κώδικα.
- Print – εκτύπωση αποτελεσμάτων.
- Loop – επανάληψη της διαδικασίας μέχρι να γίνει έξοδος από τον χρήστη.

Το REPL είναι χρήσιμο για γρήγορο πειραματισμό, αποσφαλμάτωση κώδικα, δοκιμή συναρτήσεων βιβλιοθηκών, κλήση συστήματος βοήθειας, π.χ. >>> help(str) και άλλα. Ειδικότερα, επιτρέπει την άμεση εκτέλεση κώδικα, χωρίς ανάγκη να γραφεί πλήρες πρόγραμμα (script). Η πλοήγηση στο ιστορικό των εντολών γίνεται με τα βελάκια πάνω και κάτω του πληκτρολογίου, ενώ επιτρέπεται αλλά και διευκολύνεται η εισαγωγή εντολών πολλαπλών γραμμών. Η έξοδος από το REPL γίνεται πληκτρολογώντας exit() ή με Ctrl+D. Από την έκδοση 3.13 της Python το REPL διαθέτει syntax highlighting και η έξοδος μπορεί να γίνει πληκτρολογώντας exit, χωρίς παρενθέσεις.

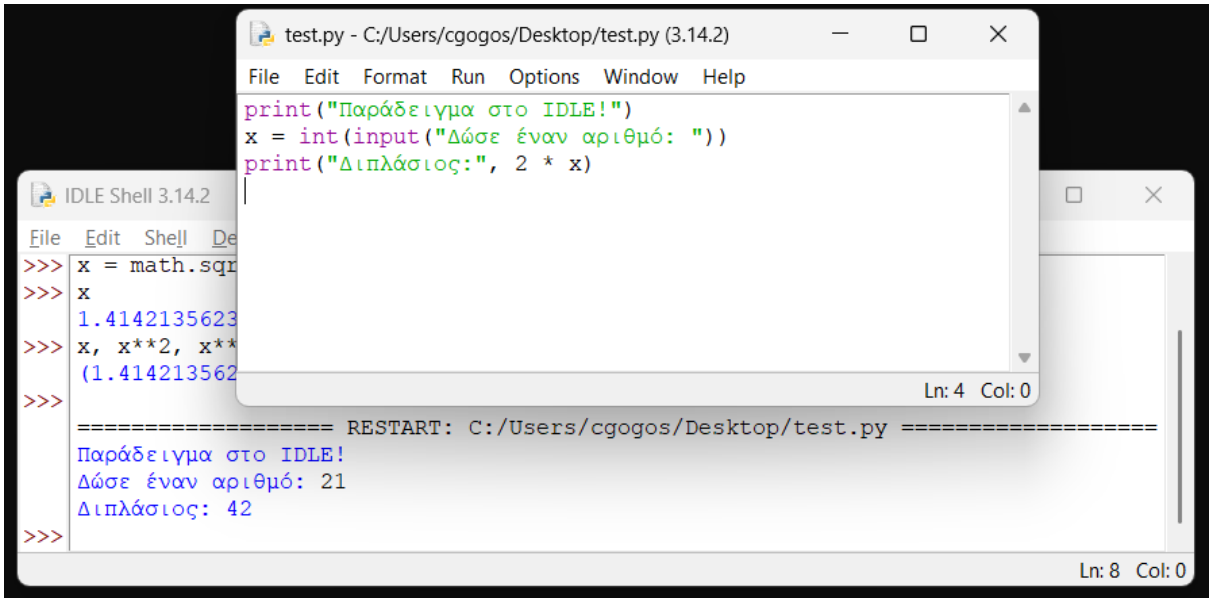
1.1.5. Συγγραφή και εκτέλεση προγραμμάτων με το IDLE

Το IDLE (Integrated Development and Learning Environment) αποτελεί το ενσωματωμένο περιβάλλον ανάπτυξης που συνοδεύει την επίσημη διανομή της Python. Πρόκειται για ένα απλό και ελαφρύ εργαλείο, προσανατολισμένο κυρίως στη διδασκαλία και στην εξοικείωση αρχάριων χρηστών με τη γλώσσα. Περιλαμβάνει διαδραστικό κέλυφος (REPL) για άμεση εκτέλεση εντολών, βασικό επεξεργαστή κώδικα με επισήμανση σύνταξης και αυτόματη στοίχιση, καθώς και στοιχειώδεις

δυνατότητες εκτέλεσης και εντοπισμού σφαλμάτων μέσα από ένα ενιαίο γραφικό περιβάλλον. Αν και δεν διαθέτει τα προηγμένα χαρακτηριστικά επαγγελματικών IDE και δεν ενδείκνυται για μεγάλης κλίμακας εφαρμογές, αποτελεί πρακτική επιλογή για τα πρώτα βήματα στην Python, καθώς λειτουργεί χωρίς σύνθετη παραμετροποίηση και παρέχει άμεση ανατροφοδότηση στον χρήστη.

Για τη δημιουργία ενός νέου προγράμματος στο IDLE, από το παράθυρο του IDLE Shell επιλέγεται η εντολή File → New File, η οποία ανοίγει ένα νέο παράθυρο επεξεργαστή κώδικα. Στο παράθυρο αυτό ο χρήστης μπορεί να πληκτρολογήσει τον πηγαίο κώδικα του προγράμματος.

Για την εκτέλεση του προγράμματος επιλέγεται η εντολή Run → Run Module (ή το πλήκτρο F5). Κατά την πρώτη εκτέλεση, το περιβάλλον ζητά την αποθήκευση του αρχείου με ένα όνομα, για παράδειγμα test.py. Μετά την αποθήκευση, το πρόγραμμα μεταγλωττίζεται σε bytecode και εκτελείται, ενώ η έξοδος και τυχόν μηνύματα σφαλμάτων εμφανίζονται στο παράθυρο του IDLE Shell. Με τον τρόπο αυτό, ο επεξεργαστής χρησιμοποιείται για τη συγγραφή του κώδικα και το Shell για την εκτέλεση και την παρακολούθηση των αποτελεσμάτων.



Εικόνα 4 – Το περιβάλλον συγγραφής και εκτέλεσης προγραμμάτων IDLE.

1.1.6. Συγγραφή και εκτέλεση προγραμμάτων στο VSCode

Το Visual Studio Code (VS Code) είναι ένας σύγχρονος, ελαφρύς αλλά ιδιαίτερα ισχυρός επεξεργαστής κώδικα που αναπτύσσεται από τη Microsoft. Χρησιμοποιείται για ανάπτυξη λογισμικού σε πλήθος γλωσσών προγραμματισμού και διακρίνεται για την επεκτασιμότητα και την ενσωμάτωση προηγμένων εργαλείων που διευκολύνουν τη διαδικασία ανάπτυξης.

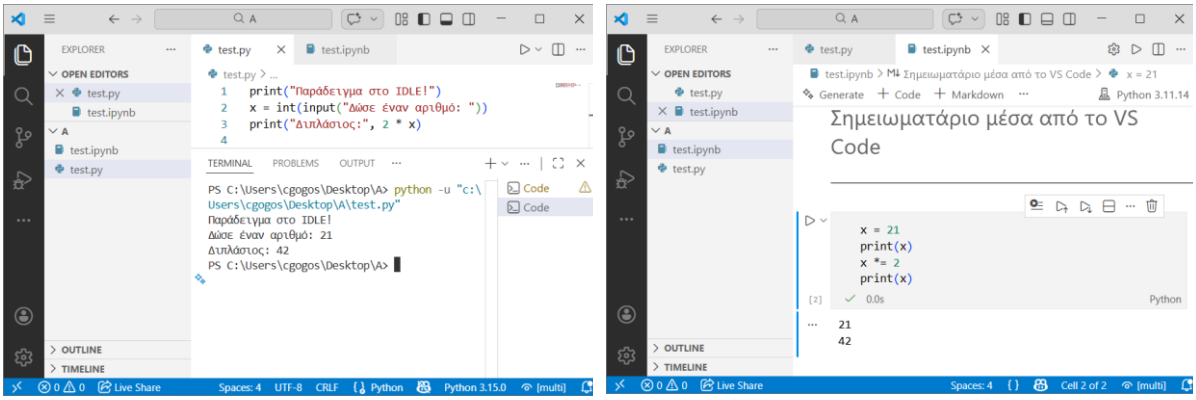
Μεταξύ των βασικών δυνατοτήτων του περιλαμβάνονται η επισήμανση σύνταξης (syntax highlighting), η αυτόματη συμπλήρωση κώδικα (IntelliSense), το ενσωματωμένο τερματικό (integrated terminal), η δυνατότητα αποσφαλμάτωσης (debugging), η πλοήγηση στον κώδικα και η

διαχείριση έργων. Μέσω επεκτάσεων (extensions), το VS Code μπορεί να μετατραπεί σε πλήρες ολοκληρωμένο περιβάλλον ανάπτυξης (IDE), με υποστήριξη για Git, ιδεατά περιβάλλοντα Python, linters, εργαλεία μορφοποίησης και στατικής ανάλυσης κώδικα, δοκιμές (testing frameworks) και πολλά άλλα. Για τον λόγο αυτό χρησιμοποιείται τόσο στην εκπαίδευση όσο και σε επαγγελματικά έργα, από μικρά προγράμματα έως μεγάλης κλίμακας συστήματα λογισμικού που αναπτύσσονται συνεργατικά.

Το VS Code είναι δωρεάν και διατίθεται για Windows, Linux και macOS. Η εγκατάσταση πραγματοποιείται εύκολα ακολουθώντας τις οδηγίες της επίσημης ιστοσελίδας του. Μετά την εγκατάσταση, ο χρήστης μπορεί να ανοίξει έναν φάκελο έργου μέσω της επιλογής File → Open Folder. Ο φάκελος αυτός περιλαμβάνει τα αρχεία που θα επεξεργαστούν και θα εκτελεστούν· αν είναι κενός, μπορούν να δημιουργηθούν νέα αρχεία κώδικα μέσα από το περιβάλλον. Για παράδειγμα, αν έχει δημιουργηθεί ένας φάκελος, μπορεί να επιλεγεί και να ανοίξει στο VS Code. Στη συνέχεια, με την επιλογή File → New File, δημιουργείται ένα νέο αρχείο, π.χ. test.py. Μετά τη συγγραφή του κώδικα και την αποθήκευση του αρχείου, η εκτέλεση μπορεί να πραγματοποιηθεί είτε μέσω της επιλογής Run → Run Without Debugging, είτε μέσω του ενσωματωμένου τερματικού με την εντολή python test.py.

Κατά την πρώτη επεξεργασία αρχείου Python (δηλαδή αρχείου με κατάληξη .py), το VS Code εντοπίζει τον τύπο του αρχείου και προτείνει την εγκατάσταση της επίσημης επέκτασης Python. Η αποδοχή της πρότασης αυτής είναι απαραίτητη για την ενεργοποίηση λειτουργιών όπως εκτέλεση, debugging, αυτόματη συμπλήρωση και επιλογή διερμηνευτή (Python interpreter). Επιπλέον, είναι σκόπιμο να ενεργοποιηθεί η ρύθμιση File → Auto Save, ώστε το αρχείο να αποθηκεύεται αυτόματα πριν από την εκτέλεση. Με τον τρόπο αυτό αποφεύγεται η εκτέλεση παλαιότερης έκδοσης του κώδικα που δεν αντιστοιχεί στο περιεχόμενο που εμφανίζεται στην οθόνη.

Το Visual Studio Code υποστηρίζει την εκτέλεση αρχείων Jupyter Notebook (.ipynb) μέσω της εγκατάστασης της επέκτασης Jupyter. Μετά την εγκατάσταση των σχετικών επεκτάσεων, τα αρχεία .ipynb ανοίγουν σε διαδραστικό περιβάλλον παρόμοιο με αυτό του κλασικού Jupyter Notebook, επιτρέποντας την εκτέλεση κώδικα ανά κελί (cell), την εμφάνιση αποτελεσμάτων, γραφημάτων και πινάκων, καθώς και την ενσωμάτωση κειμένου σε μορφή Markdown. Με τον τρόπο αυτό, όπως φαίνεται στην Εικόνα 5, το VS Code λειτουργεί ως ενιαίο περιβάλλον ανάπτυξης τόσο για αρχεία Python (.py) όσο και για διαδραστικά notebooks, διευκολύνοντας την ανάλυση δεδομένων, τον πειραματισμό και τη διδασκαλία.

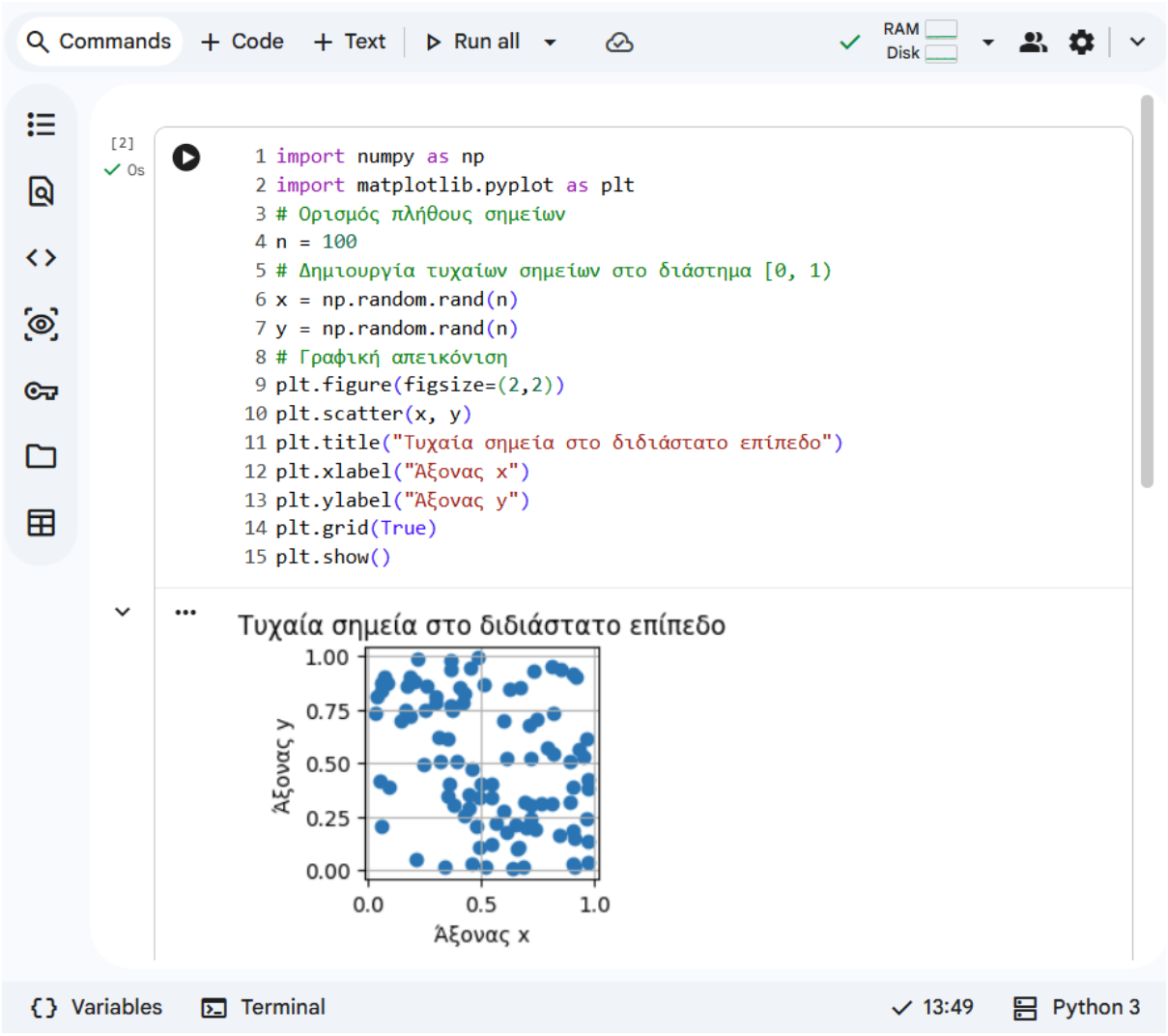


Εικόνα 5 – Εκτέλεση αρχείου κώδικα .py και αρχείου σημειωματαρίου .ipynb στο Visual Studio Code.

1.1.7. Συγγραφή και εκτέλεση τμημάτων κώδικα στο Google Colab

Το Google Colab (Colaboratory) αποτελεί μια διαδικτυακή πλατφόρμα εκτέλεσης κώδικα Python, η οποία βασίζεται στη φιλοσοφία των Jupyter Notebooks και λειτουργεί εξ ολοκλήρου μέσω φυλλομετρητή. Δεν απαιτεί εγκατάσταση της Python στον τοπικό υπολογιστή, καθώς η εκτέλεση πραγματοποιείται σε υπολογιστικούς πόρους που παρέχονται από τη Google ενώ για τη χρήση του απαιτείται λογαριασμός Google. Το Colab περιλαμβάνει προεγκατεστημένες βιβλιοθήκες που χρησιμοποιούνται ευρέως στην ανάλυση δεδομένων και στη μηχανική μάθηση, όπως NumPy, pandas, matplotlib, TensorFlow και PyTorch, ενώ παρέχει και δυνατότητα αξιοποίησης επιταχυντών, όπως GPUs και TPUs, για την επιτάχυνση υπολογισμών. Παράλληλα, υποστηρίζει συνεργατική επεξεργασία σε πραγματικό χρόνο, με αποθήκευση των αρχείων στο Google Drive, γεγονός που το καθιστά ιδιαίτερα κατάλληλο για εκπαιδευτικές και ερευνητικές εφαρμογές.

Η δημιουργία ενός νέου σημειωματαρίου πραγματοποιείται με είσοδο στην πλατφόρμα του Colab (https://colab.research.google.com/) και επιλογή νέου notebook. Το έγγραφο που δημιουργείται αποτελείται από κελιά (cells), στα οποία μπορεί να γραφεί είτε κώδικας Python είτε επεξηγηματικό κείμενο σε μορφή Markdown. Κάθε κελί εκτελείται αυτόνομα, με το κουμπί εκτέλεσης ή με τον συνδυασμό Shift+Enter, και το αποτέλεσμα εμφανίζεται αμέσως από κάτω, επιτρέποντας σταδιακή ανάπτυξη και έλεγχο του προγράμματος. Για παράδειγμα, σε ένα σημειωματάριο μπορεί να συμπεριληφθεί κώδικας που δημιουργεί τυχαία 100 σημεία στο δισδιάστατο χώρο, και παράγει γραφική απεικόνιση σε μορφή διαγράμματος διασποράς. Η σχεδίαση του γραφήματος μπορεί να πραγματοποιηθεί απευθείας με τη βιβλιοθήκη matplotlib (Εικόνα 6), η οποία είναι ήδη διαθέσιμη στο περιβάλλον χωρίς πρόσθετες εντολές εγκατάστασης.



Εικόνα 6 – Δημιουργία ενός γραφήματος διασποράς στο Google Colab.

1.2. Εγκατάσταση βιβλιοθηκών

Η Python διαθέτει ένα πλούσιο οικοσύστημα βιβλιοθηκών (libraries/packages) που επεκτείνουν σημαντικά τις δυνατότητές της, ιδιαίτερα στον χώρο της ανάλυσης δεδομένων. Η εγκατάσταση βιβλιοθηκών γίνεται μέσω ενός διαχειριστή πακέτων και ο προεπιλεγμένος διαχειριστής πακέτων στην Python είναι το pip που επικοινωνεί με το PyPI (Python Package Index - PyPI). Το PyPi είναι το επίσημο κεντρικό αποθετήριο χιλιάδων βιβλιοθηκών της Python όπως το numpy για αριθμητικούς υπολογισμούς, το pandas για ανάλυση δεδομένων, το matplotlib για οπτικοποίηση, το scikit-learn για ανάπτυξη εφαρμογών μηχανικής μάθησης κ.α.

Για να εγκατασταθεί μια βιβλιοθήκη αρκεί, από τη γραμμή εντολών, να δοθεί η εντολή:

```
pip install ονομα_πακέτου
```

Συνίσταται αν υπάρχουν εγκατεστημένες σε ένα σύστημα πολλές εκδόσεις Python η χρήση του διακόπτη -m ως εξής:

```
pip install -m ονομα_πακέτου
```

που διασφαλίζει ότι το pip αντιστοιχεί στην επιθυμητή έκδοση της Python. Έτσι για παράδειγμα για να εγκατασταθεί η βιβλιοθήκη pandas μπορεί να δοθεί η ακόλουθη εντολή:

```
pip install -m pandas
```

Αν είναι επιθυμητό να εγκατασταθεί συγκεκριμένη έκδοση της βιβλιοθήκης, αυτό μπορεί να γίνει με μια εντολή της μορφής:

```
pip install -m pandas==2.2.0
```

1.2.1. Ιδεατά περιβάλλοντα με το venv

Η εγκατάσταση βιβλιοθηκών στη βασική εγκατάσταση της Python δεν θεωρείται καλή πρακτική διότι μπορεί να οδηγήσει σε συγκρούσεις εκδόσεων μεταξύ διαφορετικών έργων (projects), να επηρεάσει άλλα προγράμματα που χρησιμοποιούν την ίδια εγκατάσταση και να δυσκολέψει την αναπαραγωγιμότητα και τη συντήρηση του λογισμικού. Συνεπώς, συνιστάται η δημιουργία και χρήση των λεγόμενων ιδεατών περιβαλλόντων (virtual environments), ώστε η εγκατάσταση επιπλέον βιβλιοθηκών να μην επιβαρύνει τη βασική εγκατάσταση της γλώσσας και να είναι δυνατή η χρήση συγκεκριμένων εκδόσεων βιβλιοθηκών, οι οποίες ενδέχεται να απαιτούνται από διαφορετικά έργα. Η δημιουργία ενός νέου ιδεατού περιβάλλοντος γίνεται σε έναν φάκελο (κατάλογο) του συστήματος αρχείων με την ακόλουθη εντολή, η οποία εκτελείται από τη γραμμή εντολών:

```
python -m venv myenv
```

Η εντολή αυτή δημιουργεί έναν υποκατάλογο με όνομα myenv στον τρέχοντα φάκελο (το όνομα myenv μπορεί να είναι διαφορετικό), ο οποίος περιλαμβάνει μια απομονωμένη εγκατάσταση της Python και των σχετικών εργαλείων.

Η ενεργοποίηση του ιδεατού περιβάλλοντος γίνεται:

- Στα Windows:

```
myenv\Scripts\activate
```

- Στο Linux και στο macOS:

```
source myenv/bin/activate
```

Μετά την ενεργοποίηση, η προτροπή (prompt) της γραμμής εντολών εμφανίζει το όνομα του ιδεατού περιβάλλοντος μέσα σε παρενθέσεις (π.χ. (myenv)), υποδηλώνοντας ότι οι εντολές που ακολουθούν εκτελούνται στο απομονωμένο περιβάλλον. Η εγκατάσταση εξωτερικών βιβλιοθηκών μπορεί πλέον

να πραγματοποιηθεί στο ιδεατό περιβάλλον μέσω του pip. Για παράδειγμα, οι ακόλουθες εντολές εγκαθιστούν τη βιβλιοθήκη pandas και στη συνέχεια εμφανίζουν την έκδοσή της:

```
pip install pandas
python -c "import pandas; print(pandas.__version__)"
```

Η απενεργοποίηση του ιδεατού περιβάλλοντος γίνεται δίνοντας στη γραμμή εντολών την εντολή deactivate.

Εκτός από το εργαλείο venv, που υπάρχει ενσωματωμένο στην Python από την έκδοση 3.3, υπάρχουν και άλλοι τρόποι δημιουργίας και διαχείρισης ιδεατών περιβαλλόντων, όπως το **uv** και το conda. Το uv αποτελεί ένα σύγχρονο, ιδιαίτερα γρήγορο εργαλείο διαχείρισης περιβαλλόντων και εξαρτήσεων, το οποίο μπορεί να δημιουργεί απομονωμένα περιβάλλοντα και να διαχειρίζεται πακέτα με έμφαση στην απόδοση και την αναπαραγωγιμότητα. Το conda, από την άλλη πλευρά, είναι ένα ευρύτερο σύστημα διαχείρισης πακέτων και περιβαλλόντων που δεν περιορίζεται μόνο στην Python, αλλά μπορεί να διαχειρίζεται βιβλιοθήκες γραμμένες και σε άλλες γλώσσες (π.χ. C, C++), γεγονός που το καθιστά ιδιαίτερα δημοφιλές σε επιστημονικές και εφαρμογές ανάλυσης δεδομένων.

1.3. Βασικές έννοιες της Python

Στην παρούσα ενότητα θα παρουσιαστεί ο τρόπος με τον οποίο η Python προσεγγίζει βασικές έννοιες προγραμματισμού όπως οι μεταβλητές, οι τύποι δεδομένων, οι τελεστές, οι εκφράσεις και η είσοδος/έξοδος.

1.3.1. Μεταβλητές

Στην Python μια μεταβλητή είναι ένα αναγνωριστικό (όνομα) που δεσμεύεται δυναμικά σε ένα αντικείμενο της μνήμης. Σε αντίθεση με γλώσσες με στατικούς τύπους (static typing) όπως για παράδειγμα η C, η C++ και η Java, η Python ακολουθεί το dynamic typing (εφαρμογή δυναμικών τύπων), πράγμα που σημαίνει ότι ο τύπος δεν συνδέεται με το όνομα της μεταβλητής αλλά με το αντικείμενο στο οποίο αυτή αναφέρεται. Η ανάθεση τιμής πραγματοποιείται με τον τελεστή =, ο οποίος δημιουργεί ή ενημερώνει τη δέσμευση του ονόματος σε ένα αντικείμενο, χωρίς να απαιτείται ρητή δήλωση τύπου. Για παράδειγμα, το ίδιο όνομα μπορεί διαδοχικά να αναφέρεται σε ακέραιο, σε δεκαδικό ή σε συμβολοσειρά, γεγονός που αποτυπώνει τη δυναμική φύση της γλώσσας.

Στο ακόλουθο απόσπασμα κώδικα (Κ. 1.3.1), στο REPL, φαίνεται ότι η μεταβλητή x αρχικά προσδένεται (γίνεται bound) σε μια ακέραια τιμή, μετά σε μια συμβολοσειρά και τέλος σε έναν αριθμό κινητής υποδιαστολής. Η συνάρτηση id() επιστρέφει την ταυτότητα του αντικειμένου που δέχεται ως όρισμα, ενώ η type() επιστρέφει τον τύπο του αντικειμένου.

```
>>> x = 1729
>>> id(x), type(x)
(2290839803824, <class 'int'>)
>>> x = "Use python!"
>>> id(x), type(x)
(2290841288240, <class 'str'>)
>>> x = 1.618
>>> id(x), type(x)
(2290826590992, <class 'float'>)
```

Κ. 1.3.1 – Πρόσδεση μεταβλητής σε αντικείμενα διαφορετικών τύπων.

Τα ονόματα των μεταβλητών σχηματίζονται από γράμματα (λατινικούς χαρακτήρες Unicode), ψηφία και τον χαρακτήρα υπογράμμισης (_). Υπάρχει διάκριση μεταξύ πεζών και κεφαλαίων χαρακτήρων (case-sensitive), επομένως τα ονόματα value, Value και VALUE θεωρούνται διαφορετικά. Επίσης, τα ονόματα μεταβλητών (και γενικότερα τα αναγνωριστικά, όπως είναι για παράδειγμα τα ονόματα συναρτήσεων) δεν επιτρέπεται να αρχίζουν από ψηφίο και δεν μπορούν να ταυτίζονται με δεσμευμένες λέξεις (keywords). Οι δεσμευμένες λέξεις της Python είναι οι ακόλουθες:

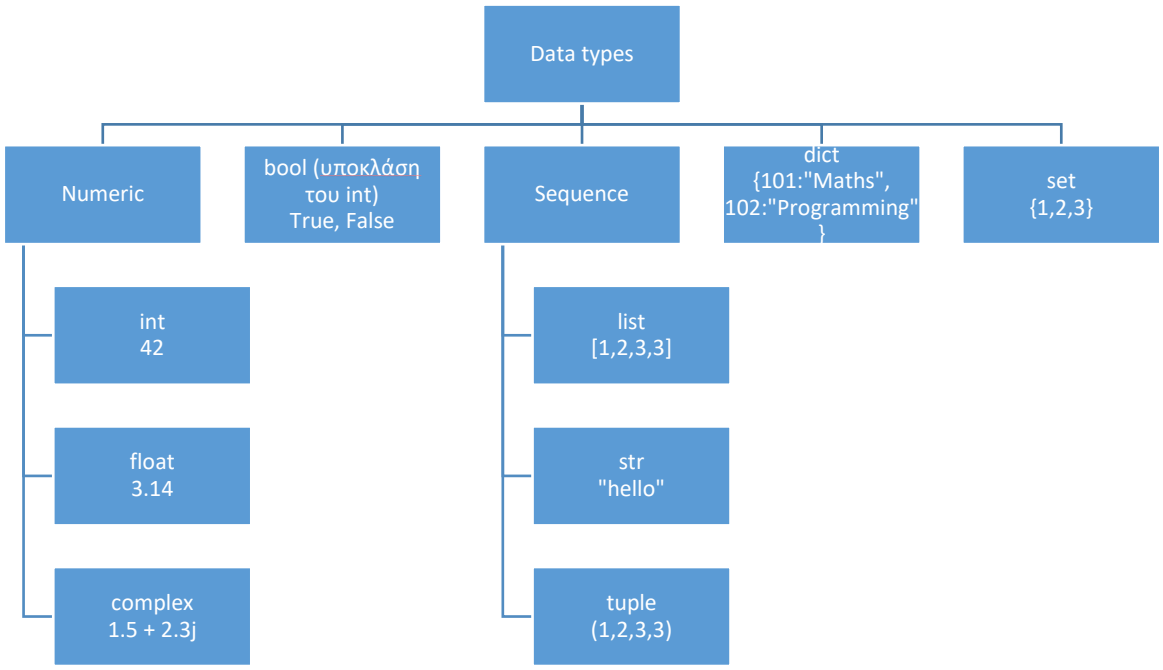
and	as	assert	async	await	break
case	class	continue	def	del	elif
else	except	False	finally	for	from
global	if	import	in	is	lambda
match	None	nonlocal	not	or	pass
raise	return	True	try	while	with
yield					

Πίνακας 1 – Οι 35 δεσμευμένες λέξεις της Python στην έκδοση 3.14 (προστέθηκαν 2 δεσμευμένες λέξεις στην Python 3.10, η match και η case).

1.3.2. Τύποι δεδομένων

Η Εικόνα 7 παρουσιάζει τις βασικές κατηγορίες ενσωματωμένων τύπων δεδομένων της Python, οι οποίες διακρίνονται σε πέντε κύριες ομάδες: Numeric, Boolean, Sequence, Dictionary και Set. Οι αριθμητικοί τύποι (Numeric) περιλαμβάνουν τον ακέραιο (π.χ. 42), τον αριθμό κινητής υποδιαστολής (π.χ. 3.14) και τον μιγαδικό αριθμό (π.χ. 1.5 + 2.3j), με ονομασίες int, float και complex αντίστοιχα. Ο τύπος bool, ο οποίος αποτελεί υποκλάση του int, αναπαριστά τις λογικές τιμές True και False.

Οι ακολουθίες (Sequence) περιλαμβάνουν τις δομές list (π.χ. [1, 2, 3, 3]), str (π.χ. "hello") και tuple (π.χ. (1, 2, 3, 3)), οι οποίες αποθηκεύουν διατεταγμένα στοιχεία. Τα λεξικά (Dictionary), των οποίων ο τύπος είναι dict (π.χ. {101: "Maths", 102: "Programming"}), αποθηκεύουν ζεύγη κλειδιού–τιμής, ενώ τα σύνολα (Set), τύπου set (π.χ. {1, 2, 3}), αποτελούν μη διατεταγμένα σύνολα μοναδικών στοιχείων.



Εικόνα 7 – Βασικοί τύποι δεδομένων της Python.

1.3.2.1 Συμβολοσειρές

Ένας τύπος δεδομένων με ιδιαίτερη σημασία είναι ο τύπος της συμβολοσειράς. Μια συμβολοσειρά ορίζεται με μονά ('...') ή διπλά εισαγωγικά ("..."), ενώ για κείμενο πολλών σειρών μπορούν να χρησιμοποιηθούν τριπλά εισαγωγικά ('''...''' ή """..."""). Οι συμβολοσειρές στην Python είναι ακολουθίες χαρακτήρων Unicode, γεγονός που επιτρέπει τη χρήση χαρακτήρων από διαφορετικά αλφάβητα, καθώς και ειδικά σύμβολα.

Κάθε χαρακτήρας μιας συμβολοσειράς μπορεί να προσπελαστεί μέσω δεικτοδότησης (indexing), με αρίθμηση που ξεκινά από το μηδέν. Οι συμβολοσειρές είναι αμετάβλητες (immutable), πράγμα που σημαίνει ότι το περιεχόμενό τους δεν μπορεί να τροποποιηθεί μετά τη δημιουργία τους. Οποιαδήποτε πράξη φαίνεται να τις αλλάζει, όπως η συνένωση (+) ή η επανάληψη (*), δημιουργεί ένα νέο αντικείμενο τύπου str.

Οι συμβολοσειρές διαθέτουν πληθώρα μεθόδων, όπως είναι οι: upper(), lower(), strip(), replace() και split(), οι οποίες επιτρέπουν μετασχηματισμούς κειμένου χωρίς να αλλοιώνεται η αρχική συμβολοσειρά. Επιπλέον, η Python υποστηρίζει σύγχρονους μηχανισμούς μορφοποίησης συμβολοσειρών, όπως τα f-strings (π.χ. f"Το αποτέλεσμα είναι {x}"), τη μέθοδο format() και τον παλαιότερο τελεστή %. Οι μηχανισμοί αυτοί επιτρέπουν την ενσωμάτωση τιμών μεταβλητών μέσα σε συμβολοσειρές με ευέλικτο και εύκολα αναγνώσιμο τρόπο.

1.3.2.1.1 Δεικτοδότηση συμβολοσειρών

Στην Python, κάθε χαρακτήρας μιας συμβολοσειράς διαθέτει έναν δείκτη (index), μέσω του οποίου μπορεί να προσπελαστεί μεμονωμένα. Η δεικτοδότηση ξεκινά από το 0 για τον πρώτο χαρακτήρα (αριστερά) και αυξάνεται κατά 1 για κάθε επόμενο χαρακτήρα προς τα δεξιά. Για παράδειγμα, στη συμβολοσειρά:

s = "this is a test"

ο χαρακτήρας 't' στη θέση 0 είναι ο πρώτος χαρακτήρας, ενώ ο τελευταίος χαρακτήρας 't' βρίσκεται στη θέση 13. Επιπλέον, η Python υποστηρίζει και αρνητική δεικτοδότηση όπως φαίνεται στον Πίνακας 2. Για την αρνητική δεικτοδότηση, η μέτρηση ξεκινά από το τέλος της συμβολοσειράς: ο τελευταίος χαρακτήρας έχει δείκτη -1, ο προτελευταίος -2 κ.ο.κ.

t	**h**	**i**	**s**		**i**	**s**		**a**		**t**	**e**	**s**	**t**
0	1	2	3	4	5	6	7	8	9	10	11	12	13
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Πίνακας 2 – Δεικτοδότηση συμβολοσειράς "this is a test".

1.3.2.1.2 Τεμαχισμός συμβολοσειρών

Ο τεμαχισμός (slicing) επιτρέπει την εξαγωγή ενός τμήματος μιας συμβολοσειράς με τη σύνταξη:

```
s[αρχή:τέλος:βήμα]
```

όπου:

- **αρχή (start)**: ο δείκτης από τον οποίο ξεκινά το τμήμα (συμπεριλαμβάνεται),
- **τέλος (end)**: ο δείκτης στον οποίο σταματά το τμήμα (δεν συμπεριλαμβάνεται),
- **βήμα (step)**: το διάστημα μετακίνησης μεταξύ χαρακτήρων.

Όλα τα παραπάνω είναι προαιρετικά:

- Αν παραλειφθεί η αρχή, θεωρείται η αρχή της συμβολοσειράς.
- Αν παραλειφθεί το τέλος, θεωρείται το τέλος της συμβολοσειράς.
- Αν παραλειφθεί το βήμα, θεωρείται ίσο με 1.

Μια σημαντική παρατήρηση είναι ότι το τμήμα s[start:end] περιλαμβάνει τον χαρακτήρα στη θέση start, αλλά όχι τον χαρακτήρα στη θέση end. Δηλαδή, το end είναι αποκλειστικό (exclusive). Η δυνατότητα καθορισμού βήματος επιτρέπει, μεταξύ άλλων, την αντιστροφή της συμβολοσειράς (s[::-1]), καθώς και την επιλογή χαρακτήρων με συγκεκριμένη περιοδικότητα. Μια σύνοψη των δυνατοτήτων του τεμαχισμού συμβολοσειρών παρουσιάζεται στον Πίνακας 3.

Τεμαχισμός	Ερμηνεία	Παράδειγμα
`s[start:end:step]`	Τμήμα της s από τον δείκτη start μέχρι και τον δείκτη end-1	`s[5:7]` → `'is'`
`s[start:]`	Τμήμα της s από τον δείκτη start μέχρι το τέλος	`s[8:]` → `'a test'`
`s[:end]`	Τμήμα της s από την αρχή μέχρι μέχρι και τον δείκτη end-1	`s[:4]` → `'this'`
`s[:]`	Αντιγραφή της συμβολοσειράς	`s[:]` → `'this is a test'`
`s[::-1]`	Λήψη αντίστροφής συμβολοσειράς	`s[::-1]` → `'tset a si siht'`

Πίνακας 3 – Δυνατότητες τεμαχισμού συμβολοσειρών, παραδείγματα με τη συμβολοσειρά s='this is a test'.

1.3.2.1.3 Συναρτήσεις και μέθοδοι συμβολοσειρών

Οι συμβολοσειρές στην Python είναι αντικείμενα (objects), γεγονός που σημαίνει ότι διαθέτουν πληθώρα ενσωματωμένων μεθόδων για την επεξεργασία τους. Παράλληλα, μπορούν να χρησιμοποιηθούν και γενικές συναρτήσεις, όπως η len() που επιστρέφει το πλήθος των χαρακτήρων μιας συμβολοσειράς, συμπεριλαμβανομένων των κενών. Για παράδειγμα, αν s = "Python", τότε η len(s) επιστρέφει την τιμή 6.

Από την άλλη μεριά, οι μέθοδοι συμβολοσειρών καλούνται με τη σύνταξη s.μέθοδος() και δεν τροποποιούν την αρχική συμβολοσειρά αλλά επιστρέφουν μια νέα συμβολοσειρά. Μεταξύ των βασικών μεθόδων μετασχηματισμού μορφής συμβολοσειρών περιλαμβάνονται οι upper() και lower() για μετατροπή σε κεφαλαία και πεζά γράμματα αντίστοιχα, καθώς και οι capitalize(), title() και swapcase() για διαφορετικούς τύπους μορφοποίησης κειμένου.

Για αναζήτηση μέσα σε συμβολοσειρές χρησιμοποιούνται οι μέθοδοι find(), index() και count(). Η find(sub) επιστρέφει τη θέση της πρώτης εμφάνισης μιας υποσυμβολοσειράς ή -1 αν δεν βρεθεί, ενώ η index(sub) λειτουργεί παρόμοια αλλά προκαλεί τερματισμό εκτέλεσης με μήνυμα σφάλματος αν το στοιχείο δεν υπάρχει. Η count(sub) επιστρέφει το πλήθος των εμφανίσεων μιας υποσυμβολοσειράς.

Η αντικατάσταση τμημάτων συμβολοσειράς γίνεται με τη μέθοδο replace(old, new, count), όπου προαιρετικά μπορεί να καθοριστεί ο μέγιστος αριθμός αντικαταστάσεων. Για παράδειγμα, "Προγραμματισμός".replace("α", "-") επιστρέφει "Προγρ-μμ-τισμός".

Για διαχωρισμό και σύνθεση συμβολοσειρών χρησιμοποιούνται οι split() και join(). Η split() διαχωρίζει μια συμβολοσειρά σε λίστα με βάση μια άλλη συμβολοσειρα που έχει το ρόλο του

διαχωριστή, ενώ η join() ενώνει τα στοιχεία μιας λίστας σε μία ενιαία συμβολοσειρά. Για παράδειγμα, "α,β,γ".split(",") επιστρέφει ['α', 'β', 'γ'], ενώ η " ".join(["Η", "Python", "είναι", "διασκεδαστική"]) επιστρέφει "Η Python είναι διασκεδαστική".

Επιπλέον, υπάρχουν μέθοδοι ελέγχου περιεχομένου όπως isalpha(), isdigit(), isalnum(), isspace(), καθώς και startswith() και endswith() για έλεγχο προθέματος και επιθέματος. Για αφαίρεση κενών ή άλλων χαρακτήρων από την αρχή και το τέλος χρησιμοποιούνται οι strip(), lstrip() και rstrip().

Τέλος, η μορφοποίηση συμβολοσειρών μπορεί να γίνει με f-strings (π.χ. f"{name} και {age}") ή με τη μέθοδο format(), επιτρέποντας δυναμική ενσωμάτωση τιμών μέσα σε συμβολοσειρές.

1.3.3. Τελεστές

Οι τελεστές (operators) στην Python είναι σύμβολα ή λέξεις-κλειδιά που χρησιμοποιούνται για τον συνδυασμό τιμών. Κάθε τελεστής εφαρμόζεται σε ένα ή περισσότερα ορίσματα (operands) και παράγει ένα αποτέλεσμα. Οι βασικές κατηγορίες τελεστών περιλαμβάνουν τους αριθμητικούς (+, -, *, /, //, %, **), τους συγκριτικούς (==, !=, <, >, <=, >=), τους λογικούς (and, or, not), τους τελεστές ανάθεσης (=, +=, -=, κ.ά.), καθώς και ειδικούς τελεστές όπως ο τελεστής συμμετοχής (in, not in) και ο τελεστής ταυτότητας (is, is not). Μια ακόμα κατηγορία τελεστών αποτελούν οι δυαδικοί τελεστές (bitwise operators), οι οποίοι εφαρμόζονται σε ακέραιους αριθμούς και λειτουργούν πάνω στη δυαδική αναπαράστασή τους. Σε αυτούς ανήκουν οι & (bitwise AND), | (bitwise OR), ^ (bitwise XOR), ~ (bitwise NOT), καθώς και οι τελεστές μετατόπισης << (αριστερή μετατόπιση) και >> (δεξιά μετατόπιση). Οι τελεστές αυτοί χρησιμοποιούνται κυρίως σε χαμηλού επιπέδου υπολογισμούς για λόγους απόδοσης.

1.3.4. Εκφράσεις

Έκφραση (expression) είναι κάθε συντακτική δομή που αποτιμάται και παράγει μια τιμή. Οι εκφράσεις μπορεί να αποτελούνται από σταθερές (π.χ. 42, "hello"), μεταβλητές, τελεστές (αριθμητικούς, συγκριτικούς, λογικούς), κλήσεις συναρτήσεων ή συνδυασμό των προηγούμενων. Για παράδειγμα, η παράσταση 3 * x + 5 αποτελεί αριθμητική έκφραση, ενώ η x > 10 and y != 0 αποτελεί λογική έκφραση που επιστρέφει τιμή τύπου bool. Οι εκφράσεις μπορούν να εμφανίζονται οπουδήποτε αναμένεται τιμή, όπως για παράδειγμα σε αναθέσεις. Η αξιολόγηση των εκφράσεων ακολουθεί συγκεκριμένους κανόνες προτεραιότητας και προσεταιριστικότητας τελεστών (operator precedence και associativity), οι οποίοι καθορίζουν τη σειρά υπολογισμού όταν συνυπάρχουν πολλαπλοί τελεστές.

Στο ακόλουθο απόσπασμα κώδικα (Κ. 1.3.2), παρουσιάζονται στο REPL μερικά παραδείγματα εκφράσεων.

```
>>> # Αριθμητικές εκφράσεις
>>> 2 + 3 * 4
14
>>> 10 // 3
3
>>> # Λογικές εκφράσεις
>>> 7 > 5
True
>>> not (3 == 3)
False
>>> # Εκφράσεις με μεταβλητές
>>> x = 10
>>> x += 1
>>> x
11
>>> # Εκφράσεις με συμβολοσειρές
>>> "Hello " + "world!"
'Hello world!'
>>> "Hello " * 3
'Hello Hello Hello '
```

Κ. 1.3.2 – Αριθμητικές και άλλες εκφράσεις.

1.3.5. Είσοδος/έξοδος

Η είσοδος δεδομένων από τον χρήστη υλοποιείται με τη συνάρτηση input(), η οποία διαβάζει δεδομένα από την τυπική είσοδο (standard input), συνήθως από το πληκτρολόγιο, και επιστρέφει τιμή τύπου str. Η input() μπορεί να δεχθεί προαιρετικά ένα μήνυμα προτροπής, π.χ. name = input("Δώσε όνομα: "). Επειδή το αποτέλεσμα είναι συμβολοσειρά, απαιτείται ρητή μετατροπή όταν αναμένονται αριθμητικά δεδομένα, όπως age = int(input("Δώσε ηλικία: ")).

Η έξοδος πραγματοποιείται με τη συνάρτηση print(), η οποία εμφανίζει δεδομένα στην οθόνη. Η print() μπορεί να δεχθεί πολλαπλά ορίσματα, τα οποία εμφανίζονται διαχωρισμένα με τον κενό χαρακτήρα, ενώ παρέχει και παραμέτρους όπως sep (διαχωριστικό μεταξύ ορισμάτων) και end (χαρακτήρας τερματισμού της γραμμής). Η print() μετατρέπει αυτόματα τα ορίσματα σε συμβολοσειρές πριν την εμφάνιση, γεγονός που επιτρέπει την εκτύπωση αντικειμένων διαφορετικών τύπων χωρίς ρητή μετατροπή.

Στο ακόλουθο απόσπασμα κώδικα (Κ. 1.3.3), παρουσιάζονται στο REPL μερικά παραδείγματα χρήσης των input() και print().

```
>>> name = input("Δώσε όνομα: ")
Δώσε όνομα: Νίκος
>>> name
'Νίκος'
>>> name = input()
Νίκος
>>> name
'Νίκος'
>>> name = input("Δώσε όνομα: ")
Δώσε όνομα: Νίκος
>>> name
'Νίκος'
```

```
>>> age = int(input("Δώσε ηλικία: "))
Δώσε ηλικία: 20
>>> age
20
>>> print("2025", "12", "31", sep="-")
2025-12-31
>>> print("Loading", end="...")
Loading...>>> print("OK")
OK
```

Κ. 1.3.3 – Είσοδος και έξοδος τιμών.

1.4. Εντολές επιλογής και επανάληψης

Οι βασικές δομές ελέγχου ροής ενός προγράμματος περιλαμβάνουν τις εντολές επιλογής (conditional statements) και τις εντολές επανάληψης (repetition statements). Στην Python οι εντολές επιλογής υλοποιούνται μέσω της δομής if / elif / else, της έκφρασης επιλογής (conditional expression ή ternary operator) της μορφής x if condition else y, καθώς και της εντολής match, η οποία εισήχθη στην έκδοση 3.10 και υποστηρίζει δομική αντιστοίχιση προτύπων (structural pattern matching).

Αντίστοιχα, οι βασικές εντολές επανάληψης είναι η while, η οποία εκτελεί επαναληπτικά ένα τμήμα κώδικα όσο μια συνθήκη παραμένει αληθής, και η for, η οποία χρησιμοποιείται κυρίως για επανάληψη πάνω σε στοιχεία μιας ακολουθίας ή γενικότερα ενός επαναλήψιμου αντικειμένου (iterable).

1.4.1. Η εντολή επιλογής if

Η εντολή επιλογής if επιτρέπει την εκτέλεση ενός μπλοκ εντολών ανάλογα με το αποτέλεσμα της αποτίμησης μιας λογικής συνθήκης. Αν η συνθήκη αποτιμηθεί ως True, τότε εκτελείται το αντίστοιχο μπλοκ κώδικα, διαφορετικά παραλείπεται. Στην Python, μπλοκ εντολών είναι μία ή περισσότερες εντολές που βρίσκονται στο ίδιο επίπεδο εσοχής (indentation) και εκτελούνται ως ενιαία λογική ενότητα. Συνεπώς, οι εσοχές στην Python δεν αποτελούν απλώς στοιχείο μορφοποίησης αλλά είναι μέρος της σύνταξης της γλώσσας.

Η γενική σύνταξη της εντολής if παρουσιάζεται στη συνέχεια:

```
if <συνθήκη1>:
      <μπλοκ εντολών 1>
elif <συνθήκη2>:
      <μπλοκ εντολών 2>
...
else:
      <μπλοκ εντολών n>
```

Η εκτέλεση ξεκινά με την αποτίμηση της <συνθήκη1>. Αν η συνθήκη αποτιμηθεί ως True, εκτελείται το αντίστοιχο μπλοκ εντολών και η δομή τερματίζεται, χωρίς να ελεγχθούν οι υπόλοιπες συνθήκες. Αν αποτιμηθεί ως False, η ροή μεταφέρεται στην επόμενη συνθήκη elif. Η διαδικασία αυτή

επαναλαμβάνεται διαδοχικά μέχρι να βρεθεί συνθήκη που είναι αληθής ή να εξαντληθούν όλες οι εναλλακτικές. Η ύπαρξη του elif (συντομογραφία του *else if*) επιτρέπει τον έλεγχο πολλαπλών, αμοιβαία αποκλειόμενων συνθηκών, ενώ το else είναι προαιρετικό και εκτελείται μόνο αν καμία από τις προηγούμενες συνθήκες δεν είναι αληθής. Σε κάθε περίπτωση, εκτελείται το πολύ ένα μπλοκ εντολών από ολόκληρη τη δομή. Όπως έχει ήδη τονιστεί η σωστή χρήση της εσοχής είναι απαραίτητη, καθώς καθορίζει ποια εντολή ανήκει σε κάθε μπλοκ.

Ένα παράδειγμα που χρησιμοποιεί την εντολή if είναι ο κώδικας Κ. 1.4.1 ο οποίος δέχεται έναν ακέραιο βαθμό από τον χρήστη και εμφανίζει τον χαρακτηρισμό της επίδοσης (Αποτυχία, Επιτυχία, Άριστα). Αν ο χρήστης εισάγει την τιμή 17 τότε θα εμφανίσει το μήνυμα «Επιτυχία». Για κάθε πιθανή τιμή που δίνει ο χρήστης εκτελείται μόνο το πρώτο μπλοκ του οποίου η συνθήκη αποτιμάται ως True.

```
grade = int(input("Εισάγετε βαθμό (0-20): "))

if grade < 0 or grade > 20:
    print("Μη έγκυρη τιμή βαθμού")
elif grade < 10:
    print("Αποτυχία")
elif grade < 18:
    print("Επιτυχία")
else:
    print("Άριστα")
```

Κ. 1.4.1 – Ένα παράδειγμα με την εντολή if…elif…else:.

Μια εντολή if μπορεί να βρίσκεται μέσα στο μπλοκ μιας άλλης εντολής if. Στην περίπτωση αυτή προκύπτουν τα λεγόμενα εμφωλευμένα if (nested if), τα οποία επιτρέπουν τη διαδοχική ή ιεραρχική αξιολόγηση πολλαπλών συνθηκών. Αυτό συμβαίνει στον κώδικα Κ. 1.4.2, όπου μέσα στην if που εξετάζει αν η θερμοκρασία είναι μεγαλύτερη του μηδενός, υπάρχει μια δεύτερη if που εξετάζει την ταχύτητα ανέμου. Έτσι αν εισαχθεί θερμοκρασία 5°C και ταχύτητα ανέμου 35km/h τότε θα εμφανιστεί το μήνυμα «Η ένταση του ανέμου είναι υψηλή.».

```
temperature = float(input("Εισάγετε τη θερμοκρασία (°C): "))
wind_speed = float(input("Εισάγετε την ταχύτητα ανέμου (km/h): "))
if temperature > 0:
    print("Η θερμοκρασία είναι πάνω από το μηδέν.")
    if wind_speed < 30:
        print("Οι συνθήκες είναι κατάλληλες για υπαίθρια δραστηριότητα.")
    else:
        print("Η ένταση του ανέμου είναι υψηλή.")
else:
    print("Η θερμοκρασία είναι πολύ χαμηλή.")
```

Κ. 1.4.2 – Παράδειγμα κώδικα με εμφωλευμένες if.

1.4.2. Τριαδικός τελεστής if

Ο τριαδικός τελεστής if επιτρέπει τη συγγραφή μιας απλής δομής επιλογής σε μία μόνο γραμμή, επιστρέφοντας μία τιμή ανάλογα με το αποτέλεσμα μιας συνθήκης. Η γενική μορφή του είναι:

```
<τιμή1> if <συνθήκη> else <τιμή2>
```

Αν η συνθήκη αποτιμηθεί ως True, η έκφραση επιστρέφει την <τιμή1>, διαφορετικά επιστρέφει την <τιμή2>. Ο τριαδικός τελεστής χρησιμοποιείται κυρίως όταν η επιλογή αφορά απλές τιμές και όχι ολόκληρα μπλοκ εντολών, συμβάλλοντας σε πιο συνοπτικό και αναγνώσιμο κώδικα.

Στον κώδικα Κ. 1.4.3 παρουσιάζεται ένα παράδειγμα εφαρμογής του τριαδικού τελεστή. Ο χρήστης εισάγει έναν ακέραιο και με μια γραμμή κώδικα αποδίδεται κατάλληλα η συμβολοσειρά «Άρτιος» ή «Περιττός» στη μεταβλητή result.

```
x = int(input("Εισάγετε έναν αριθμό: "))
result = "Άρτιος" if x % 2 == 0 else "Περιττός"
print(result)
```

Κ. 1.4.3 – Παράδειγμα με τον τριαδικό τελεστή if.

1.4.3. Η εντολή match

Η εντολή match, υλοποιεί το μηχανισμό της λεγόμενης δομικής αντιστοίχισης προτύπων (structural pattern matching). Επιτρέπει τον έλεγχο μιας τιμής έναντι πολλαπλών προτύπων (patterns) και την εκτέλεση του αντίστοιχου μπλοκ εντολών όταν εντοπιστεί ταύτιση. Η βασική σύνταξη περιλαμβάνει τη λέξη-κλειδί match, ακολουθούμενη από την προς εξέταση έκφραση, και διαδοχικές περιπτώσεις case, καθεμία από τις οποίες ορίζει ένα πρότυπο. Η δομή match μοιάζει το switch άλλων γλωσσών, αλλά είναι πιο ισχυρή, καθώς μπορεί να αντιστοιχίζει όχι μόνο απλές τιμές αλλά και δομές δεδομένων, όπως πλειάδες ή λίστες.

Στον κώδικα Κ. 1.4.4 παρουσιάζεται μια απλή χρήση της match όπου η μεταβλητή day συγκρίνεται διαδοχικά με τα πρότυπα που υπάρχουν δεξιά από κάθε case. Το σύμβολο | επιτρέπει την ένωση πολλαπλών τιμών στο ίδιο πρότυπο, ενώ το _ λειτουργεί ως προεπιλεγμένη περίπτωση (wildcard), αντίστοιχη της else σε μια δομή if. Συνεπώς αν ο χρήστης εισάγει την τιμή «Πέμπτη» θα εμφανιστεί το μήνυμα «Ενδιάμεση ημέρα».

```
day = input("Εισάγετε ημέρα: ")
match day:
    case "Δευτέρα":
        print("Αρχή εβδομάδας")
    case "Σάββατο" | "Κυριακή":
        print("Σαββατοκύριακο")
    case _:
        print("Ενδιάμεση ημέρα")
```

Κ. 1.4.4 – Παράδειγμα με την εντολή αντιστοίχισης προτύπων match.

1.4.4. Εντολές επανάληψης

Οι εντολές επανάληψης επιτρέπουν την εκτέλεση ενός μπλοκ κώδικα περισσότερες από μία φορές. Οι δύο δομές επανάληψης στην Python είναι η εντολή while και η εντολή for. Παράλληλα, η Python παρέχει υψηλού επιπέδου εναλλακτικές, όπως list comprehensions, γεννήτριες (generators) και

ενσωματωμένες συναρτήσεις (map(), filter(), sum() κ.ά.), οι οποίες συχνά επιτρέπουν πιο συνοπτικό και σαφή κώδικα. Επομένως, αν και οι εντολές επανάληψης αποτελούν θεμελιώδες εργαλείο, καλό είναι να εξετάζεται αν υπάρχει πιο κατάλληλη ή πιο εκφραστική λύση πριν από τη ρητή χρήση τους.

1.4.4.1 Η εντολή while

Η εντολή while χρησιμοποιείται για την επαναληπτική εκτέλεση ενός μπλοκ εντολών όσο μια λογική συνθήκη παραμένει αληθής (True). Η συνθήκη αξιολογείται πριν από κάθε επανάληψη και, όταν αποτιμηθεί ως False, η επανάληψη τερματίζεται. Η while μπορεί προαιρετικά να συνοδεύεται από το else, το μπλοκ του οποίου εκτελείται όταν η επανάληψη ολοκληρωθεί κανονικά, δηλαδή χωρίς να διακοπεί μέσω εντολής break.

Η σύνταξη της εντολής while είναι η ακόλουθη:

```
while <συνθήκη>:
    <μπλοκ εντολών>
[else:
    <μπλοκ εντολών>]
```

Στον κώδικα Κ.5 παρουσιάζεται ένα παράδειγμα χρήσης της while. Ο χρήστης εισάγει έναν θετικό ακέραιο αριθμό και το πρόγραμμα υπολογίζει το άθροισμα όλων των ακεραίων από το 1 μέχρι και την τιμή αυτή. Η μεταβλητή total λειτουργεί ως αθροιστής (accumulator), στον οποίο προστίθεται σε κάθε επανάληψη η τρέχουσα τιμή της μεταβλητής i. Η μεταβλητή i αυξάνεται κατά 1 σε κάθε βήμα, μέχρι η συνθήκη i <= n να γίνει ψευδής. Καθώς η επανάληψη ολοκληρώνεται κανονικά, εκτελείται το μπλοκ της else, το οποίο εμφανίζει το τελικό αποτέλεσμα.

```
n = int(input("Εισάγετε έναν θετικό ακέραιο: "))
i = 1
total = 0
while i <= n:
    total += i
    i += 1
else:
    print("Το άθροισμα είναι:", total)
```

Κ. 1.4.5 – Παράδειγμα με την εντολή while.

1.4.4.2 Η εντολή for

Η εντολή for χρησιμοποιείται για την επαναληπτική εκτέλεση ενός μπλοκ εντολών διατρέχοντας τα στοιχεία ενός επαναλήψιμου αντικειμένου (iterable), όπως είναι μια λίστα, μια συμβολοσειρά ή το αποτέλεσμα της συνάρτησης range(). Σε αντίθεση με τη while, η οποία βασίζεται σε λογική συνθήκη, η for διατρέχει διαδοχικά τα στοιχεία μιας ακολουθίας μέχρι να εξαντληθούν. Όπως και η while, μπορεί προαιρετικά να συνοδεύεται από else, που εκτελείται όταν η επανάληψη ολοκληρωθεί κανονικά, δηλαδή χωρίς να διακοπεί με την εντολή break.

Η σύνταξη της εντολής for είναι η ακόλουθη:

```
for <μεταβλητή> in <iterable>:
    <μπλοκ εντολών>
[else:
    <μπλοκ εντολών>]
```

Στον κώδικα Κ.6 παρουσιάζεται ένα παράδειγμα χρήσης της for. Ο χρήστης εισάγει έναν θετικό ακέραιο αριθμό και το πρόγραμμα υπολογίζει το άθροισμα όλων των ακεραίων από το 1 μέχρι και την τιμή αυτή, αξιοποιώντας τη συνάρτηση range() για την παραγωγή της αντίστοιχης ακολουθίας τιμών. Κατά την εκτέλεση του κώδικα η μεταβλητή i λαμβάνει διαδοχικά τις τιμές από το 1 έως και το n. Μετά την ολοκλήρωση της επανάληψης, εκτελείται το μπλοκ της else, το οποίο εμφανίζει το τελικό αποτέλεσμα.

```
n = int(input("Εισάγετε έναν θετικό ακέραιο: "))
total = 0
for i in range(1, n + 1):
    total += i
else:
    print("Το άθροισμα είναι:", total)
```

Κ. 1.4.6 – Παράδειγμα με την εντολη for.

Σχετικά με τη συνάρτηση range() αξίζει να σημειωθεί ότι δεν δημιουργεί λίστα, αλλά ένα αντικείμενο τύπου range, το οποίο παράγει διαδοχικές τιμές όταν χρειάζεται (lazy evaluation). Η βασική μορφή είναι range(stop), που παράγει τιμές από το 0 έως το stop - 1. Μπορεί επίσης να χρησιμοποιηθεί ως range(start, stop) ή range(start, stop, step), όπου το start είναι η αρχική τιμή, το stop το άνω όριο (μη συμπεριλαμβανόμενο) και το step το βήμα μεταβολής. Για παράδειγμα, η range(1, 6) παράγει τις τιμές 1 έως 5, ενώ η range(0, 10, 2) παράγει τους άρτιους αριθμούς από το 0 έως το 8.

1.4.4.3 Οι εντολές break και continue

Οι εντολές `break` και `continue` χρησιμοποιούνται για τον έλεγχο της ροής εκτέλεσης μέσα σε δομές επανάληψης (`while` και `for`). Η `break` διακόπτει άμεσα την εκτέλεση της επανάληψης και η ροή μεταφέρεται στην πρώτη εντολή μετά το μπλοκ της. Από την άλλη μεριά, η `continue` δεν τερματίζει την επανάληψη, αλλά παραλείπει το υπόλοιπο των εντολών της τρέχουσας επανάληψης και προχωρά στην επόμενη επανάληψη. Οι εντολές αυτές επιτρέπουν ευέλικτο έλεγχο της συμπεριφοράς των επαναλήψεων σε ορισμένες περιπτώσεις.

Στον κώδικα Κ. 1.4.7 παρουσιάζεται ένα παράδειγμα με την εντολή break και στον κώδικα Κ. 1.4.8 παρουσιάζεται ένα παράδειγμα με την εντολή continue. Στο παράδειγμα με την break η επανάληψη τερματίζεται μόλις η μεταβλητή i λάβει την τιμή 5. Οι τιμές 1 έως 4 εμφανίζονται, αλλά οι υπόλοιπες επαναλήψεις που θα εμφάνιζαν τις εναπομείνασες τιμές δεν εκτελούνται λόγω της break. Στο παράδειγμα με την continue όταν η τιμή της μεταβλητής i είναι άρτια, η continue παραλείπει την εντολή print και η επανάληψη συνεχίζεται με την επόμενη τιμή. Ως αποτέλεσμα εμφανίζονται μόνο οι περιττοί αριθμοί από το 1 έως το 5.

```
for i in range(1, 10):
    if i == 5:
        break
    print(i)
```
Κ. 1.4.7 – Παράδειγμα επανάληψης με break.

```
for i in range(1, 6):
    if i % 2 == 0:
        continue
    print(i)
```
Κ. 1.4.8 – Παράδειγμα επανάληψης με continue.

1.5. Συναρτήσεις

Οι συναρτήσεις (functions) είναι τμήματα κώδικα στα οποία έχει αποδοθεί ένα όνομα. Ορίζονται σε ένα σημείο του προγράμματος και μπορούν να καλούνται, με το όνομα που τους έχει αποδοθεί, από διαφορετικά σημεία όπου είναι ορατές (σύμφωνα με τους κανόνες εμβέλειας – scope). Η κλήση και, συνεπώς, η εκτέλεση μιας συνάρτησης γίνεται με τη χρήση του ονόματός της ακολουθούμενου από παρενθέσεις. Μια συνάρτηση μπορεί να δέχεται εισόδους (ενδεχομένως και καμία), να εκτελεί εντολές και να επιστρέφει αποτελέσματα (ενδεχομένως και κανένα) με χρήση της εντολής return.

Η γενική μορφή του ορισμού μιας συνάρτησης είναι η ακόλουθη:

```
def όνομα_συνάρτησης(λίστα_παραμέτρων):
    εντολές
    [return αποτέλεσμα]
```

Η χρήση συναρτήσεων καθιστά τη συγγραφή, την κατανόηση, την ανάγνωση και τη διόρθωση του κώδικα που επιλύει ένα πρόβλημα ευκολότερη. Αυτό οφείλεται στο ότι ένα σύνθετο πρόβλημα διασπάται σε μικρότερα υποπροβλήματα. Η διαδικασία αυτή μπορεί να συνεχιστεί έως ότου τα επιμέρους υποπροβλήματα γίνουν επαρκώς απλά ώστε να επιλυθούν άμεσα. Με αυτόν τον τρόπο καθίσταται δυνατή η συστηματική αντιμετώπιση και επίλυση ακόμη και ιδιαίτερα σύνθετων προβλημάτων. Ένα παράπλευρο όφελος είναι ότι με τη χρήση συναρτήσεων οι απαιτούμενες διορθώσεις και αλλαγές εντοπίζονται σε μικρότερα τμήματα κώδικα και αυτό διευκολύνει το έργο του προγραμματιστή.

Στον κώδικα Κ. 1.5.1 παρουσιάζονται δύο παραδείγματα απλών συναρτήσεων. Η συνάρτηση print_greeting(), δέχεται ως είσοδο μια τιμή που προορίζεται να είναι το όνομα ενός ατόμου, περιέχει στο σώμα της μια εντολή εκτύπωσης μηνύματος και δεν περιέχει εντολή return. Η συνάρτηση calculate_area() δέχεται ως είσοδο δύο τιμές που προορίζονται να είναι το μήκος και το πλάτος ενός ορθογωνίου, υπολογίζει το εμβαδόν του ορθογωνίου και το επιστρέφει. Οι κλήσεις των συναρτήσεων ακολουθούν του ορισμού τους. Για την print_greeting(), στην περίπτωση που η κλήση της βρίσκεται στο δεξί μέλος ανάθεσης σε μεταβλητή, η τιμή που θα λάβει η μεταβλητή θα είναι None, διότι δεν έχει συμπεριληφθεί ρητή εντολή return στην print_greeting() κατά τον ορισμό της.

```python
# Συνάρτηση που δεν επιστρέφει τιμή
def print_greeting(name):
    print(f"Γεια σου {name}!")


# Συνάρτηση που επιστρέφει μια τιμή
def calculate_area(length, width):
    area = length * width
    return area


# Κλήση συνάρτησης που δεν επιστρέφει τιμή
print_greeting("Μαρία")  # Γεια σου Μαρία!

# Κλήση συνάρτησης που δεν επιστρέφει τιμή και ανάθεση σε μεταβλητή
result = print_greeting("Ελένη")  # Γεια σου Ελένη!
print("Αποτέλεσμα:", result)  # Αποτέλεσμα: None

# Κλήση συνάρτησης calculate_area()
area = calculate_area(4.5, 3.2)
print(f"Το εμβαδόν είναι: {area} τ.μ.")  # Το εμβαδόν είναι: 14.4 τ.μ.
```

Κ. 1.5.1 – Ορισμοί και κλήσεις συναρτήσεων.

1.5.1. Παράμετροι και ορίσματα συναρτήσεων

Μια συνάρτηση στον ορισμό της μπορεί να δέχεται καμία, μια ή περισσότερες παραμέτρους (parameters) που ο σκοπός τους είναι να λειτουργήσουν ως είσοδοι τιμών που θα χρησιμοποιηθούν στο σώμα της συνάρτησης για την υλοποίηση της απαιτούμενης λειτουργικότητας. Αντίστοιχα κατά την κλήση μιας συνάρτησης οι τιμές που δίνονται στις παραμέτρους ονομάζονται ορίσματα (arguments).

Στο παράδειγμα του κώδικα Κ. 1.5.2η συνάρτηση celcsius_fahrenheit() μετατρέπει μια θερμοκρασία από βαθμούς Κελσίου σε βαθμούς Φάρεναιτ. Στον ορισμό της συνάρτησης, η μεταβλητή celcius είναι παράμετρος ενώ στη συνέχεια η μεταβλητή c είναι ένα όρισμα που περνά στη συνάρτηση κατά την κλήση της. Επίσης όρισμα είναι και η σταθερά 100 που περνά κατά τη δεύτερη κλήση της συνάρτησης.

```python
# Ορισμός συνάρτησης με 1 παράμετρο
def celsius_to_fahrenheit(celsius):
    fahrenheit = (celsius * 9 / 5) + 32
    return fahrenheit


c = 0.0  # Θερμοκρασία σε βαθμούς Κελσίου
# Κλήσεις συνάρτησης
temp1 = celsius_to_fahrenheit(c)  # Όρισμα: 0
temp2 = celsius_to_fahrenheit(100)  # Όρισμα: 100
print(f"0°C = {temp1}°F")  # 0°C = 32.0°F
print(f"100°C = {temp2}°F")  # 100°C = 212.0°F
```

Κ. 1.5.2 – Συνάρτηση μετατροπής θερμοκρασιών από βαθμούς Κελσίου σε βαθμούς Φάρεναιτ.

1.5.2. Επιστροφή αποτελεσμάτων από συναρτήσεις

Οι συναρτήσεις στην Python πάντα επιστρέφουν μια τιμή. Όπως ήδη αναφέρθηκε, αν μια συνάρτηση δεν επιστρέφει ρητά κάποια τιμή, με την εντολή return, τότε επιστρέφει None. Επίσης, είναι συνηθισμένο μια συνάρτηση να επιστρέφει περισσότερες από 1 τιμές. Αυτό συμβαίνει για παράδειγμα στον κώδικα Κ. 1.5.3 όπου η συνάρτηση circle_properties() επιστρέφει τρεις τιμές.

```
import math


def circle_properties(radius):
    area = math.pi * radius**2
    circumference = 2 * math.pi * radius
    diameter = 2 * radius
    return diameter, circumference, area


diam, circum, area = circle_properties(5)
print(f"Διάμετρος: {diam:.2f}") # Διάμετρος: 10.00
print(f"Περίμετρος: {circum:.2f}") # Περίμετρος: 31.42
print(f"Εμβαδόν: {area:.2f}") # Εμβαδόν: 78.54
```

Κ. 1.5.3 – Συνάρτηση που επιστρέφει 3 τιμές.

1.5.3. Ορίσματα θέσης και ονοματισμένα ορίσματα

Στην Python υπάρχουν δύο τρόποι «περάσματος» ορισμάτων κατά την κλήση μιας συνάρτησης:

- **Ορίσματα θέσης (positional arguments):** Τα ορίσματα αντιστοιχίζονται στις παραμέτρους της συνάρτησης με βάση τη σειρά εμφάνισής τους. Το πρώτο όρισμα αντιστοιχίζεται στην πρώτη παράμετρο, το δεύτερο στη δεύτερη κ.ο.κ.
- **Ορίσματα λέξεων-κλειδιών (keyword arguments), ή αλλιώς ονοματισμένα ορίσματα (named arguments):** Κάθε όρισμα δηλώνεται ρητά με το όνομα της αντίστοιχης παραμέτρου (π.χ. length=10). Στην περίπτωση αυτή, η σειρά εμφάνισης των ορισμάτων στην κλήση δεν επηρεάζει την αντιστοίχισή τους.

Οι δύο τρόποι μπορούν να συνδυαστούν στην ίδια κλήση συνάρτησης, υπό την προϋπόθεση ότι τα ορίσματα θέσης προηγούνται των ονοματισμένων ορισμάτων. Στο κώδικα Κ. 1.5.4 παρουσιάζεται ένα παράδειγμα κλήσεων συνάρτησης με ορίσματ θέσης και με ονοματισμένα ορίσματα.

```
def calculate_rectangle_area(length, width):
    return length * width


r1 = calculate_rectangle_area(10, 20)  # ορίσματα θέσης
r2 = calculate_rectangle_area(length=10, width=20)  # ονοματισμένα ορίσματα
r3 = calculate_rectangle_area(width=20, length=10)  # ονοματισμένα ορίσματα
r4 = calculate_rectangle_area(
    10, width=10
)  # συνδυασμός ορισμάτων θέσης και ονοματισμένων ορισμάτων
print(f"{r1=}, {r2=}, {r3=}, {r4=}")  # r1=200, r2=200, r3=200, r4=100
```

Κ. 1.5.4 – Ορίσματα θέσης και ονοματισμένα ορίσματα.

Αν κατά την κλήση προηγηθούν των ονοματισμένων ορισμάτων, ορίσματα θέσης, τότε προκαλείται σφάλμα “SyntaxError: positional argument follows keyword argument”. Αυτό θα συνέβαινε στο προηγούμενο παράδειγμα αν επιχειρούταν μια κλήση της μορφής: calculate_rectangle_area(length=10, 20).

1.5.4. Προαιρετικές παράμετροι

Κατά την κλήση των συναρτήσεων οι παράμετροι μπορούν είτε να είναι υποχρεωτικές να συμπληρωθούν, είτε να είναι προαιρετικές. Για κάθε προαιρετική παράμετρο ορίζεται μια προκαθορισμένη τιμή (default value) που χρησιμοποιείται αν δεν δοθεί αντίστοιχη τιμή ορίσματος.

Στο παράδειγμα του κώδικα Κ. 1.5.5 η συνάρτηση calculate_interest() υπολογίζει το ποσό των τόκων για ένα κεφάλαιο (principal), επιτόκιο (rate), έτη τοκισμού (time) και αριθμός ανατοκισμών ανά έτος (compound_frequency). Όλες οι παράμετροι, πλην της πρώτης, είναι προαιρετικές και οι προκαθορισμένες τιμές τους ορίζονται στην επικεφαλίδα της συνάρτησης. Η συνάρτηση θα πρέπει να κληθεί με όρισμα τουλάχιστον την τιμή του κεφαλαίου, ενώ αν τα άλλα ορίσματα δεν λάβουν τιμή κατά την κλήση της συνάρτησης, θα συμπληρωθούν με τις προκαθορισμένες τιμές.

```
def calculate_interest(principal, rate=0.05, time=1, compound_frequency=1):
    amount = principal * (1 + rate / compound_frequency) **
(compound_frequency * time)
    interest = amount - principal
    return round(interest, 2)


# default τιμές: επιτόκιο 5%, 1 έτος, ετήσια εφαρμογή επιτοκίου
interest1 = calculate_interest(1000)
print(f"Τόκοι: {interest1}€")  # Τόκοι: 50.0€

# Προσαρμοσμένες τιμές: επιτόκιο 8%, 2 έτη, ετήσια εφαρμογή επιτοκίου
interest2 = calculate_interest(1000, rate=0.08, time=2)
print(f"Τόκοι: {interest2}€")  # Τόκοι: 166.4€
```

Κ. 1.5.5 – Συνάρτηση με προκαθορισμένες τιμές ορισμάτων.

1.5.5. Συναρτήσεις με μεταβλητό πλήθος ορισμάτων

Στην Python η δημιουργία συναρτήσεων με μεταβλητό πλήθος ορισμάτων είναι εύκολη υπόθεση. Τα ορίσματα μεταβλητού πλήθους μπορεί να είναι είτε ορίσματα θέσης (args) είτε ονοματισμένα ορίσματα (kwargs). Στην πρώτη περίπτωση χρησιμοποιείται ο τελεστής *, ενώ στη δεύτερη ο τελεστής **. Θα πρέπει να σημειωθεί ότι τα συνηθισμένα ονόματα για τα μεταβλητού πλήθους ορίσματα θέσης είναι args, ενώ για τα μεταβλητού πλήθους ονοματισμένα ορίσματα είναι kwargs. Ωστόσο, τα ονόματα args και kwargs είναι απλές συμβάσεις και μπορούν να χρησιμοποιηθούν άλλα ονόματα στη θέση τους.

1.5.5.1 Ορίσματα τύπου *args

Η χρήση *args επιτρέπει την ευέλικτη χρήση ορισμάτων θέσης που το πλήθος τους δεν έχει προκαθοριστεί στον ορισμό της συνάρτησης. Για παράδειγμα στο κώδικα Κ. 1.5.6, η συνάρτηση my_function() μπορεί να κληθεί με οποιοδήποτε πλήθος ορισμάτων, όπως φαίνεται και στις κλήσεις που ακολουθούν τον ορισμό της.

```
def my_function(*args):
    print(f"Ελήφθησαν {len(args)} ορίσματα")
    for i, arg in enumerate(args):
        print(f"Όρισμα {i}: {arg}", end=" ")
    print("\n")


my_function()  # 0 ορίσματα
my_function(1)  # 1 όρισμα
my_function(1, 2, 3)  # 3 ορίσματα
my_function("a", "b", "c", "d")  # 4 ορίσματα
```

*Κ. 1.5.6 – Συνάρτηση με όρισμα τύπου *args.*

Η εκτέλεση του προγράμματος θα παράξει την έξοδο Ε. 1.

```
Ελήφθησαν 0 ορίσματα


Ελήφθησαν 1 ορίσματα
Όρισμα 0: 1

Ελήφθησαν 3 ορίσματα
Όρισμα 0: 1 Όρισμα 1: 2 Όρισμα 2: 3

Ελήφθησαν 4 ορίσματα
Όρισμα 0: a Όρισμα 1: b Όρισμα 2: c Όρισμα 3: d
```

Ε. 1 – Έξοδος κλήσεων συνάρτησης που δέχεται μεταβλητό πλήθος ορισμάτων.

1.5.5.2 Ορίσματα τύπου **kwargs

Η χρήση **kwargs επιτρέπει την ευέλικτη χρήση ονοματισμένων ορισμάτων που δεν έχουν προκαθοριστεί στον ορισμό της συνάρτησης. Το kwargs είναι ένα λεξικό και ένα λεξικό αποτελείται από ζεύγη της μορφή «κλειδί: τιμή». Στο παράδειγμα του κώδικα Κ. 1.5.7 η συνάρτηση my_function() μπορεί να κληθεί με κανένα, ένα ή περισσότερα ονοματισμένα ορίσματα που θα αποτελέσουν ζεύγη «κλειδί: τιμή» στο kwargs.

```
def my_function(**kwargs):
    print(f"Ελήφθησαν {len(kwargs)} ονοματισμένα ορίσματα")
    for key, value in kwargs.items():
        print(f"{key}: {value}")


my_function()  # 0 ορίσματα
my_function(name="Νίκος")  # 1 ονοματισμένο όρισμα
my_function(name="Μαρία", age=30)  # 2 ονοματισμένα ορίσματα
my_function(city="Ιωάννινα", country="GR", population=115000)  # 3 ον. ορ.
```

*Κ. 1.5.7 – Συνάρτηση με ορίσματα τύπου **kwargs.*

Η έξοδος που θα παραχθεί κατά την εκτέλεση θα είναι η ακόλουθη:

```
Ελήφθησαν 0 ονοματισμένα ορίσματα
Ελήφθησαν 1 ονοματισμένα ορίσματα
name: Νίκος
Ελήφθησαν 2 ονοματισμένα ορίσματα
name: Μαρία
age: 30
Ελήφθησαν 3 ονοματισμένα ορίσματα
city: Ιωάννινα
country: GR
population: 115000
```

Ε. 2 – Έξοδος εκτέλεσης του κώδικα Κ. 1.5.7.

*1.5.5.3 Κανόνες για συνδυασμό args, kwargs, *args και **kwargs*

Υπάρχουν κανόνες που ο διερμηνευτής της Python ελέγχει ότι τηρούνται, όταν συνδυάζονται ορίσματα θέσης, ονοματισμένα ορίσματα, *args και **kwargs που είναι οι ακόλουθοι:

Κατά τον ορισμό συναρτήσεων:

- Οι παράμετροι **kwargs, αν υπάρχουν, πρέπει πάντα να είναι τελευταίοι.

Κατά την κλήση συναρτήσεων:

- Τα ορισματα θέσης (args) πρέπει να βρίσκονται πριν τα ονοματισμένα ορίσματα (kwargs).
- Σε συναρτήσεις με παραμέτρους μετά το *args, τα ορίσματα που τους δίνουν τιμές πρέπει να είναι kwargs.

Ο κώδικας Κ. 1.5.8 επιδεικνύει τους παραπάνω κανόνες, έχοντας τοποθετήσει σε σχόλια τις περιπτώσεις που ο διερμηνευτής θα εντόπιζε ότι κάποιος κανόνας θα παραβιάζονταν.

```
# def f1(**kwargs, x): # SyntaxError: invalid syntax
#   print(f"{kwargs=}, {x=}")


def f2(x, y):
    print(f"{x=}, {y=}")


f2(1, 2)  # x=1, y=2
f2(x=1, y=2)  # x=1, y=2
f2(1, y=2)  # x=1, y=2
# f2(1, x=2) # TypeError: f2() got multiple values for argument 'x'
# f2(x=1, 2) # SyntaxError: positional argument follows keyword argument


def f3(x, *args, y, **kwargs):
    print(f"{x=}, {args=}, {y=}, {kwargs=}")


f3(1, 2, 3, y=4, z=5, w=6)  # x=1, args=(2, 3), y=4,
                            # kwargs={'z': 5, 'w': 6}
# f3(1, 2, 3, 4, z=5, w=6)  # TypeError: f3() missing 1 required
```

```
                                    # keyword-only argument: 'y'
```

*Κ. 1.5.8 – Συναρτήσεις με συνδυασμό ορισμάτων θέσης, *args, **kwargs.*

1.5.6. Συμβολοσειρές τεκμηρίωσης συνάρτησης

Οι συμβολοσειρές τεκμηρίωσης (docstrings) είναι συμβολοσειρές, συνήθως πολλών γραμμών, οι οποίες περικλείονται σε τριπλά εισαγωγικά και τοποθετούνται στην αρχή του σώματος μιας συνάρτησης, αμέσως μετά τη γραμμή ορισμού της, με σκοπό την περιγραφή της λειτουργικότητάς της. Η χρήση docstring δεν είναι υποχρεωτική, ωστόσο θεωρείται καλή πρακτική τεκμηρίωσης. Εφόσον μια συνάρτηση περιλαμβάνει docstring, η έκφραση <όνομα_συνάρτησης>.__doc__ επιστρέφει το κείμενο της τεκμηρίωσής της. Το __doc__ αποτελεί παράδειγμα “dunder” (double underscore), δηλαδή ειδικού αναγνωριστικού που αρχίζει και τελειώνει με δύο κάτω παύλες. Επιπλέον, η κλήση της συνάρτησης help() με όρισμα το όνομα της συνάρτησης εμφανίζει το docstring στο ενσωματωμένο σύστημα βοήθειας της Python. Η έξοδος από το περιβάλλον βοήθειας γίνεται με την πίεση του πλήκτρου q, αν το περιεχόμενο της βοήθειας υπερβαίνει την μια οθόνη σε μήκος. Στο ακόλουθο απόσπασμα κώδικα φαίνεται το περιεχόμενο του docstring για τη συνάρτηση sqrt() από το module math της τυπικής βιβλιοθήκης, όπως μπορεί να ληφθεί από το REPL (Κ. 1.5.9).

```
>>> import math
>>> math.sqrt.__doc__
'Return the square root of x.'
>>> help(math.sqrt)
Help on built-in function sqrt in module math:

sqrt(x, /)
    Return the square root of x.
```

Κ. 1.5.9 – Εμφάνιση docstrings συναρτήσεων.

Στο παράδειγμα του κώδικα Κ. 1.5.10 η συνάρτηση calculate_area() διαθέτει docstring που εκτυπώνεται με την εντολή print(calculate_area.__doc__).

```
import math


def calculate_area(radius):
    """
    Υπολογισμός του εμβαδού ενός κύκλου δεδομένης της ακτίνας του.

    Parameters:
    radius (float): Η ακτίνα του κύκλου.

    Returns:
    float: Το εμβαδόν του κύκλου.
    """
    return math.pi * radius**2


print(calculate_area.__doc__)
```

Κ. 1.5.10 – Ορισμός docstring συνάρτησης και εμφάνιση του.

Κατά την εκτέλεση του συγκεκριμένου κώδικα, η συνάρτηση calculate_area() δεν καλείται, αλλά εμφανίζεται η έξοδος Ε. 3, που είναι το περιεχόμενο της μεταβλητής __doc__, όπως έχει οριστεί για τη συνάρτηση.

```
    Υπολογισμός του εμβαδού ενός κύκλου δεδομένης της ακτίνας του.

    Parameters:
    radius (float): Η ακτίνα του κύκλου.

    Returns:
    float: Το εμβαδόν του κύκλου.
```

Ε. 3 – Εμφάνιση του docstring για τη συναάρτηση calculate_area().

1.5.7. Η εντολή pass

Κατά τη συγγραφή ενός προγράμματος μπορεί να ανιχνευθεί η ανάγκη ύπαρξης μιας συνάρτησης αλλά να μην επιλεγεί η συγγραφή του σώματός της στη συγκεκριμένη στιγμή. Τότε, προκειμένου να διατηρηθεί η συντακτική ορθότητα του κώδικα, γράφεται κανονικά η πρώτη γραμμή (επικεφαλίδα) της συνάρτησης, και ακολουθείται από την λέξη pass ή τρεις τελείες ..., ακριβώς για να υποδηλωθεί ότι το σώμα της συνάρτησης θα συμπληρωθεί αργότερα. Η εντολή pass είναι χρήσιμη για τη συγγραφή του «σκελετού» ενός κώδικα. Επίσης, το pass μπορεί να χρησιμοποιηθεί στη θέση ενός μπλοκ κώδικα στις if, for, while που πρόκειται να υλοποιηθεί αργότερα.

Ο κώδικας Κ. 1.5.11, περιέχει παραδείγματα χρήσης του pass.

```
def foo():
    pass


def bar(a_parameter, another_parameter):
    ...


foo()
bar(1, 2)
age = 26
if age >= 18:
    pass  # Θα υλοποιηθεί αργότερα
else:
    print("Ανήλικος")
```

Κ. 1.5.11 – Συναρτήσεις με κενό μπλοκ κώδικα, με ... και με pass.

1.5.8. Συναρτήσεις μέσα σε συναρτήσεις

Η Python παρέχει τη δυνατότητα ορισμού μιας συνάρτησης μέσα στο σώμα μιας άλλης συνάρτησης. Σε αυτή την περίπτωση, η συνάρτηση που ορίζεται στο εσωτερικό μιας άλλης ονομάζεται εμφωλευμένη (nested function) ή εσωτερική συνάρτηση (inner function). Συνήθως χρησιμοποιείται ως βοηθητικός μηχανισμός, όταν επιθυμούμε ο συγκεκριμένος κώδικας να μην είναι ορατός εκτός της περικλείουσας (εξωτερικής) συνάρτησης, αλλά να αξιοποιείται αποκλειστικά από αυτή. Οι

εσωτερικές συναρτήσεις έχουν πρόσβαση σε μεταβλητές της εξωτερικής εμβέλειας, γεγονός που επιτρέπει τη συνεργασία τους με το περιβάλλον στο οποίο ορίζονται. Επιπλέον, η εσωτερική συνάρτηση δημιουργείται εκ νέου κάθε φορά που καλείται η εξωτερική συνάρτηση. Η συμπεριφορά αυτή συνδέεται με μια προχωρημένη προγραμματιστική έννοια, γνωστή ως κλειστότητα (closure), όπου μια συνάρτηση «θυμάται» το περιβάλλον μέσα στο οποίο δημιουργήθηκε.

Στο παράδειγμα του κώδικα Κ. 1.5.12 η συνάρτηση total_price() περιέχει μια άλλη συνάρτηση, τη συνάρτηση tax() που χρησιμοποιεί στους υπολογισμούς που πραγματοποιεί. Η συνάρτηση tax() είναι «κρυμμένη» από το υπόλοιπο πρόγραμμα και πρόσβαση σε αυτή δίνεται μόνο εντός της συνάρτησης total_price().

```
def total_price(base_price, tax_rate):
    # εσωτερική συνάρτηση: υπολογίζει το φόρο
    def tax(amount):
        return amount * tax_rate

    # κλήση εσωτερικής συνάρτησης
    tax_amount = tax(base_price)
    return base_price + tax_amount


print(total_price(100, 0.24))  # 124.0
print(total_price(50, 0.10))  # 55.0
```

Κ. 1.5.12 – Ορισμός συνάρτησης μέσα σε συνάρτηση.

1.5.9. Εμβέλεια μεταβλητών

Ο όρος εμβέλεια (scope) μεταβλητής αναφέρεται στο τμήμα του προγράμματος στο οποίο μια μεταβλητή είναι ορατή και μπορεί να προσπελαστεί με το όνομά της. Στην Python υποστηρίζεται η τοπική εμβέλεια, η καθολική εμβέλεια και η μη-τοπική εμβέλεια.

1.5.9.1 Τοπικές μεταβλητές

Οι μεταβλητές που δημιουργούνται μέσα σε μια συνάρτηση έχουν τοπική εμβέλεια, ονομάζονται τοπικές μεταβλητές και μπορούν να χρησιμοποιηθούν μόνο μέσα στη συνάρτηση. Αν γίνει απόπειρα αναφοράς σε μια τοπική μεταβλητή εκτός της συνάρτησης, τότε θα προκληθεί εξαίρεση NameError και η εκτέλεση του προγράμματος θα διακοπεί, εμφανίζοντας μήνυμα σφάλματος ότι το συγκεκριμένο όνομα μεταβλητής δεν έχει οριστεί (not defined).

Στον κώδικα Κ. 1.5.13 παρουσιάζεται ένα παράδειγμα όπου η μεταβλητή message είναι τοπική για τη συνάρτηση greet(). Αυτό σημαίνει ότι μπορεί να χρησιμοποιηθεί εντός της συνάρτησης, αλλά οποιαδήποτε απόπειρα αναφοράς της εκτός της συνάρτησης θα επιστρέψει μήνυμα σφάλματος και θα τερματίσει την εκτέλεση του προγράμματος.

```
def greet():
    message = "Hello, world!"  # τοπική μεταβλητή
```

```
    print(message)  # τοπική μεταβλητή


greet()  # ok
print(message)  # Error: NameError: name 'message' is not defined
```
Κ. 1.5.13 – Παράδειγμα με τοπική μεταβλητή.

1.5.9.2 Καθολικές μεταβλητές

Οι καθολικές μεταβλητές (global variables) δηλώνονται εκτός συναρτήσεων και είναι, καταρχήν, προσπελάσιμες από οποιοδήποτε σημείο του προγράμματος, τόσο εντός όσο και εκτός συναρτήσεων. Ωστόσο, αν επιχειρηθεί τροποποίηση της τιμής μιας καθολικής μεταβλητής μέσα σε μια συνάρτηση, τότε η μεταβλητή πρέπει να δηλωθεί ρητά ως global μέσα στο σώμα της συνάρτησης. Γενικά, μέσα σε μια συνάρτηση οι καθολικές μεταβλητές μπορούν να αναγνωστούν, αλλά δεν μπορούν να τροποποιηθούν, εκτός αν δηλωθούν ως global.

Στο παράδειγμα του κώδικα Κ. 1.5.14, η exchange_rate είναι καθολική μεταβλητή και είναι ορατή τόσο στη συνάρτηση convert_to_usd() όσο και στη συνάρτηση update_rate(). Ωστόσο, για να μπορεί η συνάρτηση update_rate() να τροποποιεί την τιμή της exchange_rate, θα πρέπει να υπάρχει η δήλωση μέσα στη συνάρτηση ότι η μεταβλητή exchange_rate είναι global. Αν δεν υπήρχε αυτή η δήλωση τότε η exchange_rate εντός της συνάρτησης update_rate() θα ήταν μια νέα τοπική μεταβλητή και οποιαδήποτε ανάθεση τιμής σε αυτή δεν θα επηρέαζε την καθολική μεταβλητή exchange_rate.

```
# καθολική μεταβλητή
exchange_rate = 1.15  # EUR -> USD


def convert_to_usd(amount_eur):
    return amount_eur * exchange_rate


def update_rate(new_rate):
    global exchange_rate  # πρέπει να υπάρχει
    exchange_rate = new_rate


print(f"{convert_to_usd(50):.2f}") # 57.50
update_rate(1.2)
print(f"{convert_to_usd(50):.2f}") # 60.00
```
Κ. 1.5.14 – Παράδειγμα με καθολική μεταβλητή.

1.5.9.3 Μη-τοπικές μεταβλητές

Οι μη τοπικές (nonlocal) μεταβλητές εμφανίζονται στην περίπτωση συναρτήσεων που περιέχουν εσωτερικές (εμφωλευμένες) συναρτήσεις. Όταν μια μεταβλητή μέσα σε μια εσωτερική συνάρτηση δηλωθεί με τη λέξη-κλειδί nonlocal, τότε αναφέρεται σε μεταβλητή με το ίδιο όνομα που έχει οριστεί στην πλησιέστερη περικλείουσα συνάρτηση (και όχι στο καθολικό επίπεδο). Στην περίπτωση αυτή, η εσωτερική συνάρτηση μπορεί όχι μόνο να αναγνώσει αλλά και να τροποποιήσει την τιμή της

μεταβλητής αυτής. Οι μη τοπικές μεταβλητές είναι χρήσιμες όταν μια μεταβλητή που ορίζεται σε μια περικλείουσα συνάρτηση είναι επιθυμητό να ενημερώνεται από τον κώδικα της εσωτερικής συνάρτησης.

Στο παράδειγμα του κώδικα XXX ορίζεται η τοπική μεταβλητή total της συνάρτησης shopping_cart(). Προκειμένου η μεταβλητή total να μπορεί να ενημερώνεται από την εσωτερική συνάρτηση add_items(), δηλώνεται ως nonlocal.

```
def shopping_cart():
    total = 0  # εξωτερική μεταβλητή για την add_items()

    def add_items(prices):
        nonlocal total  # επιτρέπει τροποποίηση της total
        for price in prices:
            total += price
            print(f"+ {price}, νέο σύνολο {total}")

    add_items([10, 25, 5])
    print(f"Τελικό σύνολο: {total}")


shopping_cart()
```

Κ. 1.5.15 – Παραδειγμα με μη-τοπική μεταβλητή.

Η έξοδος Ε. 4 δείχνει το αποτέλεσμα της εκτέλεσης.

```
+ 10, νέο σύνολο 10
+ 25, νέο σύνολο 35
+ 5, νέο σύνολο 40
Τελικό σύνολο: 40
```

Ε. 4 – Η έξοδος από την εκτέλεση του κώδικα Κ. 1.5.15

1.5.10. Λάμδα συναρτήσεις

Μια λάμδα συνάρτηση ή ανώνυμη συνάρτηση είναι μια συνάρτηση στην οποία δεν αποδίδεται όνομα που ορίζεται με τη λέξη-κλειδί lambda, με την ακόλουθη σύνταξη:

```
lambda ορίσματα: έκφραση
```

Η λάμδα συνάρτηση μπορεί να δέχεται κανένα, ένα ή περισσότερα ορίσματα και επιστρέφει ως αποτέλεσμα την τιμή της έκφρασης που ακολουθεί την άνω και κάτω τελεία. Το σώμα της περιορίζεται αποκλειστικά σε μία έκφραση (και όχι σε πολλαπλές εντολές). Συνεπώς, οι λάμδα συναρτήσεις προορίζονται για σύντομες και απλές λειτουργίες. Όταν η απαιτούμενη λογική είναι πιο σύνθετη, είναι προτιμότερο να χρησιμοποιείται μια κανονική συνάρτηση που ορίζεται με def, ώστε ο κώδικας να παραμένει ευανάγνωστος και σαφής.

Στον κώδικα Κ. 1.5.16, ορίζεται η παραδοσιακή συνάρτηση square() που επιστρέφει το τετράγωνο του ορίσματος της καθώς και μια ισοδύναμη λάμδα συνάρτηση, δύο φορές. Στην πρώτη φορά η λάμδα συνάρτηση καλείται με όρισμα την τιμή 5 και το αποτέλεσμα ανατίθεται στη μεταβλητή a, ενώ

τη δεύτερη φορά η λάμδα συνάρτηση ανατίθεται στη μεταβλητή f και στη συνέχεια η κλήση της με όρισμα την τιμή 5 γίνεται απλά γράφοντας f(5).

```
def square(x):
    return x**2


print(square(5))  # 25
a = (lambda x: x**2)(5)
print(a)  # 25
f = lambda x: x**2
print(f(5))  # 25
```

Κ. 1.5.16 – Παράδειγμα με λάμδα συναρτήσεις.

Οι λάμδα συναρτήσεις χρησιμοποιούνται συχνά ως ορίσματα σε κλήσεις άλλων συναρτήσεων, ιδιαίτερα όταν απαιτείται η παροχή μιας σύντομης, προσωρινής συνάρτησης χωρίς την ανάγκη ορισμού της με def. Αυτό είναι συνηθισμένο να συμβαίνει σε συναρτήσεις ανώτερης τάξης (higher-order functions), δηλαδή συναρτήσεις που δέχονται άλλες συναρτήσεις ως παραμέτρους, όπως οι sorted(), min(), max() με όρισμα key. Σε τέτοιες περιπτώσεις, η χρήση λάμδα συναρτήσεων επιτρέπει τον επιτόπου ορισμό της λογικής που θα εφαρμοστεί, καθιστώντας τον κώδικα πιο συνοπτικό και εκφραστικό. Για παράδειγμα, η sorted(alist, key=lambda x: x[1]) ταξινομεί τη λίστα alist με βάση το δεύτερο στοιχείο κάθε υπο-δομής, χωρίς να απαιτείται ξεχωριστός ορισμός βοηθητικής συνάρτησης, όπως φαίνεται στο ακόλουθο παράδειγμα στο REPL (Κ. 1.5.17).

```
>>> students = [("Μαρία", 8.5), ("Νίκος", 9.2), ("Ελένη", 7.8)]
>>> sorted(students)
[('Ελένη', 7.8), ('Μαρία', 8.5), ('Νίκος', 9.2)]

>>> sorted(students, key=lambda x: x[1])
[('Ελένη', 7.8), ('Μαρία', 8.5), ('Νίκος', 9.2)]
```

Κ. 1.5.17 – Παράδειγμα ταξινόμησης με όρισμα της συνάρτησης sorted(), μια λάμδα συνάρτηση.

Μια ακόμη συνηθισμένη χρήση των λάμδα συναρτήσεων είναι στη διαδοχική εφαρμογή των map(), filter() και reduce(). Αξίζει να σημειωθεί ότι οι map() και filter() είναι ενσωματωμένες (built-in) συναρτήσεις της Python, ενώ η reduce() ορίζεται στο module functools της τυπικής βιβλιοθήκης και απαιτεί ρητή εισαγωγή (import) πριν από τη χρήση της. Στον κώδικα Κ. 1.5.18 παρουσιάζεται ένα παράδειγμα σταδιακής εφαρμογής των map(), filter() και reduce(): η map() μετασχηματίζει μια λίστα τιμών σε μια νέα λίστα, η filter() επιλέγει ορισμένα από τα στοιχεία της και η reduce() συνδυάζει τα αποτελέσματα σε μία τελική τιμή. Στο παράδειγμα, το τελικό αποτέλεσμα είναι ο υπολογισμός του αθροίσματος των άρτιων τετραγώνων της λίστας nums. Οι λειτουργίες αυτές μπορούν, όπως φαίνεται και στην τελευταία εντολή του προγράμματος, να συνδυαστούν σε μία ενιαία έκφραση.

```
from functools import reduce


nums = [1, 2, 3, 4, 5, 6]
```

```python
# map: υπολογισμός τετραγώνου κάθε τιμής
squared = list(map(lambda x: x**2, nums))
print("Τετράγωνα:", squared)  # Τετράγωνα: [1, 4, 9, 16, 25, 36]

# filter: διατήρηση μόνο των περιττών τιμών
evens = list(filter(lambda x: x % 2 == 0, squared))
print("Άρτιοι αριθμοί:", evens)  # Άρτιοι αριθμοί: [4, 16, 36]

# reduce: άθροισμα τιμών
total = reduce(lambda x, y: x + y, evens)
print("Σύνολο:", total)  # Σύνολο: 56


print(
    "Σύνολο:",
    reduce(lambda x, y: x + y, filter(lambda x: x % 2 == 0,
                                      map(lambda x: x**2, nums))),
) # Σύνολο: 56
```

Κ. 1.5.18 – Παράδειγμα map, filter, reduce.

1.6. Δομές δεδομένων

Η Python παρέχει ενσωματωμένες βασικές δομές δεδομένων που επιτρέπουν την αποδοτική οργάνωση και διαχείριση συλλογών τιμών. Οι κυριότερες είναι η λίστα (list), η πλειάδα (tuple), το λεξικό (dict) και το σύνολο (set). Επιπλέον, η Python διαθέτει και άλλες δομές, όπως το frozenset, το οποίο αποτελεί μια μη-μεταλλάξιμη εκδοχή του συνόλου, καθώς και εξειδικευμένους τύπους από την τυπική βιβλιοθήκη (π.χ. collections.deque, defaultdict), που επεκτείνουν τις βασικές λειτουργικότητες ανάλογα με τις ανάγκες της εφαρμογής.

1.6.1. Λίστες

Οι λίστες (lists) είναι δομές δεδομένων που μπορούν να περιέχουν ετερογενή στοιχεία, τα οποία διαχωρίζονται με κόμματα και περικλείονται σε αγκύλες [ ]. Αποτελούν τροποποιήσημες ή αλλιώς μεταλλάξιμες (mutable) δομές, καθώς μπορούν να τροποποιούνται κατά την εκτέλεση του προγράμματος, είτε με προσθήκη ή αφαίρεση στοιχείων είτε με αντικατάσταση υπαρχόντων τιμών. Οι λίστες ανήκουν στις ακολουθίες (sequences), συνεπώς τα στοιχεία τους είναι διατεταγμένα (ordered) και η πρόσβαση σε αυτά γίνεται μέσω της θέσης τους (δείκτη), με αρίθμηση που ξεκινά από το 0.

Στο ακόλουθο παράδειγμα στο REPL (Κ. 1.6.1) ορίζεται μια λίστα με ετερογενή στοιχεία. Στη συνέχεια πραγματοποιείται προσπέλαση του 2ου στοιχείου και αντικατάσταση του 3ου στοιχείου της λίστας.

```
>>> a_list = [1, 3.14, "Java", True]
>>> a_list[1]
3.14

>>> a_list[2] = "Python"
>>> a_list
```

```
[1, 3.14, 'Python', True]
```

Κ. 1.6.1 – Δημιουργία λίστας και τροποποίηση ενός στοιχείου της.

1.6.1.1 Τεμαχισμός λιστών

Οι λίστες μπορούν να τεμαχίζονται όπως και οι συμβολοσειρές χρησιμοποιώντας σύνταξη της μορφής [start:end:step] στο τέλος της λίστας. Ο δείκτης start δηλώνει τη θέση από την οποία αρχίζει ο τεμαχισμός (συμπεριλαμβάνεται), ο end τη θέση στην οποία σταματά (δεν συμπεριλαμβάνεται), ενώ το step καθορίζει το βήμα μετακίνησης. Αν κάποια από τα τρία μέρη παραλειφθούν, χρησιμοποιούνται προεπιλεγμένες τιμές (αρχή λίστας, τέλος λίστας και βήμα 1 αντίστοιχα). Ο τεμαχισμός δημιουργεί νέα λίστα χωρίς να τροποποιεί την αρχική.

Ακολουθούν μερικά παραδείγματα στο REPL (Κ. 1.6.2) που επιδεικνύουν δυνατότητες που δίνονται με τον τεμαχισμό λιστών.

```
>>> a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> a[2:6]
[2, 3, 4, 5]            # στοιχεία από θέση 2 έως 5

>>> a[:4]
[0, 1, 2, 3]            # από την αρχή έως θέση 3

>>> a[5:]
[5, 6, 7, 8, 9]         # από θέση 5 έως το τέλος

>>> a[::2]
[0, 2, 4, 6, 8]         # κάθε δεύτερο στοιχείο

>>> a[::-1]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]  # αντιστροφή λίστας
```

Κ. 1.6.2 – Παραδείγματα τεμαχισμού λίστας.

1.6.1.2 Τελεστές που εφαρμόζονται σε λίστες και η εντολή del

Οι τελεστές που μπορούν να εφαρμοστούν σε λίστες είναι οι: +, *, in, not in, ==, !=, <, <=, >, >=, =, +=, *=, [ ], [:]. Ο τελεστής + προκαλεί συνένωση δύο λιστών σε μία νέα λίστα που περιέχει τα στοιχεία και των δύο. Ο τελεστής * δημιουργεί μια νέα λίστα που περιέχει επαναλήψεις των στοιχείων μιας λίστας όσες φορές ορίζει ένας ακέραιος πολλαπλασιαστής. Οι τελεστές in και not in χρησιμοποιούνται για τον έλεγχο συμμετοχής ενός στοιχείου σε μια λίστα, επιστρέφοντας λογική τιμή (True ή False). Οι τελεστές σύγκρισης == και != ελέγχουν αν δύο λίστες είναι ίσες ή διαφορετικές συγκρίνοντας τα στοιχεία τους με βάση τη σειρά εμφάνισης, ενώ οι <, <=, > και >= πραγματοποιούν λεξικογραφική σύγκριση στοιχείο προς στοιχείο. Ο τελεστής = αναθέτει μια λίστα σε μια μεταβλητή, δημιουργώντας νέα αναφορά στο ίδιο αντικείμενο. Οι τελεστές επαυξημένης ανάθεσης += και *= τροποποιούν τη λίστα επιτόπου (in place), επεκτείνοντάς τη με νέα στοιχεία ή επαναλαμβάνοντας το περιεχόμενό της αντίστοιχα. Τα σύμβολα [ ] χρησιμοποιούνται για πρόσβαση σε στοιχείο μέσω δείκτη, ενώ η σύνταξη

[:] (τεμαχισμός, slicing) επιτρέπει τη δημιουργία υπολίστας ή αντιγράφου μέρους ή ολόκληρης της λίστας.

Επίσης, υπάρχει και η εντολή del που χρησιμοποιείται για τη διαγραφή στοιχείου ή τμήματος λίστας με βάση μια θέση ή ένα εύρος θέσεων. Παραδείγματα, σε περιβάλλον REPL (Κ. 1.6.3), των παραπάνω τελεστών και της εντολής del ακολουθούν στη συνέχεια.

```
>>> a = [1, 2, 3]; b = [4, 5]
>>> a + b
[1, 2, 3, 4, 5]          # συνένωση

>>> a * 2
[1, 2, 3, 1, 2, 3]       # επανάληψη

>>> 2 in a, 10 not in a
(True, True)             # έλεγχος συμμετοχής

>>> a == [1,2,3], a != b
(True, True)             # ισότητα / ανισότητα

>>> [1,2] < [1,3]
True                     # λεξικογραφική σύγκριση

>>> c = a; c is a
True                     # ανάθεση: ίδια αναφορά

>>> a += [4]; a
[1, 2, 3, 4]             # επέκταση επιτόπου

>>> a *= 2; a
[1, 2, 3, 4, 1, 2, 3, 4] # επανάληψη επιτόπου

>>> a[0], a[1:4], a[:]
(1, [2, 3, 4], [1, 2, 3, 4, 1, 2, 3, 4])  # δείκτης / τεμαχισμός

>>> del a[0]; a
[2, 3, 4, 1, 2, 3, 4]    # διαγραφή στοιχείου

>>> del a[1:3]; a
[2, 1, 2, 3, 4]          # διαγραφή στοιχείων
```

Κ. 1.6.3 – Παραδείγματα εφαρμογής τελεστών σε λίστες.

1.6.1.3 Μέθοδοι λιστών

Οι λίστες στην Python διαθέτουν πληθώρα ενσωματωμένων μεθόδων που επιτρέπουν την τροποποίησή τους επιτόπου (in place). Η append(x) προσθέτει ένα νέο στοιχείο x στο τέλος της λίστας, ενώ η insert(i, x) εισάγει το στοιχείο x σε συγκεκριμένη θέση i. Η extend(b) επεκτείνει τη λίστα προσθέτοντας στο τέλος της όλα τα στοιχεία μιας άλλης λίστας b. Για αφαίρεση στοιχείων χρησιμοποιείται η remove(x), που διαγράφει την πρώτη εμφάνιση του στοιχείου x, και η pop([i]), που αφαιρεί και επιστρέφει το στοιχείο στη θέση i (ή το τελευταίο αν δεν δοθεί θέση). Η clear() διαγράφει όλα τα στοιχεία της λίστας. Για αναδιάταξη στοιχείων διατίθενται οι sort() για ταξινόμηση (με προαιρετικά ορίσματα key και reverse) και reverse() για αντιστροφή της σειράς των στοιχείων της

λίστας. Επιπλέον, η index(x) επιστρέφει τη θέση της πρώτης εμφάνισης του στοιχείου x, η count(x) μετρά πόσες φορές εμφανίζεται ένα στοιχείο και η copy() δημιουργεί ένα ρηχό αντίγραφο (shallow copy) της λίστας. Οι μέθοδοι αυτές καλύπτουν τις περισσότερες ανάγκες δυναμικής διαχείρισης λιστών στην πράξη.

Ακολουθούν μερικά παραδείγματα () στο REPL χρήσης των παραπάνω μεθόδων.

```
>>> a = [3, 1, 4]

>>> a.append(5); a
[3, 1, 4, 5]                    # προσθήκη στοιχείου στο τέλος

>>> a.insert(1, 10); a
[3, 10, 1, 4, 5]                # εισαγωγή στοιχείου στη θέση 1

>>> a.extend([7, 8]); a
[3, 10, 1, 4, 5, 7, 8]          # επέκταση με άλλη λίστα

>>> a.remove(10); a
[3, 1, 4, 5, 7, 8]              # διαγραφή πρώτης εμφάνισης στοιχείου

>>> a.pop(); a
(8, [3, 1, 4, 5, 7])            # αφαίρεση και επιστροφή τελευταίου στοιχείου

>>> a.pop(1); a
(1, [3, 4, 5, 7])               # αφαίρεση και επιστροφή στοιχείου στη θέση 1

>>> a.count(3)
1                               # πλήθος εμφανίσεων στοιχείου

>>> a.index(5)
2                               # θέση πρώτης εμφάνισης στοιχείου

>>> b = a.copy(); b is a
False                           # δημιουργία επιφανειακού αντιγράφου

>>> a.sort(); a
[3, 4, 5, 7]                    # ταξινόμηση αύξουσα

>>> a.sort(reverse=True); a
[7, 5, 4, 3]                    # ταξινόμηση φθίνουσα

>>> a.reverse(); a
[3, 4, 5, 7]                    # αντιστροφή σειράς στοιχείων

>>> a.clear(); a
[]                              # διαγραφή όλων των στοιχείων
```

Κ. 1.6.4 – Παραδείγματα διαφόρων μεθόδων που εφαρμόζονται σε λίστες.

1.6.1.4 Διάσχιση λίστας

Η διάσχιση (iteration) μιας λίστας μπορεί να πραγματοποιηθεί είτε με χρήση της while είτε με χρήση της for, με τη δεύτερη να θεωρείται συνήθως προτιμότερη λόγω απλούστερης και πιο εκφραστικής σύνταξης. Με τη for μπορούμε να επεξεργαστούμε διαδοχικά κάθε στοιχείο της λίστας χωρίς να διαχειριζόμαστε ρητά δείκτες θέσης. Όταν απαιτείται ταυτόχρονα πρόσβαση τόσο στο στοιχείο όσο

και στη θέση του, ιδιαίτερα χρήσιμη είναι η ενσωματωμένη συνάρτηση enumerate(), η οποία επιστρέφει ζεύγη της μορφής (δείκτης, στοιχείο), διευκολύνοντας τη συγγραφή καθαρού και αναγνώσιμου κώδικα κατά τη διάσχιση της λίστας.

Στον κώδικα Κ. 1.6.5 παρουσιάζεται ένα παράδειγμα όπου μια λίστα διασχίζεται με τέσσερις τρόπους, με τους δύο τελευταίους να είναι αυτοί που προτιμώνται συνήθως.

```
numbers = [10, 20, 30]

i = 0
while i < len(numbers):
    print(numbers[i], end=" ")
    i += 1

print(); print("-" * 8)
for i in range(len(numbers)):
    print(numbers[i], end=" ")

print(); print("-" * 8)
for num in numbers:
    print(num, end=" ")

print(); print("-" * 8)
for index, num in enumerate(numbers):
    print(f"Θέση {index}:{num}", end=" ")
```

Κ. 1.6.5 – Διάσχιση λίστας με 4 διαφορετικούς τρόπους.

Η εκτέλεση του παραπάνω κώδικα θα παράξει την έξοδο XXX που ακολουθεί:

```
10 20 30
--------
10 20 30
--------
10 20 30
--------
Θέση 0:10 Θέση 1:20 Θέση 2:30
```

Ε. 5 – Αποτέλεσμα εκτέλεσης κώδικα Κ. 1.6.5.

1.6.1.5 Ταξινόμηση λίστας

Η ταξινόμηση μιας λίστας στην Python μπορεί να πραγματοποιηθεί είτε με τη συνάρτηση sorted() είτε με τη μέθοδο .sort(). Η sorted(a_list) επιστρέφει μια νέα ταξινομημένη λίστα, χωρίς να μεταβάλλει τα αρχικά δεδομένα της a_list, γεγονός που την καθιστά κατάλληλη όταν επιθυμούμε να διατηρήσουμε αναλλοίωτη την αρχική λίστα. Αντίθετα, η μέθοδος .sort() εφαρμόζεται απευθείας στη λίστα για την οποία καλείται και την ταξινομεί επιτόπου (in place), επιστρέφοντας την τιμή None. Και στις δύο περιπτώσεις, η προεπιλεγμένη συμπεριφορά είναι η ταξινόμηση σε αύξουσα σειρά.

Η σειρά ταξινόμησης μπορεί να μεταβληθεί σε φθίνουσα με τον ορισμό του προαιρετικού ορίσματος reverse=True, τόσο στη sorted() όσο και στη .sort(). Επιπλέον, το προαιρετικό όρισμα key επιτρέπει τον καθορισμό κριτηρίου ταξινόμησης μέσω μιας συνάρτησης που εφαρμόζεται σε κάθε στοιχείο της λίστας. Για παράδειγμα, για ταξινόμηση μιας λίστας συμβολοσειρών a_list με βάση το μήκος κάθε

συμβολοσειράς μπορεί να χρησιμοποιηθεί η εντολή a_list.sort(key=len), ώστε η σύγκριση να βασιστεί στην τιμή που επιστρέφει η len για κάθε στοιχείο.

Στον κώδικα Κ. 1.6.6 παρουσιάζονται παραδείγματα κλήσεων της sorted() και της .sort().

```
numbers = [3, 1, 4, 2]
numbers.sort()  # ταξινομεί την ίδια τη λίστα
print(numbers)  # [1, 2, 3, 4]

numbers = [3, 1, 4, 2]
print(sorted(numbers))  # [1, 2, 3, 4]
print(numbers)  # [3, 1, 4, 2]

words = ["μπανάνα", "ακτινίδιο", "μήλο"]
print(sorted(words, key=len))  # ['μήλο', 'μπανάνα', 'ακτινίδιο']
print(sorted(words, key=len, reverse=True))  # ['ακτινίδιο', 'μπανάνα',
'μήλο']
```

Κ. 1.6.6 – Η μέθοδος sort() και η συνάρτηση sorted().

1.6.1.6 Εμφωλευμένες λίστες

Μια λίστα στην Python μπορεί να περιέχει ως στοιχείο της μια άλλη λίστα, επιτρέποντας έτσι τη δημιουργία εμφωλευμένων (nested) δομών. Με τον τρόπο αυτό μπορούν να αναπαρασταθούν πίνακες δύο ή περισσότερων διαστάσεων, αλλά και ιεραρχικά οργανωμένα δεδομένα που αντανακλούν φυσικές σχέσεις μεταξύ στοιχείων. Οι εμφωλευμένες λίστες είναι ιδιαίτερα χρήσιμες σε προβλήματα που απαιτούν ομαδοποίηση δεδομένων ή δομών τύπου «λίστα από λίστες». Η πρόσβαση σε στοιχεία που βρίσκονται σε βαθύτερα επίπεδα εμφωλιαμσού πραγματοποιείται με τη χρήση διαδοχικών δεικτών. Για παράδειγμα, αν μια λίστα περιέχει υπολίστες, η αναφορά σε συγκεκριμένο στοιχείο απαιτεί πρώτα τον δείκτη της υπολίστας και στη συνέχεια τον δείκτη του στοιχείου μέσα στην υπολίστα. Με αυτόν το μηχανισμό καθίσταται δυνατή η ακριβής προσπέλαση και τροποποίηση δεδομένων σε οποιοδήποτε επίπεδο της δομής.

Στον ακόλουθο κώδικα (Κ. 1.6.7) στο REPL ορίζεται ένας δισδιάστατος πίνακας 3 x 4 μέσω μιας λίστας 3 στοιχείων όπου κάθε στοιχείο της είναι μια άλλη λίστα 4 στοιχείων. Ο κώδικας τροποποιεί το στοιχείο του πίνακα που βρίσκεται στο κάτω δεξιά άκρο του, δηλαδή το στοιχείο στη γραμμή με δείκτη 2 και στη στήλη με δείκτη 3.

```
>>> a = [
... [1,1,1],
... [2,2,2],
... [3,3,3],
... [4,4,4]
... ]
>>> a
[[1, 1, 1], [2, 2, 2], [3, 3, 3], [4, 4, 4]]
>>> a[0][0]
1
>>> a[2][3] = 99
```

```
>>> a
[[1, 1, 1], [2, 2, 2], [3, 3, 3], [4, 4, 99]]
```

Κ. 1.6.7 – Παράδειγμα με εμφωλευμένες λίστες (λίστες ως στοιχεία μιας λίστας).

1.6.1.7 Αντιγραφή λίστας

Η αντιγραφή λίστας στην Python απαιτεί ιδιαίτερη προσοχή. Η εντολή new_list = a_list δεν δημιουργεί νέο, ανεξάρτητο αντίγραφο των δεδομένων, αλλά απλώς ένα νέο όνομα (ψευδώνυμο) που αναφέρεται στο ίδιο αντικείμενο μνήμης. Αυτό σημαίνει ότι οποιαδήποτε μεταβολή πραγματοποιηθεί μέσω του new_list θα επηρεάσει και το a_list, καθώς και τα δύο ονόματα δείχνουν στα ίδια δεδομένα. Πρόκειται επομένως για αντιγραφή αναφοράς και όχι για αντιγραφή περιεχομένου.

Για να δημιουργηθεί πραγματικό, αλλά ρηχό (shallow) αντίγραφο μιας λίστας, θα πρέπει να χρησιμοποιηθεί είτε τεμαχισμός ολόκληρης της λίστας με new_list = a_list[:] είτε η μέθοδος new_list = a_list.copy(). Και στις δύο περιπτώσεις δημιουργείται νέο αντικείμενο λίστας με τα ίδια στοιχεία. Ωστόσο, αν η λίστα περιέχει εμφωλευμένες δομές (π.χ. άλλες λίστες), τότε αντιγράφονται μόνο οι αναφορές στα εσωτερικά αντικείμενα και όχι τα ίδια τα αντικείμενα, γεγονός που καθιστά την αντιγραφή ρηχή και όχι βαθιά (deep copy).

Όταν απαιτείται πλήρως ανεξάρτητο αντίγραφο μιας λίστας, συμπεριλαμβανομένων και των εμφωλευμένων αντικειμένων της, πρέπει να χρησιμοποιηθεί βαθιά αντιγραφή (deep copy). Η βαθιά αντιγραφή δημιουργεί νέο αντικείμενο λίστας και αναδρομικά νέα αντίγραφα για όλα τα εσωτερικά αντικείμενα που περιέχει, ώστε καμία μεταβολή στο αντίγραφο να μην επηρεάζει την αρχική δομή. Στην Python αυτό επιτυγχάνεται με τη συνάρτηση deepcopy() του module copy, ως εξής: import copy και στη συνέχεια new_list = copy.deepcopy(a_list). Η χρήση βαθιάς αντιγραφής είναι απαραίτητη όταν η λίστα περιέχει άλλες δομές και απαιτούνται πλήρη αντίγραφα των δεδομένων.

Ο κώδικας Κ. 1.6.8 παρουσιάζει τις διαφορές μεταξύ αντιγραφής αναφοράς, ρηχής αντιγραφής (shallow copy) και βαθιάς αντιγραφής (deep copy) σε λίστες. Αρχικά, δείχνει ότι η απλή ανάθεση (new_list = a_list) δημιουργεί ψευδώνυμο στο ίδιο αντικείμενο, με αποτέλεσμα κάθε μεταβολή να επηρεάζει και τις δύο μεταβλητές. Στη συνέχεια, με τεμαχισμό (a_list[:]) επιτυγχάνεται ρηχή αντιγραφή, η οποία λειτουργεί σωστά σε απλές (flat) λίστες, καθώς οι δύο λίστες γίνονται ανεξάρτητες. Ωστόσο, όταν η λίστα είναι εμφωλευμένη, η ρηχή αντιγραφή αντιγράφει μόνο το εξωτερικό επίπεδο και οι εσωτερικές λίστες παραμένουν κοινές, οπότε μεταβολές σε αυτές επηρεάζουν και την αρχική λίστα. Τέλος, με τη χρήση της copy.deepcopy() δημιουργείται πλήρως ανεξάρτητο αντίγραφο όλων των επιπέδων της δομής, ώστε καμία μεταβολή στο αντίγραφο να μην επηρεάζει την αρχική λίστα.

```
import copy
```

```
a_list = [1, 2, 3]
new_list = a_list  # Αντιγραφή αναφοράς (alias)
new_list.append(4)
print(a_list)  # [1, 2, 3, 4]  => επηρεάζεται
print(new_list)  # [1, 2, 3, 4]

a_list = [1, 2, 3]
new_list = a_list[:]  # Ρηχή αντιγραφή (slice)
new_list.append(4)
print(a_list)  # [1, 2, 3]     => δεν επηρεάζεται
print(new_list)  # [1, 2, 3, 4]

a_list = [[1, 2], [3, 4]]  # εμφωλευμένη λίστα
new_list = a_list.copy()  # Ρηχή αντιγραφή
new_list[0].append(99)
print(a_list)  # [[1, 2, 99], [3, 4]]  => επηρεάζεται (εσωτερική λίστα)
print(new_list)  # [[1, 2, 99], [3, 4]]

a_list = [[1, 2], [3, 4]]
new_list = copy.deepcopy(a_list)  # Βαθιά αντιγραφή
new_list[0].append(99)
print(a_list)  # [[1, 2], [3, 4]]      => δεν επηρεάζεται
print(new_list)  # [[1, 2, 99], [3, 4]]
```

Κ. 1.6.8 – Αντιγραφή αναφοράς vs. ρηχή αντιγραφή vs. βαθιά αντιγραφή.

1.6.2. Πλειάδες

Οι πλειάδες (tuples) είναι δομές δεδομένων που μπορούν να περιέχουν ετερογενή στοιχεία, τα οποία διαχωρίζονται με κόμματα και περικλείονται σε παρενθέσεις. Στην πράξη, οι παρενθέσεις είναι προαιρετικές σε πολλές περιπτώσεις, καθώς η πλειάδα ορίζεται από τα κόμματα και όχι από τις παρενθέσεις. Τα περιεχόμενα μιας πλειάδας δεν μπορούν να τροποποιηθούν κατά την εκτέλεση του προγράμματος, είναι δηλαδή immutable (μη-τροποποιήσιμες ή αλλιώς μη-μεταλλάξιμες δομές). Όπως και οι λίστες, οι πλειάδες ανήκουν στις ακολουθίες (sequences), επομένως τα στοιχεία τους είναι διατεταγμένα (ordered) και η πρόσβαση σε αυτά γίνεται μέσω δεικτών θέσης με αρίθμηση που ξεκινά από το 0.

Μια λίστα μπορεί να μετατραπεί σε πλειάδα με τη συνάρτηση tuple(), ενώ μια πλειάδα μπορεί να μετατραπεί σε λίστα με τη συνάρτηση list(). Ιδιαίτερη προσοχή απαιτείται κατά τη δημιουργία πλειάδας ενός στοιχείου: η έκφραση (1) θεωρείται απλώς ακέραιος αριθμός, ενώ η (1,) δημιουργεί πλειάδα ενός στοιχείου, όπως φαίνεται από τις εκφράσεις type((1)) που επιστρέφει <class 'int'> και την type((1,)) που επιστρέφει <class 'tuple'>. Οι πλειάδες είναι γενικά πιο αποδοτικές από τις λίστες ως προς τη μνήμη που καταλαμβάνουν και ως προς την ταχύτητα εκτέλεσης, λόγω του ότι είναι μη-τροποποιήσιμες.

Όπως και οι λίστες, οι πλειάδες υποστηρίζουν τεμαχισμό και εμφωλιασμό, δηλαδή πλειάδες με στοιχεία άλλες πλειάδες. Προφανώς στην περίπτωση των εμφωλευμένων δομών μπορεί για παράδειγμα μια λίστα να περιέχει εμφωλευμένες πλειάδες και το αντίστροφο.

Στα ακόλουθα παραδείγματα (Κ. 1.6.9) στο REPL παρουσιάζονται τα χαρακτηριστικά και ο τρόπος χειρισμού πλειάδων που αναφέρθηκαν παραπάνω.

```
>>> t = (1, 2, 3)
>>> t
(1, 2, 3)                    # βασική δημιουργία πλειάδας

>>> t = 1, 2, 3
>>> t
(1, 2, 3)                    # οι παρενθέσεις είναι προαιρετικές

>>> t[0], t[2]
(1, 3)                       # πρόσβαση με δείκτη θέσης

>>> t[1:]
(2, 3)                       # τεμαχισμός (slicing)

>>> type((1))
<class 'int'>                # δεν είναι πλειάδα

>>> type((1,))
<class 'tuple'>              # πλειάδα ενός στοιχείου

>>> lst = [1, 2, 3]
>>> tuple(lst)
(1, 2, 3)                    # μετατροπή λίστας σε πλειάδα

>>> list(t)
[1, 2, 3]                    # μετατροπή πλειάδας σε λίστα

>>> nested = ((1, 2), (3, 4))
>>> nested[1][0]
3                            # εμφωλευμένη πλειάδα και πολλαπλοί δείκτες

>>> mixed = [ (1, 2), (3, 4) ]
>>> mixed[0]
(1, 2)                       # λίστα που περιέχει πλειάδες

>>> complex_structure = ( [1, 2], (3, 4) )
>>> complex_structure
([1, 2], (3, 4))             # πλειάδα που περιέχει λίστα και πλειάδα
```

Κ. 1.6.9 – Παραδείγματα με πλειάδες.

1.6.3. Λεξικά

Τα λεξικά (dictionaries) είναι μια ιδιαίτερα χρήσιμη δομή δεδομένων, στην οποία αποθηκεύονται ζεύγη της μορφής key: value. Τα κλειδιά (keys) πρέπει να είναι μη-τροποποιήσιμες (immutable) τιμές, όπως συμβολοσειρές, ακέραιοι ή πλειάδες, ενώ οι τιμές (values) μπορούν να είναι οποιουδήποτε τύπου δεδομένων. Η δημιουργία ενός λεξικού γίνεται με χρήση αγκυλών { }, μέσα στις οποίες ορίζονται ζεύγη key: value χωρισμένα με κόμματα. Από την Python 3.7 και μετά, τα λεξικά διατηρούν τη σειρά εισαγωγής των στοιχείων τους, γεγονός που σημαίνει ότι η διάσχισή τους αποδίδει τα ζεύγη με τη σειρά που προστέθηκαν.

Σε ένα λεξικό μπορούν να προστεθούν νέα ζεύγη ή να τροποποιηθούν υπάρχοντα μέσω ανάθεσης σε συγκεκριμένο κλειδί, ενώ η διαγραφή γίνεται επίσης με βάση το κλειδί. Η πρόσβαση σε μια τιμή πραγματοποιείται μέσω του αντίστοιχου κλειδιού, χρησιμοποιώντας σύνταξη της μορφής dict_name[key]. Τα κλειδιά ενός λεξικού πρέπει να είναι μοναδικά, ενώ αν επιχειρηθεί η εισαγωγή ίδιου κλειδιού δεύτερη φορά, η νέα τιμή αντικαθιστά την υπάρχουσα.

Στο κώδικα Κ. 1.6.10 παρουσιάζονται παραδείγματα χειρισμού ενός λεξικού στο REPL.

```
>>> d = {"name": "Άννα", "age": 25}

>>> d["name"]
'Άννα'                          # πρόσβαση σε τιμή μέσω κλειδιού

>>> d["age"] = 26
>>> d
{'name': 'Άννα', 'age': 26}     # τροποποίηση υπάρχοντος ζεύγους

>>> d["city"] = "Αθήνα"
>>> d
{'name': 'Άννα', 'age': 26, 'city': 'Αθήνα'}  # προσθήκη νέου ζεύγους

>>> d["age"] = 30
>>> d
{'name': 'Άννα', 'age': 30, 'city': 'Αθήνα'}  # αντικατάσταση τιμής

>>> del d["city"]
>>> d
{'name': 'Άννα', 'age': 30}     # διαγραφή ζεύγους με βάση το κλειδί

>>> d["height"]
Traceback (most recent call last):
KeyError: 'height'              # σφάλμα αν το κλειδί δεν υπάρχει
```

Κ. 1.6.10 – Παραδείγματα με λεξικό.

1.6.3.1 Μέθοδοι για διάσχιση λεξικών

Οι μέθοδοι keys(), values() και items() χρησιμοποιούνται για την πρόσβαση στα περιεχόμενα ενός λεξικού με οργανωμένο τρόπο. Η keys() επιστρέφει μια προβολή (view) με τα κλειδιά του λεξικού, η .values() επιστρέφει αντίστοιχα τις τιμές, ενώ η items() επιστρέφει ζεύγη της μορφής (key, value) ως πλειάδες δύο στοιχείων. Οι προβολές αυτές είναι δυναμικές, δηλαδή αντανακλούν τυχόν μεταβολές του λεξικού. Η διάσχιση ενός λεξικού πραγματοποιείται συνήθως με την εντολή for, είτε απευθείας πάνω στα κλειδιά είτε με χρήση των παραπάνω μεθόδων για ταυτόχρονη πρόσβαση σε κλειδιά και τιμές, όπως φαίνεται στα ακόλουθα παραδείγματα στο REPL (Κ. 1.6.11).

```
>>> d = {"name": "Άννα", "age": 30}

>>> d.keys()
dict_keys(['name', 'age'])          # προβολή κλειδιών

>>> d.values()
dict_values(['Άννα', 30])           # προβολή τιμών
```

```
>>> d.items()
dict_items([('name', 'Άννα'), ('age', 30)])  # ζεύγη (key, value)

>>> for key in d:
...     print(key)
...
name
age                                  # διάσχιση κλειδιών

>>> for value in d.values():
...     print(value)
...
Άννα
30                                   # διάσχιση τιμών

>>> for key, value in d.items():
...     print(key, "->", value)
...
name -> Άννα
age -> 30                            # διάσχιση ζευγών
```

Κ. 1.6.11 – Πρόσβαση στα κλειδιά και στις τιμές λεξικών και διάσχιση λεξικών.

1.6.3.2 Μέθοδοι λεξικών

Οι μέθοδοι get(), pop(), update() και clear() παρέχουν επιπλέον δυνατότητες διαχείρισης λεξικών, ενώ υπάρχουν και άλλες μέθοδοι που εφαρμόζονται σε λεξικά. Η get(k) επιστρέφει την τιμή που αντιστοιχεί στο κλειδί k, χωρίς να προκαλεί σφάλμα αν το κλειδί δεν υπάρχει καθώς στην περίπτωση αυτή επιστρέφει None (ή προαιρετικά μια τιμή που μπορεί να δοθεί ως δεύτερο όρισμα). Η pop(k) αφαιρεί το ζεύγος με κλειδί k και επιστρέφει την αντίστοιχη τιμή, ενώ αν το κλειδί δεν υπάρχει προκαλεί KeyError (εκτός αν δοθεί προεπιλεγμένη τιμή). Η update(a_dict) ενημερώνει το λεξικό με τα ζεύγη key: value που περιέχονται σε ένα άλλο λεξικό ή γενικά σε ένα διασχίσιμο (iterable) αντικείμενο ζευγών, αντικαθιστώντας τυχόν υπάρχοντα κλειδιά. Τέλος, η clear() αφαιρεί όλα τα ζεύγη από το λεξικό, αφήνοντάς το κενό.

Στο κώδικα Κ. 1.6.12 παρουσιάζονται στο REPL παραδείγματα πρόσβασης και ενημέρωσης τιμών σε λεξικό με βάση το κλειδί καθώς και ενημέρωσης του λεξικού με νέα ζεύγη κλειδιού-τιμής.

```
>>> d = {"name": "Άννα", "age": 30}

>>> d.get("name")
'Άννα'                          # επιστροφή τιμής

>>> d.get("city")
None                            # δεν προκαλεί σφάλμα

>>> d.get("city", "Άγνωστο")
'Άγνωστο'                       # προεπιλεγμένη τιμή

>>> d.pop("age")
30                              # αφαίρεση και επιστροφή τιμής

>>> d
{'name': 'Άννα'}
```

```
>>> d.update({"city": "Αθήνα", "age": 25})
>>> d
{'name': 'Άννα', 'city': 'Αθήνα', 'age': 25}  # ενημέρωση/προσθήκη

>>> d.clear()
>>> d
{}                                  # κενό λεξικό
```

Κ. 1.6.12 – Μέθοδοι λεξικών για λήψη και ενημέρωση τιμών καθώς και για ενημέρωση λεξικού με νέα ζεύγη.

1.6.4. Σύνολα

Τα σύνολα (sets) είναι δομές δεδομένων που μπορούν να περιέχουν ετερογενή στοιχεία, χωρίς διπλότυπες τιμές, τα οποία διαχωρίζονται με κόμματα και περικλείονται σε αγκύλες { }. Σε αντίθεση με τις λίστες και τις πλειάδες, τα σύνολα δεν διατηρούν διάταξη μεταξύ των στοιχείων τους, επομένως δεν υποστηρίζουν πρόσβαση μέσω δεικτών θέσης. Τα σύνολα είναι τροποποιήσιμες (mutable) δομές, δηλαδή επιτρέπουν την προσθήκη και αφαίρεση στοιχείων μετά τη δημιουργία τους.

Ιδιαίτερη προσοχή απαιτείται κατά τη δημιουργία ενός κενού συνόλου ή ενός συνόλου με ένα μόνο στοιχείο. Η έκφραση { } δεν δημιουργεί σύνολο αλλά κενό λεξικό. Για τη δημιουργία κενού συνόλου χρησιμοποιείται η συνάρτηση set(), ενώ για σύνολο με ένα στοιχείο μπορούν να χρησιμοποιηθούν αγκύλες με τιμή, π.χ. {99}. Η έκφραση a = set([99]) δημιουργεί το σύνολο {99}, ενώ η a = {99} επίσης δημιουργεί σύνολο και όχι λεξικό, ενώ λεξικό δημιουργείται μόνο όταν υπάρχουν ζεύγη κλειδιού - τιμής, π.χ. {99: "τιμή"}.

1.6.4.1 Μέθοδοι προσθήκης, διαγραφής και ενημέρωσης συνόλων

Οι βασικές μέθοδοι των συνόλων περιλαμβάνουν τις add(x), η οποία προσθέτει το στοιχείο x στο σύνολο (αν υπάρχει ήδη, το σύνολο παραμένει αμετάβλητο), και .update(x), όπου το όρισμα x μπορεί να είναι λίστα, πλειάδα, σύνολο ή γενικά οποιοδήποτε επαναλήψιμο αντικείμενο (iterable), και κάθε στοιχείο του προστίθεται στο σύνολο. Για διαγραφή στοιχείων χρησιμοποιούνται οι discard(x), που αφαιρεί το στοιχείο x χωρίς να προκαλεί σφάλμα αν αυτό δεν υπάρχει, και remove(x), η οποία επίσης διαγράφει το στοιχείο αλλά προκαλεί KeyError αν δεν βρεθεί. Επιπλέον, η pop() αφαιρεί και επιστρέφει ένα τυχαίο στοιχείο (λόγω έλλειψης διάταξης), ενώ η clear() διαγράφει όλα τα στοιχεία του συνόλου.

Στη συνέχεια ακολουθούν παραδείγματα, στο REPL, των μεθόδων που αναφέρθηκαν (Κ. 1.6.13).

```
>>> s = {1, 2, 3}

>>> s.add(4); s
{1, 2, 3, 4}                # προσθήκη στοιχείου

>>> s.add(4); s
{1, 2, 3, 4}                # δεν προστίθεται διπλότυπο
```

```
>>> s.update([4, 5, 6]); s
{1, 2, 3, 4, 5, 6}           # προσθήκη πολλών στοιχείων από iterable

>>> s.discard(10); s
{1, 2, 3, 4, 5, 6}           # δεν προκαλεί σφάλμα αν δεν υπάρχει

>>> s.remove(6); s
{1, 2, 3, 4, 5}              # διαγραφή στοιχείου

>>> s.remove(10)
Traceback (most recent call last):
KeyError: 10                 # σφάλμα αν το στοιχείο δεν υπάρχει

>>> x = s.pop(); x, s
(1, {2, 3, 4, 5})            # αφαίρεση και επιστροφή τυχαίου στοιχείου

>>> s.clear(); s
set()                        # διαγραφή όλων των στοιχείων
```

Κ. 1.6.13 – Μέθοδοι συνόλων.

1.6.4.2 Μέθοδοι πράξεων συνόλων

Τα σύνολα υποστηρίζουν μαθηματικές πράξεις συνόλων. Η union() ή ο τελεστής | υπολογίζει την ένωση, η intersection() ή & την τομή, η difference() ή - τη διαφορά και η symmetric_difference() ή ^ τη συμμετρική διαφορά δύο συνόλων. Η διαφορά δύο συνόλων A και B (A – B) είναι το σύνολο των στοιχείων που ανήκουν στο A αλλά δεν ανήκουν στο B. Δηλαδή, αφαιρούνται από το πρώτο σύνολο όλα τα στοιχεία που υπάρχουν και στο δεύτερο. Η συμμετρική διαφορά δύο συνόλων A και B είναι το σύνολο των στοιχείων που ανήκουν είτε στο A είτε στο B, αλλά όχι και στα δύο ταυτόχρονα.

Υπάρχουν επίσης μέθοδοι σύγκρισης όπως η issubset() και η issuperset() για έλεγχο αν ένα σύνολο είναι υποσύνολο ή υπερύνολο ενός άλλου συνόλου αντίστοιχα, καθώς και η isdisjoint() για έλεγχο αν δύο σύνολα είναι ξένα, δηλαδή δεν έχουν κοινά στοιχεία.

Στα ακόλουθα παραδείγματα στο REPL, χρησιμοποιούνται οι πράξεις και οι μέθοδοι συνόλων που αναφέρθηκαν παραπάνω (Κ. 1.6.14).

```
>>> A = {1, 2, 3, 4}
>>> B = {3, 4, 5, 6}

>>> A.union(B)
{1, 2, 3, 4, 5, 6}           # ένωση

>>> A | B
{1, 2, 3, 4, 5, 6}           # ένωση με τελεστή |

>>> A.intersection(B)
{3, 4}                       # τομή

>>> A & B
{3, 4}                       # τομή με τελεστή &

>>> A.difference(B)
{1, 2}                       # διαφορά A – B (στο A αλλά όχι στο B)
```

```
>>> A - B
{1, 2}                      # διαφορά με τελεστή -

>>> B - A
{5, 6}                      # διαφορά B – A

>>> A.symmetric_difference(B)
{1, 2, 5, 6}                # συμμετρική διαφορά

>>> A ^ B
{1, 2, 5, 6}                # συμμετρική διαφορά με τελεστή ^

>>> {1, 2}.issubset(A)
True                        # έλεγχος υποσυνόλου

>>> A.issuperset({1, 2})
True                        # έλεγχος υπερσυνόλου

>>> A.isdisjoint({7, 8})
True                        # δεν έχουν κοινά στοιχεία

>>> A.isdisjoint(B)
False                       # έχουν κοινά στοιχεία (3, 4)
```

Κ. 1.6.14 – Πράξεις συνόλων.

1.6.5. Συνδυασμός ακολουθιών

Η συνάρτηση zip() συνδυάζει δύο ή περισσότερα διασχίσιμα αντικείμενα (iterables), όπως λίστες, πλειάδες, συμβολοσειρές κ.α., δημιουργώντας ένα νέο διασχίσιμο αντικείμενο του οποίου κάθε στοιχείο είναι μια πλειάδα. Κάθε πλειάδα περιέχει στοιχεία που βρίσκονται στην ίδια θέση στα αρχικά αντικείμενα. Η zip() δεν επιστρέφει λίστα αλλά ένα αντικείμενο τύπου iterator, το οποίο μπορεί να μετατραπεί σε λίστα με τη list() αν απαιτείται. Η λειτουργία της σταματά μόλις εξαντληθεί το συντομότερο από τα δοθέντα αντικείμενα, δηλαδή το μήκος του αποτελέσματος καθορίζεται από το μικρότερο iterable.

Στα παραδείγματα στο REPL που ακολουθούν (Κ. 1.6.15) παρουσιάζεται η χρήση της συνάρτησης zip().

```
>>> a = [1, 2, 3]
>>> b = ["α", "β", "γ"]

>>> list(zip(a, b))
[(1, 'α'), (2, 'β'), (3, 'γ')]   # συνδυασμός στοιχείων ίδιας θέσης

>>> c = [10, 20]
>>> list(zip(a, c))
[(1, 10), (2, 20)]               # σταματά στο μικρότερο μήκος

>>> x = [1, 2, 3]
>>> y = ["A", "B", "Γ"]
>>> z = ["x", "y", "z"]

>>> list(zip(x, y, z))
```

```
[(1, 'A', 'x'), (2, 'B', 'y'), (3, 'Γ', 'z')]  # συνδυασμός τριών λιστών
```

Κ. 1.6.15 – Συνδυασμός λιστών με τη zip().

1.6.6. Περιφραστικές λίστες

Μια περιφραστική λίστα (list comprehension) αποτελεί έναν συμπαγή και εκφραστικό τρόπο δημιουργίας λιστών, ενώ αντίστοιχη σύνταξη υπάρχει και για σύνολα και λεξικά. Συνδυάζει έναν βρόχο for και μια έκφραση σε μία μόνο γραμμή κώδικα, επιτρέποντας τη δημιουργία νέας λίστας χωρίς τη χρήση πολλαπλών εντολών. Συχνά οι περιφραστικές λίστες είναι πιο ευανάγνωστες και ταχύτερες στη συγγραφή σε σύγκριση με τον παραδοσιακό τρόπο χρήσης βρόχου for. Η βασική σύνταξη ενός list comprehension είναι:

```
[expression for item in iterable if condition]
```

όπου expression είναι η έκφραση που παράγει τα στοιχεία της νέας λίστας, item η μεταβλητή που αναπαριστά κάθε στοιχείο της ακολουθίας, iterable η δομή που διασχίζεται και condition ένα προαιρετικό φίλτρο που καθορίζει ποια στοιχεία θα συμπεριληφθούν. Αν το condition παραλειφθεί, όλα τα στοιχεία του iterable συμμετέχουν στην παραγωγή της νέας λίστας.

Ακολουθούν μερικά παραδείγματα στο REPL (Κ. 1.6.16) με περιφραστικές λίστες:

```
>>> nums = [1, 2, 3, 4, 5]

>>> [x**2 for x in nums]
[1, 4, 9, 16, 25]     # τετράγωνα όλων των στοιχείων

>>> [x for x in nums if x % 2 == 0]
[2, 4]                # φιλτράρισμα άρτιων

>>> [x**2 for x in nums if x % 2 == 0]
[4, 16]               # συνδυασμός μετασχηματισμού και φίλτρου
```

Κ. 1.6.16 – Περιφραστικές λίστες.

Αντίστοιχη σύνταξη με τις περιφραστικές λίστες υπάρχει και για τη δημιουργία συνόλων και λεξικών. Στην περίπτωση των συνόλων χρησιμοποιούνται αγκύλες { } αντί για [ ], ενώ για τα λεξικά η έκφραση πρέπει να παράγει ζεύγη της μορφής key: value. Ακολουθούν παραδείγματα (Κ. 1.6.17) στο REPL:

```
>>> words = ["μήλο", "μπανάνα", "αχλάδι", "μήλο", "αχλάδι"]
>>> {w for w in words}
{'μήλο', 'μπανάνα', 'αχλάδι'}         # απομάκρυνση διπλοτύπων

>>> {w[0] for w in words}
{'μ', 'α'}                            # αρχικά γράμματα (μοναδικά)

>>> students = ["Άννα", "Νίκος", "Ελένη"]
>>> {name: len(name) for name in students}
{'Άννα': 4, 'Νίκος': 5, 'Ελένη': 5}   # όνομα -> μήκος ονόματος

>>> grades = {"Άννα": 85, "Νίκος": 72, "Ελένη": 91}
>>> {name: grade for name, grade in grades.items() if grade >= 80}
{'Άννα': 85, 'Ελένη': 91}             # φιλτράρισμα βάσει βαθμού
```

Κ. 1.6.17 – Περιφραστικός τρόπος δημιουργίας λεξικών.

1.7. Οργάνωση κώδικα σε τμήματα και πακέτα

Η οργάνωση του κώδικα σε τμήματα (modules) και πακέτα (packages) στη Python αποτελεί βασική πρακτική για την ανάπτυξη επεκτάσιμων και συντηρήσιμων εφαρμογών. Ένα module επιτρέπει τη λογική ομαδοποίηση συναφών συναρτήσεων, κλάσεων και μεταβλητών σε ξεχωριστό αρχείο, διευκολύνοντας την επαναχρησιμοποίηση μέσω import. Τα packages, ως συλλογές από modules οργανώνονται σε ιεραρχική δομή καταλόγων, υποστηρίζουν την ανάπτυξη μεγάλων έργων και επιτρέπουν ξεκαθαρό διαχωρισμό ρόλων για τα επιμέρους υποσυστήματα των εφαρμογών. Μέσω της οργάνωσης του κώδικα σε τμήματα και πακέτα επιτυγχάνεται καλύτερη αναγνωσιμότητα, ευκολότερος έλεγχος (testing), και αποτελεσματικότερη συνεργασία σε ομαδικά έργα, καθώς κάθε υποσύστημα μπορεί να αναπτυχθεί αυτόνομα.

1.7.1. Modules

Ένα module είναι μια συλλογή από αντικείμενα Python που μπορεί να γίνει import. Πρόκειται για ένα αρχείο .py που χρησιμοποιείται για τη λογική οργάνωση του κώδικα σε επαναχρησιμοποιήσιμα στοιχεία. Ένα module μπορεί να περιέχει συναρτήσεις, κλάσεις, μεταβλητές, καθώς και εκτελέσιμο κώδικα που αρχικοποιεί το module.

Τα modules διακρίνονται στις εξής κατηγορίες:

- **Built-in modules:** Έχουν υλοποιηθεί σε C και αποτελούν μέρος του πυρήνα του διερμηνευτή της Python. Παρέχονται έτοιμα προς χρήση χωρίς να απαιτείται εγκατάσταση από τον χρήστη. Ορισμένα παραδείγματα built-in modules είναι τα ακόλουθα:
    - sys → πρόσβαση σε παραμέτρους και λειτουργίες του διερμηνευτή
    - math → μαθηματικές συναρτήσεις
    - time → συναρτήσεις διαχείρισης χρόνου
    - os → αλληλεπίδραση με το λειτουργικό σύστημα
    - random → παραγωγή ψευδοτυχαίων αριθμών
- **Modules τρίτων κατασκευαστών (third-party modules):** Δεν περιλαμβάνονται στην τυπική εγκατάσταση της Python και εγκαθίστανται συνήθως μέσω του pip. Παραδείγματα αποτελούν οι βιβλιοθήκες NumPy, pandas και Matplotlib, οι οποίες είναι οργανωμένες ως packages που περιέχουν πολλαπλά modules και επεκτείνουν σημαντικά τις δυνατότητες της γλώσσας.

- **Modules ορισμένα από τον χρήστη (user-defined modules):** Πρόκειται για αρχεία .py που δημιουργεί ο ίδιος ο προγραμματιστής για να οργανώσει τον κώδικα που αναπτύσσει. Επιτρέπουν το διαχωρισμό της λειτουργικότητας σε ανεξάρτητα τμήματα.

1.7.1.1 Χρήση modules

Στην παράγραφο αυτή θα περιγραφεί η χρήση σταθερών και συναρτήσεων του module math, ως ένα παράδειγμα των διαφορετικών εναλλακτικών τρόπων με τους οποίους μπορεί να επιτευχθεί πρόσβαση στα στοιχεία των modules.

Έστω ότι επιδιώκεται η χρήση της σταθεράς pi και της συνάρτησης ημιτόνου, sin() για τον υπολογισμό του ημιτόνου των 30°, δηλαδή του π/6. Για να καταστεί αυτό δυνατό, απαιτείται αρχικά η εισαγωγή του module με την εντολή:

```
import math
```

Στην περίπτωση αυτή, η πρόσβαση στα στοιχεία του module πραγματοποιείται μέσω του namespace του (math), ως εξής:

```
math.sin(math.pi / 6)
```

Η ρητή αναφορά στο όνομα του module (math.sin, math.pi) διασφαλίζει τη σαφή προέλευση κάθε αντικειμένου και αποτρέπει πιθανές συγκρούσεις ονομάτων. Αν περισσότερα από ένα modules ορίζουν αντικείμενα με το ίδιο όνομα (π.χ. sin ή pi), η χρήση της μορφής module_name.object_name επιτρέπει την ακριβή ταυτοποίηση του επιθυμητού στοιχείου.

Είναι επίσης δυνατός ο ορισμός ψευδωνύμου (alias) για ένα module κατά την εισαγωγή του. Για την περίπτωση του module math ο ορισμός ενός ψευδωνύμου γίνεται ως εξής:

```
import math as m
m.sin(m.pi / 6)
```

Με τον τρόπο αυτό, το όνομα math αντικαθίσταται στο τρέχον namespace από το ψευδώνυμο m. Η πρακτική αυτή είναι ιδιαίτερα χρήσιμη όταν το όνομα του module είναι μεγάλο ή όταν χρησιμοποιείται συχνά μέσα στο πρόγραμμα (π.χ. import numpy as np).

Εναλλακτικά, μπορεί να πραγματοποιηθεί επιλεκτική εισαγωγή συγκεκριμένων αντικειμένων:

```
from math import sin, pi
sin(pi / 6)
```

Με τον τρόπο αυτό, τα ονόματα sin και pi εισάγονται απευθείας στο τρέχον namespace και μπορούν να χρησιμοποιηθούν χωρίς το πρόθεμα του module. Τέλος, είναι δυνατή και η καθολική εισαγωγή:

```
from math import *
```

Η τελευταία μορφή, αν και συντακτικά έγκυρη, δεν συνιστάται σε μεγάλα προγράμματα, διότι εισάγει όλα τα ονόματα του module στο τρέχον namespace, αυξάνοντας την πιθανότητα συγκρούσεων και μειώνοντας τη σαφήνεια και τη συντηρησιμότητα του κώδικα.

1.7.1.2 Παράδειγμα module ορισμένου από τον χρήστη

Έστω ένα module με όνομα my_mod.py (Κ. 1.7.1), το οποίο περιέχει συναρτήσεις και μεταβλητές στις οποίες έχουν ανατεθεί τιμές. Τα στοιχεία αυτά αποτελούν τα λεγόμενα attributes (χαρακτηριστικά) του module, δηλαδή τα ονόματα που ορίζονται στο namespace του.

```
"""
Ένα απλό module χρήστη
"""

PI = 3.14159

def square(x):
    return x * x

def hello(name):
    return f"Hello, {name}!"
```

Κ. 1.7.1 – Ο κώδικας ενός απλού module.

Πέρα των attributes που ορίζονται ρητά από τον προγραμματιστή, κάθε module διαθέτει και ορισμένα προκαθορισμένα (built-in) attributes, που υπάρχουν σε όλα τα modules της Python. Ενδεικτικά αναφέρονται τα ακόλουθα:

- __file__ → περιέχει το όνομα και τη διαδρομή του αρχείου από το οποίο φορτώθηκε το module.
- __name__ → λαμβάνει την τιμή "__main__" όταν το ίδιο το module εκτελείται απευθείας ως κύριο πρόγραμμα. Όταν όμως γίνεται import από άλλο πρόγραμμα, λαμβάνει ως τιμή το όνομα του module.
- __doc__ → περιέχει το docstring του module, δηλαδή τη συμβολοσειρά τεκμηρίωσης που έχει τοποθετηθεί στην αρχή του αρχείου.

Το module μπορεί να εισαχθεί (import) σε άλλα προγράμματα και τα attributes του να προσπελαστούν με χρήση της σημειογραφίας τελείας όπως στον κώδικα Κ. 1.7.2 που υποθετικά βρίσκεται στο αρχείο main.py, στον ίδιο φάκελο με το αρχείο my_mod.py.

```
import my_mod

print(my_mod.PI)  # 3.14159
print(my_mod.square(5))  # 25
print(my_mod.hello("Μαρία"))  # Hello, Μαρία!

print(my_mod.__file__)  # εμφανίζει το path του my_mod.py
print(my_mod.__name__)  # εμφανίζει "my_mod"
print(__name__)  # εμφανίζει "__main__"
```

```
print(my_mod.__doc__)  # εμφανίζει το docstring: "Ενα απλό module χρήστη"
```
Κ. 1.7.2 – Εισαγωγή του module my_mod και κλήση.

1.7.1.3 Ο ρόλος του __name__ == "__main__"

Συχνά στον κώδικα συναντάται η εντολή:

```
if __name__ == "__main__":
    ...
```

που σημαίνει ότι οι εντολές που βρίσκονται στο μπλοκ της if θα εκτελεστούν μόνο αν το αρχείο κώδικα που τις περιέχει είναι αυτό που πρωτογενώς εκτελείται, δηλαδή δεν γίνεται απλά import από κάποιοο άλλο αρχείο κώδικα που έχει επιλεγεί να εκτελεστεί. Το ακόλουθο παράδειγμα αποσαφηνίζει το παραπάνω.

Έστω ένα αρχείο με όνομα a.py, που περιέχει μια συνάρτηση αλλά και κάποιο κώδικα εκτός συναρτήσεων, όπως φαίνεται στον κώδικα Κ. 1.7.3. Αν το a.py γίνει import από ένα αρχείο b.py (κώδικας Κ. 1.7.4), που βρίσκεται στον ίδιο φάκελο με το a.py, και το b.py εκτελεστεί τότε θα εκτελεστεί ο κώδικας εκτός συναρτήσεων του a.py. Αυτό αποτελεί μια συμπεριφορά που συνήθως δεν είναι επιθυμητή.

```
def my_function():
    print("Αυτή είναι μια συνάρτηση του module a!")


for _ in range(2):
    print("Κάτι που πρέπει να γίνεται όταν εκτελείται το a.py")
```
Κ. 1.7.3 – Ο κώδικας του a.py.

```
import a

a.my_function()
```
Κ. 1.7.4 – Ο κώδικας του b.py καλεί τη συνάρτηση my_function() που ορίζεται στο module a.

Η έξοδος της εκτέλεσης του b.py θα είναι η ακόλουθη:

```
Κάτι που πρέπει να γίνεται όταν εκτελείται το a.py
Κάτι που πρέπει να γίνεται όταν εκτελείται το a.py
Αυτή είναι μια συνάρτηση του module a!
```

Για να αποφευχθεί η εκτέλεση του κώδικα εκτός συναρτήσεων του a.py όταν εκτελείται το b.py θα πρέπει ο κώδικας εκτός συναρτήσεων του a.py να τοποθετηθεί μέσα στο μπλοκ εντολών μιας εντολής if που θα ελέγχει αν η τιμή της ειδικής μεταβλητής __name__ είναι ίση με "__main__". Συνεπώς, η νέα μορφή του a.py θα είναι όπως φαίνεται στον κώδικα Κ. 1.7.5.

```
def my_function():
    print("Αυτή είναι μια συνάρτηση του module a!")

if __name__ == "__main__":
    for _ in range(2):
        print("Κάτι που πρέπει να γίνεται όταν εκτελείται το a.py")
```
Κ. 1.7.5 – Ο κώδικας του a.py μετά την προσθήκη του ελέγχου if __name__ == "__main__".

Η εκτέλεση του b.py σε αυτή την περίπτωση θα προκαλέσει την εκτέλεση μόνο του κώδικα της συνάρτησης my_function() από το module a. Η έξοδος θα είναι:

```
Αυτή είναι μια συνάρτηση του module a!
```

1.7.2. Packages

Ένα package (πακέτο) αποτελεί μια οργανωμένη συλλογή από modules και ενδεχομένως άλλα υποπακέτα (subpackages), τα οποία δομούνται ιεραρχικά μέσα σε έναν κατάλογο του συστήματος αρχείων. Η έννοια του package επιτρέπει την ομαδοποίηση σχετικού κώδικα σε θεματικές ενότητες, διευκολύνοντας τόσο την επαναχρησιμοποίηση όσο και τη συντήρηση μεγάλων έργων λογισμικού. Με άλλα λόγια, τα packages αποτελούν μια ειδική περίπτωση modules, τα οποία όμως αναπαρίστανται ως φάκελοι που περιέχουν επιμέρους αρχεία Python.

Χαρακτηριστικό παράδειγμα πακέτου αποτελεί το numpy, το οποίο περιλαμβάνει subpackages όπως τα core, linalg, random κ.α. Το subpackage numpy.linalg, για παράδειγμα, περιέχει modules σχετικά με πράξεις γραμμικής άλγεβρας, όπως υπολογισμό αντιστρόφου πίνακα, υπολογισμό ιδιοτιμών, κ.λπ. Έτσι, ένας χρήστης μπορεί είτε να εισαγάγει ολόκληρο το package (import numpy as np) είτε να εισαγάγει συγκεκριμένο subpackage (from numpy import linalg), ανάλογα με τις ανάγκες του προγράμματος. Στον κώδικα Κ. 1.7.6 παρουσιάζεται ένα παράδειγμα όπου δημιουργείται ένας πίνακας A με χρήση της numpy, και στη συνέχεια αξιοποιείται το subpackage numpy.linalg για τον υπολογισμό του αντιστρόφου πίνακα μέσω της συνάρτησης linalg.inv(A). Στο συγκεκριμένο παράδειγμα, ο τελεστής @ χρησιμοποιείται για τον υπολογισμό γινομένου πινάκων, επιβεβαιώνοντας ότι το γινόμενο $A \cdot A^{-1}$ ισούται με τον μοναδιαίο πίνακα, όπως φαίνεται στην έξοδο Ε. 6. Για να χρησιμοποιηθεί το numpy θα πρέπει πρώτα να εγκατασταθεί με το pip (ή άλλο) όπως περιγράφηκε στην ενότητα .

```python
import numpy as np
from numpy import linalg

# αποφυγή επιστημονικής σημειογραφίας
np.set_printoptions(suppress=True)

A = np.array([[1, 2], [3, 4]])
print("A:\n", A)

# χρήση από το subpackage numpy.linalg
inv_A = linalg.inv(A)
print("A^-1:\n", inv_A)

I = A @ inv_A  # το γινόμενο A * A^-1 με τον τελεστή @
print("A x A^-1:\n", I)
```

Κ. 1.7.6 – Παράδειγμα χρήσης του package numpy.

```
A:
```

```
 [[1 2]
 [3 4]]
A^-1:
 [[-2.   1. ]
 [ 1.5 -0.5]]
A x A^-1:
 [[1. 0.]
 [0. 1.]]
```

Ε. 6 – Έξοδος κώδικα Κ. 1.7.6.

1.7.2.1 Δημιουργία ενός package από τον χρήστη

Η δημιουργία ενός package από τον χρήστη είναι μια σχετική απλή διαδικασία και βοηθά στην καλύτερη οργάνωση του κώδικα για μεγαλύτερες εφαρμογές. Ένα package είναι ένας φάκελος που περιέχει αρχεία .py και, κατά κανόνα, ένα αρχείο __init__.py, το οποίο δηλώνει ότι ο φάκελος πρέπει να αντιμετωπιστεί ως πακέτο. Η χρήση packages επιτρέπει τη λογική ομαδοποίηση του κώδικα με σκοπό τη διευκόλυνση της επαναχρησιμοποίησης και συντήρησης του κώδικα.

Στη συνέχεια παρουσιάζεται ένα παράδειγμα δημιουργίας ενός package με όνομα my_package που περιέχει δύο αρχεία .py: το αρχείο math_utils.py (κώδικας Κ. 1.7.7) και το αρχείο text_utils.py (κώδικας Κ. 1.7.8). Τα αρχεία αυτά υπάρχουν σε έναν φάκελο με όνομα my_package που επιπλέον περιέχει το αρχείο __init__.py το οποίο στο συγκεκριμένο παράδειγμα είναι κενό. Το my_package χρησιμοποιείται από ένα αρχείο main.py (κώδικας Κ. 1.7.9) το οποίο βρίσκεται στο ίδιο επίπεδο φακέλων με τον φάκελο my_package. Η δομή φακέλων και αρχείων που δημιουργείται είναι η ακόλουθη:

```
my_package/
      __init__.py
      math_utils.py
      text_utils.py
main.py
```

```python
def add(a, b):
    return a + b
```

Κ. 1.7.7 – Τα περιεχόμενα του math_utils.py.

```python
def whisper(message):
    return message.lower() + "..."
```

Κ. 1.7.8 – Τα περιεχόμενα του text_utils.py.

```python
from my_package import text_utils, math_utils

print(math_utils.add(2, 3))   # 5
print(text_utils.whisper("Hello"))   # "hello..."
```

Κ. 1.7.9 – Τα περιεχόμενα του main.py μαζί με την έξοδο που προκαλείται από τις κλήσεις συναρτήσεων ως σχόλια.

Εναλλακτικά, για την ίδια δομή αρχείων και φακέλων θα μπορούσε η __init__.py να έχει το περιεχόμενο που φαίνεται στον κώδικα Κ. 1.7.10, και η main.py να έχει ως περιεχόμενο τον κώδικα Κ. 1.7.11. Το αποτέλεσμα εκτέλεσης του main.py θα είναι και σε αυτή την περίπτωση το ίδιο.

```python
from .math_utils import add
```

```
from .text_utils import whisper
```
Κ. 1.7.10 – Εναλλακτική υλοποίηση: τα περιεχόμενα του __init__.py.

```
import my_package

print(my_package.math_utils.add(2, 3))   # 5
print(my_package.text_utils.whisper("Hello")) # "hello..."
```
Κ. 1.7.11 – Εναλλακτική υλοποίηση: τα περιεχόμενα του main.py.

1.8. Αντικειμενοστραφής προγραμματισμός

Η Python είναι μια γλώσσα πολλαπλών υποδειγμάτων (multi-paradigm). Αυτό σημαίνει ότι υποστηρίζει πολλούς τρόπους με τους οποίους μπορεί να επιτευχθεί η ανάπτυξη προγραμμάτων, εφαρμογών και μεγάλων πληροφοριακών συστημάτων. Δύο βασικοί τρόποι προγραμματισμού με την Python είναι ο διαδικασιακός προγραμματισμός (procedural programming) και ο αντικειμενοστραφής προγραμματισμός (object oriented programming). Ο διαδικασιακός προγραμματισμός οργανώνει τον κώδικα σε διαδικασίες/συναρτήσεις. Ο προγραμματισμός γίνεται με τη συγγραφή συναρτήσεων που επιλύουν μικρότερα υποπροβλήματα. Μέσω της επίλυσης των μικρότερων υποπροβλημάτων και του συνδυασμού τους επιδιώκεται η επίλυση μεγαλύτερων προβλημάτων μέχρι να επιτευχθεί η επίλυση του συνολικού προβλήματος. Από την άλλη πλευρά, ο αντικειμενοστραφής προγραμματισμός οργανώνει το πρόγραμμα γύρω από αντικείμενα (objects) που συνδυάζουν δεδομένα και συμπεριφορά στοχεύοντας στην επίτευξη υψηλότερου επιπέδου αφαίρεσης για την αποτελεσματικότερη μοντελοποίηση και επίλυση του προβλήματος. Με τον όρο αφαίρεση (abstraction), που αποτελεί μια από τις 4 βασικές αρχές του αντικειμενοστραφούς προγραμματισμού, εννοείται η διατήρηση μόνο των ουσιαστικών στοιχείων που σχετίζονται με την επίλυση του προβλήματος και η απόκρυψη των μη απαραίτητων λεπτομερειών υλοποίησης. Οι άλλες τρεις αρχές του αντικειμενοστραφούς προγραμματισμού είναι η ενθυλάκωση (encapsulation), η κληρονομικότητα (inheritance) και ο πολυμορφισμός (polymorphism), που θα περιγραφούν στη συνέχεια.

Ο αντικειμενοστραφής προγραμματισμός θεωρείται γενικά σήμερα ως ένα επιτυχημένο προγραμματιστικό υπόδειγμα (programming paradigm) που με προσεκτική και σωστή εφαρμογή μπορεί να οδηγήσει στην κατασκευή μεγάλων εφαρμογών που είναι σχετικά εύκολο να επεκταθούν και να διορθωθούν ενώ ο κώδικάς τους παραμένει κατανοητός. Καθώς η σύγχρονη ανάπτυξη λογισμικού εμπλέκει ομάδες προγραμματιστών που ανανεώνονται συχνά, η καλή γνώση των αρχών της αντικειμενοστραφούς σχεδίασης και του αντικειμενοστραφούς προγραμματισμού σε μια γλώσσα προγραμματισμού που τον υποστηρίζει (π.χ., Java, C++, Python) είναι πολλές φορές προαπαιτούμενο για την ένταξη ενός νέου μέλους σε μια ομάδα ανάπτυξης λογισμικού.

Συμπερασματικά, η Python είναι μια καθαρά αντικειμενοστραφής γλώσσα προγραμματισμού κατά την έννοια ότι όλα είναι αντικείμενα όπως φαίνεται στον κώδικα Κ. 1.8.1 που εκτελείται στο REPL και χρησιμοποιεί τη συνάρτηση type(). Επιπλέον, η Python υποστηρίζει τον διαδικασιακό προγραμματισμό και διαθέτει χαρακτηριστικά συναρτησιακού προγραμματισμού (π.χ., lambdas, closures, first-class functions).

```
>>> type(1)
<class 'int'>
>>> type('Γεια')
<class 'str'>
>>> type(sum)
<class 'builtin_function_or_method'>
>>> import math
>>> type(math)
<class 'module'>
>>> type(math.sqrt)
<class 'builtin_function_or_method'>
```

Κ. 1.8.1 – Στην Python όλα είναι αντικείμενα.

1.8.1. Κλάσεις και αντικείμενα

Δύο βασικές έννοιες του αντικειμενοστραφούς προγραμματισμού είναι η έννοια της κλάσης και η έννοια του αντικειμένου. Μια κλάση (class) αποτελεί ένα αφηρημένο πρότυπο (blueprint) ή ένα καλούπι με βάση το οποίο δημιουργούνται συγκεκριμένες οντότητες. Στην Python, η κλάση ορίζει τη δομή των δεδομένων (χαρακτηριστικά/attributes) και τη συμπεριφορά (μεθόδους/methods) που θα έχουν τα αντικείμενα που θα δημιουργηθούν από αυτήν.

Στο παράδειγμα του κώδικα Κ. 1.8.2 η κλάση Car ορίζει τι σημαίνει «αυτοκίνητο» μέσα στο πρόγραμμα. Περιλαμβάνει:

- Τρία χαρακτηριστικά που δίνονται κατά τη δημιουργία του αντικειμένου: make, model, year.
- Ένα επιπλέον χαρακτηριστικό odometer, το οποίο αρχικοποιείται στην τιμή 0.
- Δύο μεθόδους:
    - __str__, που εμφανίζει τα στοιχεία του αυτοκινήτου. Η μέθοδος __str__ είναι μια ειδική μέθοδος (magic method) που καθορίζει την αναπαράσταση του αντικειμένου σε μορφή συμβολοσειράς. Εκτελείται αυτόματα όταν καλείται η print() πάνω σε ένα αντικείμενο.
    - update_odometer(new_km), που ενημερώνει τα χιλιόμετρα με έλεγχο ώστε να μην επιτρέπεται μείωση της τιμής.

Η ειδική μέθοδος __init__ μπορεί να θεωρηθεί ως ένας κατασκευαστής (constructor) και εκτελείται αυτόματα όταν δημιουργείται ένα νέο αντικείμενο της κλάσης. Χρησιμοποιείται για την αρχικοποίηση των χαρακτηριστικών του αντικειμένου. Η παράμετρος self αναφέρεται στο ίδιο το αντικείμενο που δημιουργείται και επιτρέπει την πρόσβαση στα δεδομένα και στις μεθόδους του.

Ένα αντικείμενο είναι ένα συγκεκριμένο στιγμιότυπο (instance) μιας κλάσης. Δημιουργείται καλώντας την κλάση ως συνάρτηση. Στον κώδικα με την εντολή:

```
a_car = Car("Opel", "Corsa", 2024)
```

δημιουργείται ένα αντικείμενο της κλάσης Car, το οποίο ανατίθεται στη μεταβλητή a_car. Το αντικείμενο αυτό έχει συγκεκριμένες τιμές για τα χαρακτηριστικά του: μάρκα "Opel", μοντέλο "Corsa", έτος 2024 και αρχικά 0 χιλιόμετρα.

Η εντολή:

```
print(a_car)
```

προκαλεί έμμεσα την κλήση της __str__ για την εμφάνιση της «κατάστασης» του αντικειμένου, ενώ η κλήση:

```
a_car.update_odometer(1000)
```

ενημερώνει την εσωτερική κατάσταση του αντικειμένου. Όταν επιχειρείται:

```
a_car.update_odometer(900)
```

η μέθοδος απορρίπτει την αλλαγή, διατηρώντας τη συνέπεια των δεδομένων. Η λειτουργία της μεθόδου update_odometer αποτελεί παράδειγμα ενθυλάκωσης (encapsulation) και απόκρυψης πληροφορίας (information hiding), καθώς τα δεδομένα του αντικειμένου (π.χ. η τιμή του odometer) δεν τροποποιούνται άμεσα από εξωτερικό κώδικα, αλλά μόνο μέσω ελεγχόμενων μεθόδων της κλάσης, οι οποίες επιβάλλουν κανόνες και διασφαλίζουν τη συνέπεια της εσωτερικής κατάστασης του αντικειμένου.

Στην έξοδο Ε. 7 παρουσιάζεται το αποτέλεσμα της εκτέλεσης του κώδικα συνολικά.

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer = 0

    def __str__(self):
        return f"Αυτοκίνητο μάρκας {self.make}, μοντέλου {self.model},
έτους {self.year} με {self.odometer} χιλιόμετρα."

    def update_odometer(self, new_km):
        print(f"Το αυτοκίνητο έχει {self.odometer} χιλιόμετρα. Επιχειρείται
να αλλάξουν σε {new_km}.")
        if new_km > self.odometer:
            self.odometer = new_km
        else:
            print("Δεν επιτρέπεται να γυρίσουν πίσω τα χιλιόμετρα.")
```

```
a_car = Car("Opel", "Corsa", 2024)
print(a_car)
a_car.update_odometer(1000)
print(a_car)
a_car.update_odometer(900)
print(a_car)
```

Κ. 1.8.2 – Ένα εισαγωγικό παράδειγμα αντικειμενοστραφούς προγραμματισμού: η κλάση Car και το αντικείμενο a_car.

```
Αυτοκίνητο μάρκας Opel, μοντέλου Corsa, έτους 2024 με 0 χιλιόμετρα.
Το αυτοκίνητο έχει 0 χιλιόμετρα. Επιχειρείται να αλλάξουν σε 1000.
Αυτοκίνητο μάρκας Opel, μοντέλου Corsa, έτους 2024 με 1000 χιλιόμετρα.
Το αυτοκίνητο έχει 1000 χιλιόμετρα. Επιχειρείται να αλλάξουν σε 900.
Δεν επιτρέπεται να γυρίσουν πίσω τα χιλιόμετρα.
Αυτοκίνητο μάρκας Opel, μοντέλου Corsa, έτους 2024 με 1000 χιλιόμετρα.
```

Ε. 7 – Έξοδος που δημιουργείται κατά την εκτέλεση του κώδικα Κ. 1.8.2

1.8.2. Κληρονομικότητα

Η κληρονομικότητα επιτρέπει τον ορισμό μιας νέας κλάσης με βάση μια ήδη υπάρχουσα. Η νέα κλάση (υποκλάση – subclass) κληρονομεί τα χαρακτηριστικά και τις μεθόδους της αρχικής κλάσης (υπερκλάση – superclass), επεκτείνοντας ή εξειδικεύοντας τη συμπεριφορά της.

Στον κώδικα Κ. 1.8.3 παρουσιάζεται ένα παράδειγμα κληρονομικότητας με δύο κλάσεις, την υπερκλάση BankAccount και την υποκλάση SavingsAccount. Η κλάση BankAccount ορίζει τα βασικά χαρακτηριστικά ενός τραπεζικού λογαριασμού, δηλαδή τον κάτοχο (owner) και το υπόλοιπο (balance). Η κλάση SavingsAccount δηλώνεται ως υποκλάση της BankAccount, γεγονός που σημαίνει ότι κληρονομεί όλα τα χαρακτηριστικά της υπερκλάσης χωρίς να χρειάζεται να τα επαναπροσδιορίσει. Στον κατασκευαστή της υποκλάσης χρησιμοποιείται η συνάρτηση super(), ώστε να κληθεί ο κατασκευαστής της υπερκλάσης και να αρχικοποιηθούν τα κοινά χαρακτηριστικά. Επιπλέον, η SavingsAccount επεκτείνει τη βασική λειτουργικότητα προσθέτοντας το χαρακτηριστικό interest_rate και τη μέθοδο apply_interest(), η οποία υπολογίζει και προσθέτει τους τόκους στο υπόλοιπο του λογαριασμού. Με τον τρόπο αυτό, η υποκλάση εξειδικεύει τη συμπεριφορά της υπερκλάσης, υλοποιώντας έναν ειδικό τύπο τραπεζικού λογαριασμού που υποστηρίζει ανατοκισμό.

```
class BankAccount:
    def __init__(self, owner, balance=0):
        self.owner = owner
        self.balance = balance


class SavingsAccount(BankAccount):
    def __init__(self, owner, balance=0, interest_rate=0):
        super().__init__(owner, balance)
        self.interest_rate = interest_rate

    def apply_interest(self):
        self.balance += self.balance * self.interest_rate
```

```
savings = SavingsAccount("Γιάννης Παπαδόπουλος", 3000, 0.05)
print(f"Αρχικό υπόλοιπο: {savings.balance:.2f} ευρώ")

for i in range(5):
    savings.apply_interest()
    print(f"Μετά από την {i+1}η εφαρμογή τόκων: {savings.balance:.2f}
ευρώ")

print(f"Τελικό υπόλοιπο: {savings.balance:.2f} ευρώ")
```

Κ. 1.8.3 – Παράδειγμα κληρονομικότητας.

Ακολουθεί στην έξοδο Ε. 8 το αποτέλεσμα της εκτέλεσης του κώδικα.

```
Αρχικό υπόλοιπο: 3000.00 ευρώ
Μετά από την 1η εφαρμογή τόκων: 3150.00 ευρώ
Μετά από την 2η εφαρμογή τόκων: 3307.50 ευρώ
Μετά από την 3η εφαρμογή τόκων: 3472.88 ευρώ
Μετά από την 4η εφαρμογή τόκων: 3646.52 ευρώ
Μετά από την 5η εφαρμογή τόκων: 3828.84 ευρώ
Τελικό υπόλοιπο: 3828.84 ευρώ
```

Ε. 8 – Αποτέλεσμα εκτέλεσης κώδικα Κ. 1.8.3.

1.8.3. Πολυμορφισμός

Ο πολυμορφισμός (polymorphism) στον αντικειμενοστραφή προγραμματισμό είναι η ιδιότητα σύμφωνα με την οποία διαφορετικά αντικείμενα μπορούν να απαντούν στο ίδιο μήνυμα (δηλαδή στην ίδια κλήση μεθόδου) με διαφορετικό, κατάλληλο για αυτά τρόπο. Στην πράξη αυτό σημαίνει ότι ο ίδιος κώδικας, μπορεί να εκτελεί διαφορετική συμπεριφορά ανάλογα με τον τύπο του αντικειμένου στο οποίο εφαρμόζεται.

Στον κώδικα XXX παρουσιάζεται ένα παράδειγμα πολυμορφισμού από τον χώρο των video games. Ορίζεται η υπερκλάση Character, η οποία περιλαμβάνει τη μέθοδο attack(). Η μέθοδος αυτή δεν υλοποιεί συγκεκριμένη συμπεριφορά, αλλά θέτει την απαίτηση να υλοποιηθεί (να γίνει override) από τις υποκλάσεις. Οι κλάσεις Warrior και Archer κληρονομούν από την Character και επαναπροσδιορίζουν (υπερκαλύπτουν) τη μέθοδο attack() με διαφορετικό τρόπο: ο πολεμιστής επιτίθεται με σπαθί, ενώ ο τοξότης με βέλος. Η συνάρτηση play_turn() δέχεται μια λίστα από αντικείμενα τύπου Character και καλεί για καθένα τη μέθοδο attack() χωρίς να χρειάζεται να γνωρίζει τον ακριβή τύπο του αντικειμένου. Παρότι η κλήση είναι ίδια (character.attack()), η συμπεριφορά που εκτελείται εξαρτάται από το πραγματικό αντικείμενο (πολεμιστής ή τοξότης). Όπως φαίνεται στην έξοδο Ε. 9, στον ίδιο «γύρο παιχνιδιού» εμφανίζονται δύο διαφορετικά μηνύματα επίθεσης, δείχνοντας ότι η ίδια διεπαφή (intreface) οδηγεί σε διαφορετική εξειδικευμένη συμπεριφορά.

```
class Character:
    def __init__(self, name, health=100):
        self.name = name
        self.health = health
```

```
    def attack(self):
        # Βασική μέθοδος που αναμένεται να υπερκαλυφθεί (override)
        raise NotImplementedError(
            "Η μέθοδος attack πρέπει να υλοποιηθεί στις υποκλάσεις."
        )


class Warrior(Character):
    def attack(self):
        print(f"{self.name} επιτίθεται με σπαθί και προκαλεί 15 μονάδες
ζημιάς!")


class Archer(Character):
    def attack(self):
        print(f"{self.name} ρίχνει ένα βέλος και προκαλεί 12 μονάδες
ζημιάς!")


def play_turn(characters):
    print("== Γύρος παιχνιδιού ==")
    for character in characters:
        character.attack()


hero1 = Warrior("Valkar")
hero2 = Archer("Sylvaris")
party = [hero1, hero2]
play_turn(party)
```

Κ. 1.8.4 – Παράδειγμα πολυμορφισμού.

```
== Γύρος παιχνιδιού ==
Valkar επιτίθεται με σπαθί και προκαλεί 15 μονάδες ζημιάς!
Sylvaris ρίχνει ένα βέλος και προκαλεί 12 μονάδες ζημιάς!
```

Ε. 9 – Αποτέλεσμα εκτέλεσης κώδικα Κ. 1.8.4.

1.9. Αρχεία

Κατά την εκτέλεση ενός προγράμματος, τα δεδομένα αποθηκεύονται προσωρινά στην κύρια μνήμη του υπολογιστή, δηλαδή στη μνήμη RAM. Η RAM είναι πτητική μνήμη, που σημαίνει ότι το περιεχόμενό της χάνεται μόλις διακοπεί η τροφοδοσία ρεύματος. Επομένως, αν απαιτείται μόνιμη αποθήκευση δεδομένων τότε πρέπει να αποθηκευτούν σε κάποιο μέσο δευτερεύουσας αποθήκευσης. Αυτό επιτυγχάνεται μέσω αρχείων (files), τα οποία γράφονται σε αποθηκευτικά μέσα όπως σε σκληρούς δίσκους (HDD - Hard Disk Drives), σε δίσκους στερεάς κατάστασης (SSD - Solid State Drives), σε οπτικούς δίσκους (CD - Compact Discs) και σε άλλα μέσα αποθήκευσης. Επιπλέον, τα αρχεία οργανώνονται σε ιεραρχικές δομές φακέλων (directories), επιτρέποντας την εύκολη πρόσβαση στα δεδομένα.

Η διαχείριση ενός αρχείου σε ένα πρόγραμμα ακολουθεί συγκεκριμένη διαδικασία: αρχικά το αρχείο πρέπει να ανοιχθεί (open), καθορίζοντας τον τρόπο πρόσβασης, δηλαδή αν θα πραγματοποιηθεί ανάγνωση (read), εγγραφή (write) ή προσθήκη δεδομένων (append). Στη συνέχεια εκτελούνται οι

επιθυμητές ενέργειες επεξεργασίας, όπως ανάγνωση, εγγραφή ή τροποποίηση δεδομένων. Τέλος, το αρχείο πρέπει να κλείσει (close), ώστε να απελευθερωθούν οι πόροι του συστήματος και να διασφαλιστεί ότι τα δεδομένα έχουν αποθηκευτεί σωστά.

1.9.1. Αρχεία κειμένου

Τα αρχεία κειμένου (text files) αποτελούν τον απλούστερο και πιο διαδεδομένο τύπο αρχείων. Το περιεχόμενό τους είναι αναγνώσιμο από τον άνθρωπο και μπορεί να προβληθεί ή να επεξεργαστεί με έναν απλό επεξεργαστή κειμένου, όπως το Notepad, το Notepad++, το VS Code ή οποιοδήποτε αντίστοιχο εργαλείο. Τα δεδομένα σε ένα αρχείο κειμένου αποθηκεύονται ως ακολουθίες χαρακτήρων, συνήθως με κάποια κωδικοποίηση (όπως UTF-8), γεγονός που τα καθιστά εύκολα στη μεταφορά και στην ανταλλαγή μεταξύ διαφορετικών συστημάτων.

Συνήθως τα αρχεία κειμένου έχουν επέκταση .txt, ωστόσο πολλοί τύποι αρχείων που χρησιμοποιούνται στον προγραμματισμό και στην ανάλυση δεδομένων είναι επίσης αρχεία κειμένου, παρότι έχουν διαφορετική επέκταση. Παραδείγματα αποτελούν τα αρχεία CSV (Comma-Separated Values) για την αποθήκευση δεδομένων οργανωμένων σε στήλες τιμών, τα XML και JSON για δομημένη αναπαράσταση δεδομένων, καθώς και τα YAML αρχεία για ρυθμίσεις και παραμετροποίηση εφαρμογών. Παρόλο που η δομή τους μπορεί να είναι πιο σύνθετη από ένα απλό .txt, το περιεχόμενό τους παραμένει αναγνώσιμο και επεξεργάσιμο ως κείμενο. Επιπλέον, τα αρχεία πηγαίου κώδικα (source code) σε όλες τις γλώσσες προγραμματισμού, συμπεριλαμβανομένης της Python, είναι επίσης αρχεία κειμένου. Για παράδειγμα, τα αρχεία με επέκταση .py περιέχουν εντολές και ορισμούς σε μορφή απλού κειμένου.

1.9.1.1 Άνοιγμα αρχείου και ανάγνωση περιεχομένων

Το άνοιγμα ενός αρχείου πραγματοποιείται μέσω της συνάρτησης open(), η οποία δέχεται ως πρώτο όρισμα το όνομα (ή τη διαδρομή) του αρχείου και ως δεύτερο όρισμα την κατάσταση λειτουργίας (mode). Η κλήση της συνάρτησης επιστρέφει ένα αντικείμενο αρχείου, μέσω του οποίου μπορούν να διαβαστούν ή να εγγραφούν δεδομένα. Για παράδειγμα, η εντολή f = open(<όνομα_αρχείου>, "r") ανοίγει το αρχείο σε λειτουργία ανάγνωσης. Αν το αρχείο δεν υπάρχει και επιχειρηθεί άνοιγμα με "r", προκαλείται σφάλμα. Εφόσον ανοιχθεί το αρχείο, το διάβασμα του περιεχομένου του γίνεται με μεθόδους όπως είναι οι: read(), readline() ή readlines().

Στο παράδειγμα του κώδικα Κ. 1.9.1, υποτίθεται ότι υπάρχει ένα αρχείο lorem_ipsum.txt[1] στον ίδιο φάκελο με το πρόγραμμα. Στόχος είναι η καταμέτρηση των φωνηέντων που περιέχει το κείμενο. Για να συμβεί αυτό ορίζεται ένα σύνολο χαρακτήρων που αντιστοιχεί στα φωνήεντα (πεζά και κεφαλαία)

[1] Το περιεχόμενο του αρχείου lorem_ipsum.txt μπορεί να μεταφορτωθεί από το https://www.lipsum.com/.

και ένας μετρητής. Το αρχείο ανοίγει σε λειτουργία "r", διαβάζεται το περιεχόμενό του και πραγματοποιείται επανάληψη για κάθε χαρακτήρα του κειμένου. Εάν ο χαρακτήρας ανήκει στο σύνολο των φωνηέντων, ο μετρητής αυξάνεται κατά ένα. Μετά την ολοκλήρωση της επεξεργασίας, το αρχείο κλείνει με τη μέθοδο close() και εμφανίζεται το τελικό αποτέλεσμα.

```
filename = "lorem_ipsum.txt"
vowels = "aeiouAEIOU"
count = 0

f = open(filename, "r")
for line in f.read():
    for char in line:
        if char in vowels:
            count += 1
f.close()

print(f"Πλήθος φωνηέντων: {count}")  # Πλήθος φωνηέντων: 167
```

Κ. 1.9.1 – Άνοιγμα αρχείου για ανάγνωση.

Η ρητή κλήση της close() στον κώδικα Κ. 1.9.1 ενέχει τον κίνδυνο να παραλειφθεί ή να μην εκτελεστεί σε περίπτωση σφάλματος. Για τον λόγο αυτό, η προτεινόμενη πρακτική είναι η χρήση του λεγόμενου context manager με την εντολή with. Η σύνταξη with open(filename, "r") as f: εξασφαλίζει ότι το αρχείο θα κλείσει αυτόματα μόλις ολοκληρωθεί το αντίστοιχο μπλοκ κώδικα, ακόμη και αν προκύψει εξαίρεση κατά την εκτέλεση. Με αυτόν τον τρόπο, που ακολουθείται στον κώδικα Κ. 1.9.2, ο χειρισμός του αρχείου γίνεται πιο ασφαλής, και ακολουθείται μια από τις βέλτιστες πρακτικές της Python.

```
filename = "lorem_ipsum.txt"
vowels = "aeiouAEIOU"
count = 0

with open(filename, "r") as f:
    for line in f.read():
        for char in line:
            if char in vowels:
                count += 1

print(f"Πλήθος φωνηέντων: {count}")  # Πλήθος φωνηέντων: 167
```

Κ. 1.9.2 – Άνοιγμα αρχείου με context manager.

1.9.1.2 Εγγραφή δεδομένων σε αρχείο

Η εγγραφή δεδομένων σε αρχείο κειμένου αποτελεί βασική λειτουργία σε πολλές εφαρμογές, καθώς επιτρέπει την αποθήκευση αποτελεσμάτων για μελλοντική χρήση. Στο παράδειγμα που παρουσιάζεται στον κώδικα Κ. 1.9.3, ο χρήστης καλείται να εισαγάγει έναν θετικό ακέραιο αριθμό και το πρόγραμμα υπολογίζει την αντίστοιχη ακολουθία Collatz, αποθηκεύοντάς την σε αρχείο κειμένου με όνομα out.txt, μία τιμή ανά γραμμή. Η ακολουθία Collatz παράγεται επαναληπτικά: ξεκινά από τον δοθέντα θετικό ακέραιο και, αν ο αριθμός είναι άρτιος, διαιρείται ακέραια με το 2, ενώ αν είναι περιττός, πολλαπλασιάζεται με το 3 και προστίθεται το 1. Η διαδικασία

επαναλαμβάνεται μέχρι να προκύψει η τιμή 1. Αρχικά, το πρόγραμμα διαβάζει την είσοδο του χρήστη και ελέγχει αν πρόκειται πράγματι για θετικό ακέραιο αριθμό. Ο έλεγχος πραγματοποιείται με χρήση της μεθόδου isdigit() και με επιπλέον έλεγχο ότι η αριθμητική τιμή είναι μεγαλύτερη του μηδενός. Εάν η είσοδος δεν είναι έγκυρη, εμφανίζεται μήνυμα σφάλματος και το πρόγραμμα δεν προχωρά σε περαιτέρω επεξεργασία. Σε περίπτωση έγκυρης εισόδου, η τιμή μετατρέπεται σε ακέραιο και ανοίγει το αρχείο out.txt σε κατάσταση εγγραφής ("w"). Η χρήση της κατάστασης "w" σημαίνει ότι αν το αρχείο δεν υπάρχει θα δημιουργηθεί, ενώ αν υπάρχει θα διαγραφεί το προηγούμενο περιεχόμενό του. Το άνοιγμα γίνεται μέσω της with open("out.txt", "w") as f:, η οποία εξασφαλίζει ότι το αρχείο θα κλείσει αυτόματα μόλις ολοκληρωθεί το μπλοκ εντολών. Σε κάθε βήμα της διαδικασίας παραγωγής της ακολουθίας, η τρέχουσα τιμή του αριθμού γράφεται στο αρχείο με τη μέθοδο write(), συνοδευόμενη από χαρακτήρα αλλαγής γραμμής (\n), ώστε κάθε αριθμός να αποθηκεύεται σε ξεχωριστή γραμμή.

```
s = input("Εισάγετε έναν θετικό ακέραιο: ")

if not s.strip().isdigit() or int(s) <= 0:
    print("Μη έγκυρη είσοδος.")
else:
    n = int(s)
    with open("out.txt", "w") as f:
        while True:
            f.write(f"{n}\n")
            if n == 1:
                break
            if n % 2 == 0:
                n //= 2
            else:
                n = 3 * n + 1
    print("Η ακολουθία Collatz γράφτηκε στο out.txt")
```

Κ. 1.9.3 – Εγγραφή αριθμητικών τιμών σε αρχείο κειμένου.

1.9.2. Αρχεία CSV

Τα αρχεία CSV (Comma Separated Values) αποτελούν έναν από τους πιο διαδεδομένους τρόπους αποθήκευσης δομημένων δεδομένων σε μορφή πίνακα. Πρόκειται για αρχεία κειμένου στα οποία κάθε γραμμή αντιστοιχεί σε μία εγγραφή και οι τιμές των πεδίων διαχωρίζονται συνήθως με κόμμα (,), αν και μπορεί να χρησιμοποιηθεί και άλλος χαρακτήρας διαχωρισμού, όπως το tab ή το ερωτηματικό (;). Συχνά, αλλά όχι υποχρεωτικά, η πρώτη γραμμή του αρχείου περιέχει τις επικεφαλίδες των στηλών, δηλαδή τα ονόματα των μεταβλητών.

Η δημοτικότητα των αρχείων CSV οφείλεται στην απλότητά τους και στη σχεδόν καθολική υποστήριξή τους από λογιστικά φύλλα (όπως το Excel), συστήματα βάσεων δεδομένων, στατιστικά πακέτα και γλώσσες προγραμματισμού. Είναι αναγνώσιμα από τον άνθρωπο, μεταφέρονται εύκολα μεταξύ διαφορετικών συστημάτων και δεν απαιτούν σύνθετη υποδομή για την επεξεργασία τους. Στην

Python, η ανάγνωση και εγγραφή αρχείων CSV μπορεί να γίνει είτε με τη βιβλιοθήκη `pandas`, η οποία παρέχει υψηλού επιπέδου λειτουργίες, είτε με το ενσωματωμένο module `csv`, το οποίο ανήκει στην τυπική βιβλιοθήκη και είναι ελαφρύ και αποδοτικό.

Το module csv προσφέρει δύο βασικούς τρόπους ανάγνωσης ενός αρχείου CSV: μέσω της συνάρτησης reader() και μέσω της DictReader(). Η reader() επιστρέφει κάθε γραμμή ως λίστα τιμών, διατηρώντας τη σειρά των στηλών όπως εμφανίζονται στο αρχείο. Αντίθετα, η DictReader() επιστρέφει κάθε γραμμή ως λεξικό (dictionary), όπου τα κλειδιά είναι τα ονόματα των στηλών, που λαμβάνονται από την πρώτη γραμμή του αρχείου, και οι τιμές είναι τα αντίστοιχα δεδομένα της εγγραφής. Η προσέγγιση αυτή είναι ιδιαίτερα χρήσιμη όταν είναι επιθυμητή η πρόσβαση στα δεδομένα με βάση το όνομα της στήλης και όχι τη θέση της.

Στο παράδειγμα που παρουσιάζεται στη συνέχεια διαβάζεται ένα αρχείο με δεδομένα τιμών για την μετοχή της εταιρείας Google που αφορούν ένα συγκεκριμένο χρονικό διάστημα (Εικόνα 8). Στον κώδικα Κ. 1.9.4, χρησιμοποιείται η reader() για την ανάγνωση του αρχείου και τα ίδια δεδομένα εμφανίζονται ως λίστες, όπου η πρώτη γραμμή περιέχει τις επικεφαλίδες και οι επόμενες τις αντίστοιχες τιμές. Αντίστοιχα στον κώδικα Κ. 1.9.5, χρησιμοποιείται η DictReader(), για την ανάγνωση του αρχείου και κάθε εγγραφή εμφανίζεται ως λεξικό με πεδία όπως Date, Open, High, Low, Close και Volume. Και στις δύο προσεγγίσεις διαβάζονται και εκτυπώνονται μόνο οι δύο πρώτες γραμμές των δεδομένων. Η έξοδος για την περίπτωση της reader() παρουσιάζεται στην έξοδο Ε. 10 και για την περίπτωση της DictReader παρουσιάζεται στην έξοδο Ε. 11. Η κατανόηση αυτών των δύο προσεγγίσεων επιτρέπει την επιλογή της κατάλληλης μεθόδου ανάλογα με τις ανάγκες του προβλήματος. Σε απλές περιπτώσεις, η reader() είναι επαρκής και ταχύτερη, ενώ όταν απαιτείται εύκολη πρόσβαση στα δεδομένα ανά όνομα στήλης, η DictReader() προσφέρει μεγαλύτερη ευελιξία.

Google_Stock_Train (2010-2022).csv > data

```
1  Date,Open,High,Low,Close,Adj Close,Volume
2  2010-01-04,15.689439,15.753504,15.621622,15.684434,15.684434,78169752
3  2010-01-05,15.695195,15.711712,15.554054,15.615365,15.615365,120067812
4  2010-01-06,15.662162,15.662162,15.174174,15.221722,15.221722,158988852
5  2010-01-07,15.250250,15.265265,14.831081,14.867367,14.867367,256315428
```

Εικόνα 8 – Αρχείο CSV[2] με τιμές της μετοχής της εταιρείας Google για ένα χρονικό διάστημα.

```
import csv

fn = "Google_Stock_Train (2010-2022).csv"
with open(fn, newline='', encoding="utf-8") as f:
    reader = csv.reader(f)
```

[2] Το αρχείο CSV μπορεί να μεταφορτωθεί από το https://www.kaggle.com/datasets/alirezajavid1999/google-stock-2010-2023.

```
    for i, row in enumerate(reader):
        print(row)
        if i >= 2:
            break
```
Κ. 1.9.4 – Ανάγνωση δεδομένων από αρχείο CSV με τη μέθοδο reader().

```
['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']
['2010-01-04',   '15.689439',   '15.753504',   '15.621622',   '15.684434',
'15.684434', '78169752']
['2010-01-05',   '15.695195',   '15.711712',   '15.554054',   '15.615365',
'15.615365', '120067812']
```
Ε. 10 – Αποτέλεσμα εκτέλεσης ανάγνωσης αρχείου CSV με τη μέθοδο reader().

```
import csv

fn = "Google_Stock_Train (2010-2022).csv"
with open(fn, newline="", encoding="utf-8") as f:
    reader = csv.DictReader(f)
    for i, row in enumerate(reader):
        print(row)
        if i >= 1:
            break
```
Κ. 1.9.5 – Ανάγνωση δεδομένων από αρχείο CSV με τη μέθοδο DictReader().

```
{'Date': '2010-01-04', 'Open': '15.689439', 'High': '15.753504', 'Low':
'15.621622', 'Close': '15.684434', 'Adj Close': '15.684434', 'Volume':
'78169752'}
{'Date': '2010-01-05', 'Open': '15.695195', 'High': '15.711712', 'Low':
'15.554054', 'Close': '15.615365', 'Adj Close': '15.615365', 'Volume':
'120067812'}
```
Ε. 11 – Αποτέλεσμα εκτέλεσης ανάγνωσης αρχείου CSV με τη μέθοδο DictReader().

1.9.3. Αρχεία Excel

Τα αρχεία Excel αποτελούν έναν από τους πιο διαδεδομένους τρόπους αποθήκευσης και διαχείρισης δομημένων δεδομένων. Το Microsoft Excel παραμένει ιδιαίτερα δημοφιλές χάρη στην ευκολία χρήσης, τις δυνατότητες μορφοποίησης, τους ενσωματωμένους υπολογισμούς και τα εργαλεία οπτικοποίησης που παρέχει. Ωστόσο, όταν απαιτείται αυτοματοποίηση επαναλαμβανόμενων εργασιών ή επεξεργασία μεγάλου όγκου δεδομένων, η Python προσφέρει σημαντικά πλεονεκτήματα, επιτρέποντας τον συνδυασμό της ισχύος προγραμματισμού με τη δομή των αρχείων Excel.

Η ανάγνωση και η εγγραφή δεδομένων σε αρχεία Excel μπορεί να πραγματοποιηθεί εύκολα με τη βιβλιοθήκη pandas, η οποία παρέχει συναρτήσεις όπως read_excel() και to_excel() για τη μετατροπή των φύλλων εργασίας σε αντικείμενα DataFrame. Η προσέγγιση αυτή είναι ιδιαίτερα κατάλληλη για

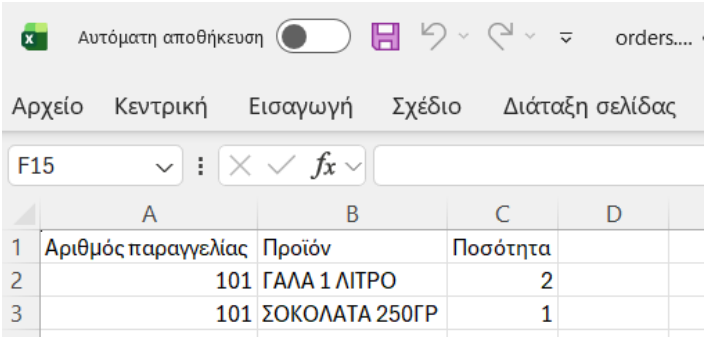
εργασίες ανάλυσης δεδομένων, καθώς επιτρέπει άμεση ενσωμάτωση των δεδομένων σε υπολογιστικά και στατιστικά μοντέλα. Εναλλακτικά, για χειρισμό αρχείων τύπου .xlsx, μπορεί να χρησιμοποιηθεί η εξωτερική βιβλιοθήκη openpyxl. Η βιβλιοθήκη αυτή είναι ελαφριά και εξειδικευμένη για ανάγνωση, εγγραφή και τροποποίηση αρχείων Excel (μορφή Excel 2010 και μεταγενέστερη). Απαιτεί εγκατάσταση μέσω pip, με την εντολή pip install openpyxl, αλλά δεν προϋποθέτει να είναι εγκατεστημένο το ίδιο το Excel στο σύστημα. Τα βασικά βήματα χειρισμού ενός αρχείου Excel με την openpyxl περιλαμβάνουν: (1) φόρτωση ενός αρχείου .xlsx, το οποίο στην ορολογία του Excel ονομάζεται workbook, (2) πρόσβαση σε ένα συγκεκριμένο φύλλο (sheet) του workbook, (3) ανάγνωση, εγγραφή ή τροποποίηση επιμέρους κελιών (cells) και (4) αποθήκευση των αλλαγών στο αρχείο. Η διαδικασία αυτή επιτρέπει ακριβή έλεγχο της δομής και του περιεχομένου του αρχείου, καθιστώντας δυνατή την αυτοματοποίηση πολύπλοκων εργασιών, όπως η δημιουργία αναφορών ή η ενημέρωση πινάκων με νέα δεδομένα.

Στο παράδειγμα που παρουσιάζεται στον κώδικα Κ. 1.9.6 αρχικά πραγματοποιείται η φόρτωση του αρχείου orders.xlsx (Εικόνα 9) ως αντικείμενο workbook και στη συνέχεια γίνεται πρόσβαση στο φύλλο με όνομα «Orders». Έπειτα, παρουσιάζονται βασικές λειτουργίες ανάγνωσης δεδομένων: εκτυπώνεται η πρώτη γραμμή του φύλλου που περιέχει τις επικεφαλίδες των στηλών και ανακτώνται τιμές από τα κελιά της δεύτερης και της τρίτης γραμμής, οι οποίες αντιστοιχούν σε στοιχεία μιας παραγγελίας. Στη συνέχεια, πραγματοποιείται εγγραφή νέων δεδομένων με την προσθήκη μιας νέας παραγγελίας στην τέταρτη γραμμή, καθώς και η τροποποίηση υπάρχουσας τιμής με την ενημέρωση της ποσότητας μιας ήδη καταχωρισμένης παραγγελίας. Τέλος, οι αλλαγές αποθηκεύονται σε νέο αρχείο (orders_updated.xlsx), όπως φαίνεται στην Εικόνα 10, διατηρώντας το αρχικό αρχείο αμετάβλητο.

```
from openpyxl import load_workbook

# 1. Φόρτωση αρχείου Excel (Workbook)
wb = load_workbook("orders.xlsx")
# 2. Πρόσβαση σε φύλλο (Sheet)
sheet = wb["Orders"]
# 3a. Ανάγνωση κελιών
print("Πρώτη σειρά του φύλλου:")
for cell in sheet[1]:  # 1η γραμμή (header)
    print(cell.value, end="\t")
print("\n")
# 3b. Ανάγνωση συγκεκριμένων κελιών
order_id = sheet["A2"].value
product = sheet["B2"].value
quantity = sheet["C2"].value
print(f"Order ID: {order_id}, Product: {product}, Quantity: {quantity}")
# 3c. Εγγραφή σε κελιά
sheet["A4"] = 102  # Αριθμός παραγγελίας
sheet["B4"] = "ΖΑΧΑΡΗ 1 ΚΙΛΟ"  # Προϊόν
sheet["C4"] = 1  # Ποσότητα
# 3d. Ενημέρωση κελιού
```

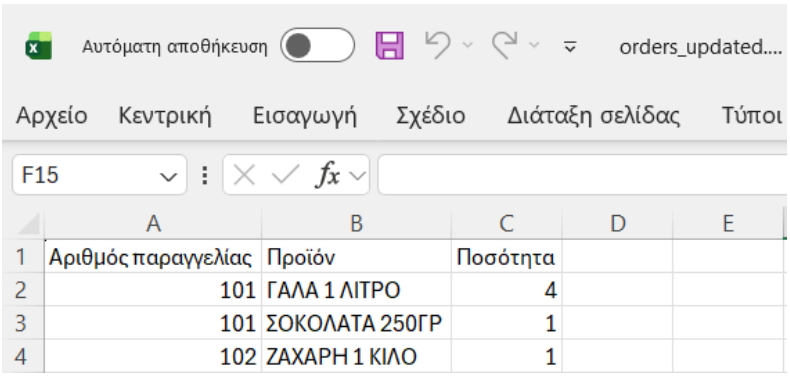
```
sheet["C2"] = quantity + 2  # αύξηση ποσότητας παραγγελίας
# 4. Αποθήκευση αλλαγών
wb.save("orders_updated.xlsx")
print("Οι αλλαγές αποθηκεύτηκαν στο 'orders_updated.xlsx'.")
```

Κ. 1.9.6 – Ανάγνωση αρχείου excel με δεδομένα παραγγελιών και δημιουργία νέου αρχείου excel.



	A	B	C	D
1	Αριθμός παραγγελίας	Προϊόν	Ποσότητα	
2	101	ΓΑΛΑ 1 ΛΙΤΡΟ	2	
3	101	ΣΟΚΟΛΑΤΑ 250ΓΡ	1	

Εικόνα 9 – Αρχείο excel από το οποίο γίνεται η ανάγνωση δεδομένων.



	A	B	C	D	E
1	Αριθμός παραγγελίας	Προϊόν	Ποσότητα		
2	101	ΓΑΛΑ 1 ΛΙΤΡΟ	4		
3	101	ΣΟΚΟΛΑΤΑ 250ΓΡ	1		
4	102	ΖΑΧΑΡΗ 1 ΚΙΛΟ	1		

Εικόνα 10 – Αρχείο excel που παράγεται κατά την εκτέλεση του κώδικα Κ. 1.9.6.

1.10. Βάσεις δεδομένων

Οι Σχεσιακές Βάσεις Δεδομένων αποτελούν το πλέον διαδεδομένο μοντέλο οργάνωσης και διαχείρισης δομημένων δεδομένων. Επιτρέπουν την οργανωμένη αποθήκευση πληροφοριών, καθώς και τη γρήγορη ανάκτηση και ενημέρωσή τους, ακόμη και όταν πρόκειται για μεγάλα σύνολα δεδομένων. Στο σχεσιακό μοντέλο, τα δεδομένα αποθηκεύονται σε πίνακες (tables), όπου κάθε πίνακας έχει διακριτό όνομα και αποτελείται από πεδία (στήλες) και εγγραφές (γραμμές). Κάθε πίνακας περιλαμβάνει ένα πεδίο ή έναν συνδυασμό πεδίων που ορίζεται ως πρωτεύον κλειδί (primary key), το οποίο αναγνωρίζει μοναδικά κάθε εγγραφή. Η ύπαρξη πρωτεύοντος κλειδιού διασφαλίζει τη μοναδικότητα και την ακεραιότητα των δεδομένων. Παράλληλα, οι πίνακες μπορούν να συσχετίζονται μεταξύ τους μέσω ξένων κλειδιών (foreign keys). Ένα ξένο κλειδί είναι ένα πεδίο ενός πίνακα που αναφέρεται στο πρωτεύον κλειδί ενός άλλου πίνακα, επιτρέποντας τη δημιουργία σχέσεων και τη διατήρηση συνέπειας μεταξύ των δεδομένων.

Η διαχείριση των σχεσιακών βάσεων δεδομένων πραγματοποιείται μέσω της γλώσσας SQL (Structured Query Language), η οποία αποτελεί πρότυπη γλώσσα ερωτημάτων για σχεσιακά συστήματα. Η SQL χρησιμοποιείται ευρέως από όλα τα δημοφιλή Συστήματα Διαχείρισης Βάσεων Δεδομένων, όπως Oracle, Microsoft SQL Server, IBM DB2, PostgreSQL, MySQL, MariaDB, DuckDB και SQLite. Μέσω της SQL παρέχεται η δυνατότητα δημιουργίας και διαγραφής βάσεων δεδομένων και πινάκων, εισαγωγής και ενημέρωσης δεδομένων, καθώς και ανάκτησης πληροφοριών με απλά ή σύνθετα ερωτήματα που μπορεί να περιλαμβάνουν συνδέσεις (joins) μεταξύ πολλών πινάκων. Επιπλέον, η SQL επιτρέπει τη διαχείριση χρηστών και την απόδοση ή την ανάκληση δικαιωμάτων πρόσβασης, στοιχείο κρίσιμο για την ασφάλεια και τον έλεγχο των δεδομένων σε οργανωμένα πληροφοριακά συστήματα. Ο Πίνακας 4 περιέχει μια σύνοψη σημαντικών εντολών SQL.

Εντολές	**Σύνταξη (γενική)**	**Παράδειγμα**
Δημιουργία Βάσης	`CREATE DATABASE όνομα;`	`CREATE DATABASE SchoolDB;`
Διαγραφή Βάσης	`DROP DATABASE όνομα;`	`DROP DATABASE SchoolDB;`
Δημιουργία Πίνακα	`CREATE TABLE όνομα_πίνακα (στήλη1 τύπος, στήλη2 τύπος, ...);`	`CREATE TABLE Students (ID INT, Name VARCHAR(50), Age INT);`
Διαγραφή Πίνακα	`DROP TABLE όνομα_πίνακα;`	`DROP TABLE Students;`
Τροποποίηση Πίνακα	`ALTER TABLE όνομα_πίνακα εντολή;`	`ALTER TABLE Students ADD Email VARCHAR(100);`
Εισαγωγή Εγγραφής	`INSERT INTO πίνακας (στήλες...) VALUES (τιμές...);`	`INSERT INTO Students (ID, Name, Age) VALUES (1, 'Maria', 20);`
Διαγραφή Εγγραφής	`DELETE FROM πίνακας WHERE συνθήκη;`	`DELETE FROM Students WHERE ID = 1;`
Ενημέρωση Εγγραφής	`UPDATE πίνακας SET στήλη = τιμή WHERE συνθήκη;`	`UPDATE Students SET Age = 21 WHERE Name = 'Maria';`
Επιλογή Δεδομένων	`SELECT στήλες FROM πίνακας WHERE συνθήκη;`	`SELECT Name, Age FROM Students WHERE Age > 18;`
Ανάθεση Δικαιωμάτων	`GRANT δικαιώματα ON βάση/πίνακας TO 'χρήστης'@'host';`	`GRANT SELECT, INSERT ON SchoolDB.* TO 'student_user'@'localhost';`

Πίνακας 4 – Ορισμένες σημαντικές εντολές SQL.

1.10.1. Sqlite

Η SQLite αποτελεί ένα ελαφρύ (lightweight) Σύστημα Διαχείρισης Σχεσιακών Βάσεων Δεδομένων, το οποίο έχει σχεδιαστεί με έμφαση στην απλότητα και τη φορητότητα. Το μέγεθος της βιβλιοθήκης της είναι μικρό (συνήθως μικρότερο από 1MB), γεγονός που την καθιστά ιδιαίτερα κατάλληλη για εφαρμογές με περιορισμένους πόρους. Σε αντίθεση με άλλα συστήματα βάσεων δεδομένων που λειτουργούν ως ξεχωριστές υπηρεσίες (server-based), η SQLite δεν δημιουργεί νέα διεργασία για την εκτέλεση των λειτουργιών της. Εκτελείται μέσα στην ίδια διεργασία (process) με την εφαρμογή που τη χρησιμοποιεί, υλοποιώντας ένα μοντέλο “serverless” αρχιτεκτονικής. Η βάση δεδομένων

αποθηκεύεται σε ένα και μόνο αρχείο, το οποίο μπορεί εύκολα να μεταφερθεί μεταξύ διαφορετικών λειτουργικών συστημάτων, όπως Windows, Linux ή Android, χωρίς ανάγκη ειδικής εγκατάστασης ή παραμετροποίησης. Η SQLite είναι ιδιαίτερα διαδεδομένη σε εφαρμογές για κινητές συσκευές (Android, iOS) και σε ενσωματωμένα (embedded) συστήματα, καθώς απαιτεί μηδενικές ρυθμίσεις (zero configuration) για να λειτουργήσει. Παρά τον ελαφρύ χαρακτήρα της, υποστηρίζει προχωρημένα χαρακτηριστικά σχεσιακών βάσεων δεδομένων, όπως transactions, triggers, indexes και views, διατηρώντας υψηλή ταχύτητα και αξιοπιστία.

1.10.1.1 Παράδειγμα με την sqlite

Στην ενότητα αυτή παρουσιάζεται η διαδικασία πρόσβασης και διαχείρισης μιας βάσης δεδομένων SQLite μέσω της Python, αξιοποιώντας το ενσωματωμένο module sqlite3. Η παρουσίαση οργανώνεται σε πέντε διαδοχικά βήματα που αναδεικνύουν τις βασικές λειτουργίες αλληλεπίδρασης μεταξύ εφαρμογής και βάσης δεδομένων.

Αρχικά, στον κώδικα Κ. 1.10.1, δημιουργείται η βάση δεδομένων με σύνδεση σε ένα αρχείο (π.χ. school.db) μέσω της εντολής sqlite3.connect(). Εφόσον το αρχείο δεν υπάρχει, η SQLite το δημιουργεί αυτόματα.

```
import sqlite3

conn = sqlite3.connect("school.db")
print("Η Βάση Δεδομένων δημιουργήθηκε και συνδέθηκε.")
conn.close()
```

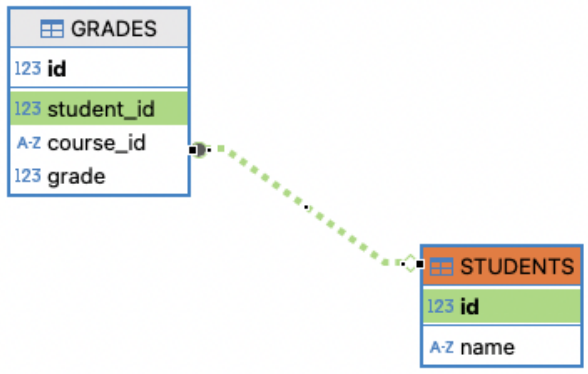
Κ. 1.10.1 – Δημιουργία της βάσης δεδομένων.

Στη συνέχεια, στον κώδικα Κ. 1.10.2, δημιουργούνται οι πίνακες STUDENTS και GRADES. Ο πίνακας STUDENTS περιλαμβάνει το πρωτεύον κλειδί (id) και το όνομα του φοιτητή, ενώ ο πίνακας GRADES περιλαμβάνει το δικό του πρωτεύον κλειδί (id) και ένα ξένο κλειδί (student_id) που αναφέρεται στο id του πίνακα STUDENTS, όπως φαίνεται στην Εικόνα 11. Με αυτόν τον τρόπο υλοποιείται η συσχέτιση των δύο πινάκων μέσω ξένου κλειδιού, διασφαλίζοντας τη σχεσιακή ακεραιότητα των δεδομένων.

```
import sqlite3

conn = sqlite3.connect("school.db")
cursor = conn.cursor()
cursor.execute(
    """
CREATE TABLE IF NOT EXISTS STUDENTS (
    id INTEGER PRIMARY KEY,
    name TEXT
)
"""
)
cursor.execute(
    """
```

```
CREATE TABLE IF NOT EXISTS GRADES (
    id INTEGER PRIMARY KEY,
    student_id INTEGER,
    course_id TEXT,
    grade INTEGER,
    FOREIGN KEY(student_id) REFERENCES STUDENTS(id)
)
"""
)
conn.commit()
conn.close()
print("Οι πίνακες STUDENTS και GRADES δημιουργήθηκαν.")
```

Κ. 1.10.2 – Δημιουργία πινάκων της βάσης δεδομένων και της συσχέτισης μεταξύ τους.

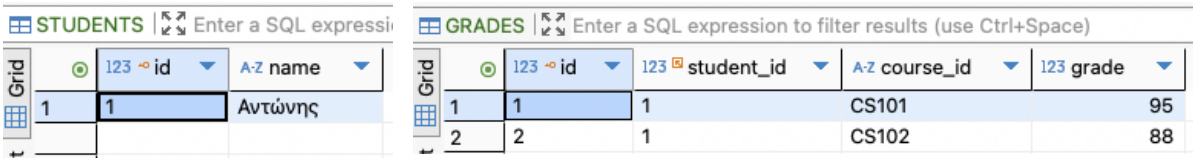


Εικόνα 11 – Η συσχέτιση ανάμεσα στον πίνακα STUDENTS και στον πίνακα GRADES.

Ακολούθως, στον κώδικα Κ. 1.10.3, πραγματοποιείται εισαγωγή εγγραφών με εντολές INSERT INTO. Εισάγεται ένας φοιτητής στον πίνακα STUDENTS και οι αντίστοιχες βαθμολογίες του στον πίνακα GRADES. Η χρήση παραμέτρων (?) στα ερωτήματα επιτρέπει την εκτέλεση εντολών SQL με την επιθυμητή διατύπωση. Το αποτέλεσμα της εισαγωγής των εγγραφών στη βάση δεδομένων φαίνεται στην Εικόνα 12.

```
import sqlite3

conn = sqlite3.connect("school.db")
cursor = conn.cursor()
cursor.execute("INSERT INTO STUDENTS (name) VALUES (?)", ("Αντώνης",))
cursor.execute(
    "INSERT INTO GRADES (student_id, course_id, grade) VALUES (?, ?, ?)",
    (1, "CS101", 95),
)
cursor.execute(
    "INSERT INTO GRADES (student_id, course_id, grade) VALUES (?, ?, ?)",
    (1, "CS102", 88),
)
conn.commit()
conn.close()
print("Εισήχθησαν εγγραφές στους πίνακες STUDENTS και GRADES.")
```

Κ. 1.10.3 – Εισαγωγή εγγραφών στους δύο πίνακες.



STUDENTS

	id	name
1	1	Αντώνης

GRADES

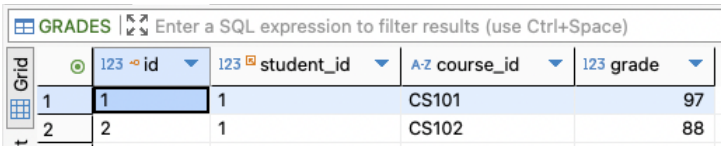
	id	student_id	course_id	grade
1	1	1	CS101	95
2	2	1	CS102	88

Εικόνα 12 – Οι πίνακες STUDENTS και GRADES μετά την εισαγωγή εγγραφών.

Έπειτα, στον κώδικα Κ. 1.10.4, πραγματοποιείται η ενημέρωση εγγραφής με εντολή UPDATE, όπου τροποποιείται η βαθμολογία ενός φοιτητή για ένα μάθημα, βάσει μιας συνθήκης στο WHERE που συνδυάζει το student_id και το course_id. Η αλλαγή οριστικοποιείται με την εντολή commit() και το αποτέλεσμά της φαίνεται στην εικόνα Εικόνα 13.

```
import sqlite3

conn = sqlite3.connect("school.db")
cursor = conn.cursor()
cursor.execute(
    """UPDATE GRADES SET grade = ?
WHERE student_id = ? AND course_id = ?""",
    (97, 1, "CS101"),
)
conn.commit()
conn.close()
print("Η εγγραφή βαθμολογίας ενημερώθηκε.")
```

Κ. 1.10.4 – Ενημέρωση εγγραφής στον πίνακα GRADES.



GRADES

	id	student_id	course_id	grade
1	1	1	CS101	97
2	2	1	CS102	88

Εικόνα 13 – Ο πίνακας GRADES μετά την ενημέρωση βαθμού (grade) για τον φοιτητή με student_id=1 για το μάθημα με course_id=CS101.

Τέλος, στον κώδικα Κ. 1.10.5, πραγματοποιείται ανάκτηση δεδομένων με εντολή SELECT και χρήση JOIN, ώστε να συνδυαστούν τα δεδομένα των πινάκων STUDENTS και GRADES. Το αποτέλεσμα επιστρέφει το όνομα του φοιτητή, τον κωδικό μαθήματος και τον βαθμό, επιδεικνύοντας τη ευκολία με την οποία ερωτήματα SQL μπορούν να εξάγουν σύνθετες πληροφορίες από συνδυασμό πολλών πινάκων.

```
import sqlite3

conn = sqlite3.connect("school.db")
cursor = conn.cursor()
cursor.execute(
    """
SELECT STUDENTS.name, GRADES.course_id, GRADES.grade
FROM STUDENTS
JOIN GRADES ON STUDENTS.id = GRADES.student_id
"""
)
for row in cursor.fetchall():
    print(row)
```

```
conn.close()
```
Κ. 1.10.5 – Ανάκληση δεδομένων από δύο πίνακες.

Η έξοδος του κώδικα είναι η ακόλουθη:

```
('Αντώνης', 'CS101', 97)
('Αντώνης', 'CS102', 88)
```

1.11. Εξαιρέσεις

Κατά την εκτέλεση ενός προγράμματος είναι πιθανό να προκύψουν σφάλματα (errors), τα οποία μπορεί να οφείλονται σε λανθασμένα δεδομένα εισόδου, σε ασυμβατότητες τύπων, σε προβλήματα αρχείων ή σε άλλες απρόβλεπτες καταστάσεις. Όταν συμβεί ένα σφάλμα, υπάρχουν διάφορες στρατηγικές αντιμετώπισης, όπως η αγνόησή του, ο άμεσος τερματισμός του προγράμματος, η εμφάνιση κατάλληλου μηνύματος και η συνέχιση της εκτέλεσης ή η προσπάθεια διόρθωσης/ανάκαμψης από το σφάλμα.

Στις σύγχρονες γλώσσες προγραμματισμού, ο βασικός μηχανισμός διαχείρισης λαθών είναι οι εξαιρέσεις (exceptions). Μια εξαίρεση σηματοδοτεί ότι συνέβη μια μη κανονική κατάσταση κατά την εκτέλεση. Η δημιουργία (πρόκληση) μιας εξαίρεσης γίνεται με την εντολή raise, η οποία διακόπτει τη φυσιολογική ροή εκτέλεσης και μεταφέρει τον έλεγχο στον κατάλληλο μηχανισμό χειρισμού. Στην Python, ο χειρισμός εξαιρέσεων υλοποιείται με τις εντολές try/except. Αυτό επιτρέπει το διαχωρισμό του κώδικα που ενδέχεται να προκαλέσει εξαίρεση από τον κώδικα που την αντιμετωπίζει. Η σύνταξη των εντολών try/except είναι η ακόλουθη:

```
try:
    # εντολές που μπορεί να προκαλέσουν εξαίρεση
except (Exception1, Exception2):
    # χειρισμός της εξαίρεσης
else:
    # εκτελείται αν δεν προκλήθηκε εξαίρεση
finally:
    # εκτελείται πάντα, ανεξάρτητα από το αν συνέβη εξαίρεση
```

Το μπλοκ try περιέχει τον κώδικα που ενδέχεται να προκαλέσει σφάλμα. Αν συμβεί εξαίρεση, εκτελείται το κατάλληλο μπλοκ except. Το μπλοκ else εκτελείται μόνο αν δεν προκύψει εξαίρεση, ενώ το finally εκτελείται πάντοτε και χρησιμοποιείται συχνά για καθαρισμό πόρων (π.χ. κλείσιμο αρχείων ή συνδέσεων βάσεων δεδομένων).

Ως ένα παράδειγμα σύλληψης και χειρισμού εξαιρέσεων παρουσιάζεται ο κώδικας Κ. 1.11.1 που επιχειρεί να ανοίξει ένα αρχείο κειμένου του οποίου το όνομα δίνεται από τον χρήστη, να διαβάσει τα περιεχόμενά του και να τα εμφανίσει στην οθόνη. Στο μπλοκ try γίνεται η προσπάθεια ανοίγματος του αρχείου σε λειτουργία ανάγνωσης και η ανάγνωση του περιεχομένου του. Αν το αρχείο δεν υπάρχει, προκαλείται εξαίρεση FileNotFoundError, η οποία συλλαμβάνεται από το αντίστοιχο μπλοκ

except και εμφανίζεται κατάλληλο μήνυμα σφάλματος. Αν το αρχείο υπάρχει αλλά δεν υπάρχουν τα απαραίτητα δικαιώματα πρόσβασης, προκαλείται PermissionError, η οποία επίσης χειρίζεται με ξεχωριστό μήνυμα. Το μπλοκ else εκτελείται μόνο εφόσον δεν προκύψει καμία εξαίρεση, δηλαδή όταν η ανάγνωση ολοκληρωθεί επιτυχώς. Τέλος, το μπλοκ finally εκτελείται σε κάθε περίπτωση, ανεξάρτητα από το αν συνέβη εξαίρεση ή όχι, επιτρέποντας την εκτέλεση κώδικα που πρέπει να πραγματοποιηθεί οπωσδήποτε (π.χ. ενημερωτικό μήνυμα που συμβαίνει σε αυτό το παράδειγμα ή αποδέσμευση πόρων).

```
try:
    filename = input("Δώσε όνομα αρχείου: ")
    f = open(filename, "r")
    content = f.read()
    print("Το αρχείο διαβάστηκε επιτυχώς.")
    f.close()
except FileNotFoundError:
    print("Σφάλμα: Το αρχείο δεν βρέθηκε.")
except PermissionError:
    print("Σφάλμα: Δεν υπάρχουν δικαιώματα πρόσβασης στο αρχείο.")
else:
    print("Η ανάγνωση ολοκληρώθηκε χωρίς σφάλματα.")
finally:
    print("Τέλος διαδικασίας.")
```

Κ. 1.11.1 – Άνοιγμα αρχείου κειμένου και εμφάνιση περιεχομένων του, με χειρισμό πιθανών εξαιρέσεων.

Συνεπώς, τα πιθανά σενάρια που μπορούν να συμβούν είναι τα ακόλουθα:

1. **Το αρχείο υπάρχει και ανοίγει κανονικά**: Η είσοδος του χρήστη αντιστοιχεί σε υπαρκτό αρχείο και υπάρχουν δικαιώματα ανάγνωσης του αρχείου. Το περιεχόμενο του αρχείου διαβάζεται επιτυχώς, εκτελείται το else, εκτελείται το finally.
2. **Το αρχείο δεν υπάρχει**: Η κλήση της open() προκαλεί FileNotFoundError. Εκτελείται το μπλοκ κώδικα του except FileNotFoundError, το else δεν εκτελείται, το finally εκτελείται.
3. **Δεν υπάρχουν δικαιώματα πρόσβασης στο αρχείο**: Η open() προκαλεί PermissionError. Εκτελείται το μπλοκ κώδικα του except PermissionError, το else δεν εκτελείται, το finally εκτελείται.
4. **Δημιουργείται ένα σφάλμα κατά την ανάγνωση του αρχείου**, που μπορεί να οφείλεται σε πρόβλημα του συστήματος αρχείων: Προκαλείται ένας άλλος τύπος εξαίρεσης (π.χ. OSError). Καθώς δεν υπάρχει αντίστοιχο except, το πρόγραμμα τερματίζεται εφόσον πρώτα εκτελεστεί το finally.

1.11.1. Ιεραρχία εξαιρέσεων

Στην Python, οι εξαιρέσεις υλοποιούνται ως στιγμιότυπα κλάσεων και οργανώνονται σε μια ιεραρχική δομή κληρονομικότητας. Η ιεραρχία αυτή ακολουθεί τη λογική από το γενικό προς το ειδικό. Στην κορυφή βρίσκεται η κλάση BaseException, ενώ οι περισσότερες συνηθισμένες εξαιρέσεις

που χρησιμοποιούνται στον προγραμματισμό κληρονομούν άμεσα ή έμμεσα από την κλάση Exception. Η ιεραρχική οργάνωση επιτρέπει την ομαδοποίηση συναφών εξαιρέσεων κάτω από κοινές υπερκλάσεις. Για παράδειγμα, εξαιρέσεις που σχετίζονται με αριθμητικά σφάλματα ή με σφάλματα εισόδου/εξόδου αποτελούν εξειδικεύσεις γενικότερων κατηγοριών. Αυτό δίνει τη δυνατότητα στον προγραμματιστή να επιλέγει το επίπεδο γενίκευσης με το οποίο θα χειριστεί ένα σφάλμα: είτε συλλαμβάνοντας μια συγκεκριμένη εξαίρεση είτε μια ευρύτερη κατηγορία.

Στον κώδικα Κ. 1.11.2 παρουσιάζεται ένα απλό παράδειγμα που αναδεικνύει τη χρήση της ιεραρχίας εξαιρέσεων. Οι ZeroDivisionError και ValueError είναι πιο εξειδικευμένες κλάσεις που κληρονομούν από την Exception. Το except Exception συλλαμβάνει οποιαδήποτε άλλη εξαίρεση που ανήκει στην ίδια ιεραρχία αλλά δεν έχει ήδη συλληφθεί. Η σειρά των except είναι σημαντική. Οι πιο ειδικές εξαιρέσεις τοποθετούνται πριν από τις γενικότερες, ώστε να μην «σκεπάζονται» από αυτές.

```
try:
    x = int(input("Δώσε έναν ακέραιο: "))
    y = 1 / x
    print("Αποτέλεσμα:", y)
except ZeroDivisionError:
    print("Σφάλμα: Διαίρεση με το μηδέν.")
except ValueError:
    print("Σφάλμα: Μη έγκυρη αριθμητική είσοδος.")
except Exception:
    print("Γενικό σφάλμα που ανήκει στην ιεραρχία της Exception.")
finally:
    print("Τέλος εκτέλεσης.")
```

Κ. 1.11.2 – Παράδειγμα χειρισμού εξαιρέσεων με αξιοποίηση της ιεραρχίας κλάσεων εξαιρέσεων στην Python.

Ο προγραμματιστής μπορεί είτε να χρησιμοποιήσει τις υπάρχουσες κλάσεις εξαιρέσεων που παρέχει η Python είτε να δημιουργήσει δικές του κλάσεις που κληρονομούν από υπάρχουσες, επεκτείνοντας έτσι το μηχανισμό χειρισμού σφαλμάτων με τρόπο προσαρμοσμένο κάθε φορά στις ανάγκες της εφαρμογής.

1.11.2. Ρητή πρόκληση εξαιρέσεων με το raise

Η εντολή raise χρησιμοποιείται για την ρητή πρόκληση (raise) μιας εξαίρεσης κατά την εκτέλεση ενός προγράμματος. Μέσω της raise, ο προγραμματιστής μπορεί να διακόψει τη φυσιολογική ροή εκτέλεσης όταν διαπιστωθεί ότι παραβιάζεται κάποια λογική προϋπόθεση ή όταν εντοπιστεί μη αποδεκτή κατάσταση. Η εντολή μπορεί να χρησιμοποιηθεί είτε με ενσωματωμένες εξαιρέσεις (π.χ. ValueError, TypeError) είτε με προσαρμοσμένες εξαιρέσεις που έχουν οριστεί από τον χρήστη. Στο παράδειγμα του κώδικα Κ. 1.11.3 υπάρχουν και οι δύο περιπτώσεις καθώς ορίζεται και προκαλείται μια προσαρμοσμένη εξαίρεση, σε συνδυασμό με πρόκληση ενσωματωμένων εξαιρέσεων και το μηχανισμό χειρισμού τους.

Αρχικά, ορίζεται η κλάση NoPositiveNumbersError, η οποία κληρονομεί από την ενσωματωμένη κλάση Exception. Η κλάση αυτή χρησιμοποιείται για να σηματοδοτήσει μια ειδική λογική συνθήκη σφάλματος: την απουσία θετικών αριθμών σε ένα σύνολο δεδομένων. Με τον τρόπο αυτό, διαχωρίζεται ένα σφάλμα επιχειρησιακής λογικής από γενικά σφάλματα. Η συνάρτηση average_positive(numbers) δέχεται ως όρισμα μια λίστα αριθμών και υπολογίζει τον μέσο όρο μόνο των θετικών στοιχείων της. Πριν τον υπολογισμό πραγματοποιούνται δύο έλεγχοι: α) αν η λίστα είναι κενή, οπότε προκαλείται εξαίρεση ValueError, καθώς δεν είναι δυνατόν να υπολογιστεί μέσος όρος χωρίς δεδομένα και β) αν υπάρχουν στοιχεία αλλά κανένα δεν είναι θετικό, προκαλείται η προσαρμοσμένη εξαίρεση NoPositiveNumbersError, με κατάλληλο μήνυμα. Αν ωστόσο οι προϋποθέσεις ικανοποιούνται, η συνάρτηση επιστρέφει τον μέσο όρο των θετικών τιμών.

Στο κύριο τμήμα του προγράμματος, ο χρήστης εισάγει ακέραιους αριθμούς χωρισμένους με κενό. Οι τιμές μετατρέπονται σε ακέραιους με χρήση της map(int, ...). Η κλήση της average_positive() τοποθετείται μέσα σε μπλοκ try, ώστε να είναι δυνατή η σύλληψη των πιθανών εξαιρέσεων. Αν η μετατροπή αποτύχει (π.χ. μη αριθμητική είσοδος), συλλαμβάνεται η ValueError. Αν δεν υπάρχουν θετικοί αριθμοί, συλλαμβάνεται η προσαρμοσμένη NoPositiveNumbersError. Σε κάθε περίπτωση, το μπλοκ finally εκτελείται υποχρεωτικά, εμφανίζοντας μήνυμα τερματισμού.

```
class NoPositiveNumbersError(Exception):
    """Εξαίρεση που προκαλείται όταν δεν υπάρχουν θετικοί αριθμοί."""
    pass


def average_positive(numbers):
    if not numbers:
        raise ValueError("Η λίστα είναι κενή.")
    positive_nums = [x for x in numbers if x > 0]
    if not positive_nums:
        raise NoPositiveNumbersError(
            "Δεν υπάρχουν θετικοί αριθμοί για υπολογισμό μέσου όρου."
        )
    return sum(positive_nums) / len(positive_nums)


try:
    data = list(map(int, input("Δώσε ακέραιους χωρισμένους με κενό:
").split()))
    avg = average_positive(data)
    print("Μέσος όρος θετικών αριθμών:", avg)
except ValueError as e:
    print("Σφάλμα εισόδου:", e)
except NoPositiveNumbersError as e:
    print("Προσαρμοσμένη εξαίρεση:", e)
finally:
    print("Τέλος εκτέλεσης.")
```

Κ. 1.11.3 – Παράδειγμα με πρόκληση ενσωματωμένων εξαιρέσεων και προσαρμοσμένων εξαιρέσεων και με τον χειρισμό τους.

Ακολουθούν 4 παραδείγματα εκτέλεσης του κώδικα Κ. 1.11.3. Στο πρώτο (Ε. 12) η εκτέλεση δεν προκαλεί καμία εξαίρεση και η εκτέλεση ολοκληρώνεται κανονικά.

```
Δώσε ακέραιους χωρισμένους με κενό: 5 -3 10 2
Μέσος όρος θετικών αριθμών: 5.666666666666667
Τέλος εκτέλεσης.
```

Ε. 12 – Κανονική ροή εκτέλεσης.

Στο δεύτερο παράδειγμα εκτέλεσης (Ε. 13) προκαλείται ValueError καθώς ο χρήστης εισάγει συμβολοσειρά αντί για ακέραιο.

```
Δώσε ακέραιους χωρισμένους με κενό: 4 7 a 9
Σφάλμα εισόδου: invalid literal for int() with base 10: 'a'
Τέλος εκτέλεσης.
```

Ε. 13 – Πρόκληση εξαίρεσης ValueError.

Στο τρίτο παράδειγμα εκτέλεσης (Ε. 14) προκαλείται η προσαρμοσμένη εξαίρεση NoPositiveNumbersError καθώς δεν υπάρχει θετική τιμή στις τιμές που εισήγαγε ο χρήστης.

```
Δώσε ακέραιους χωρισμένους με κενό: -5 -2 0 -8
Προσαρμοσμένη εξαίρεση: Δεν υπάρχουν θετικοί αριθμοί για υπολογισμό μέσου
όρου.
Τέλος εκτέλεσης.
```

Ε. 14 – Πρόκληση της προσαρμοσμένης εξαίρεσης NoPositiveNumbersError.

Στο τελευταίο παράδειγμα εκτέλεσης (Ε. 15) προκαλείται ρητά ValueError καθώς η λίστα τιμών που εισάγει ο χρήστης είναι κενή.

```
Δώσε ακέραιους χωρισμένους με κενό:
Σφάλμα εισόδου: Η λίστα είναι κενή.
Τέλος εκτέλεσης.
```

Ε. 15 – Ρητή πρόκληση της εξαίρεσης ValueError μέσα από τη συνάρτηση average_positive().

1.12. Κανονικές εκφράσεις

Οι κανονικές εκφράσεις (regular expressions) είναι ένας τυπικός μηχανισμός περιγραφής μοτίβων (patterns) σε κείμενο. Μια κανονική έκφραση αποτελείται από μια ακολουθία χαρακτήρων που ορίζει έναν κανόνα αναζήτησης, επιτρέποντας τον εντοπισμό, την επαλήθευση ή τη μετατροπή τμημάτων ενός κειμένου που έχουν συγκεκριμένη μορφή. Ειδικότερα, οι κανονικές εκφράσεις χρησιμοποιούνται ευρέως για:

- αναζήτηση μοτίβων μέσα σε κείμενο και εντοπισμό όλων των εμφανίσεών τους,
- έλεγχο εγκυρότητας μορφής δεδομένων (π.χ. emails, αριθμοί τηλεφώνου, ταχυδρομικοί κώδικες),
- αντικατάσταση υποσυμβολοσειρών που ταιριάζουν με συγκεκριμένο μοτίβο,
- διαχωρισμό κειμένου με βάση κάποιο πρότυπο διαχωριστικό.

Στην Python, η υποστήριξη για κανονικές εκφράσεις παρέχεται από το ενσωματωμένο (built-in) module re, το οποίο δεν απαιτεί εγκατάσταση και χρησιμοποιείται απλώς με import re. Παρότι υπάρχουν και βιβλιοθήκες τρίτων κατασκευστών (όπως regex ή re2), το re καλύπτει τις περισσότερες συνήθεις ανάγκες.

Οι βασικές συναρτήσεις του module re είναι οι εξής:

- re.match(pattern, text): ελέγχει αν το κείμενο ξεκινά με το συγκεκριμένο μοτίβο.
- re.search(pattern, text): εντοπίζει την πρώτη εμφάνιση του μοτίβου οπουδήποτε μέσα στο κείμενο.
- re.findall(pattern, text): επιστρέφει λίστα με όλες τις εμφανίσεις του μοτίβου.
- re.finditer(pattern, text): λειτουργεί όπως το findall, αλλά επιστρέφει iterator με αντικείμενα match, επιτρέποντας πιο λεπτομερή επεξεργασία.
- re.sub(pattern, repl, text): αντικαθιστά όλες τις εμφανίσεις του μοτίβου με την τιμή repl.
- re.split(pattern, text): διαχωρίζει το κείμενο χρησιμοποιώντας το μοτίβο ως διαχωριστικό.

Στον Πίνακας 5 παρουσιάζονται τα βασικά σύμβολα των κανονικών εκφράσεων και στον Πίνακας 6 οι ποσοδείκτες και ο τρόπος ορισμού ομάδων. Στον Πίνακας 7 και στον Πίνακας 8 παρουσιάζονται οι τρόπο με τους οποίους μπορούν να οριστούν σύνολα χαρακτήρων και όρια λέξεων αντίστοιχα.

Μοτίβο	**Περιγραφή**	**Παράδειγμα**	**Αποτέλεσμα**
`.`	Οποιοσδήποτε χαρακτήρας (εκτός newline)	`re.findall(r"a.c", "abc acb a-c a1c")`	`['abc', 'a-c', 'a1c']`
`^`	Αρχή κειμένου	`re.search(r"^Hello", "Hello World")`	`<re.Match object; span=(0, 5), match='Hello'>`
`$`	Τέλος κειμένου	`re.search(r"World$", "Hello World")`	`<re.Match object; span=(6, 11), match='World'>`
`\d`	Ψηφίο	`re.findall(r"\d", "A1B2")`	`['1', '2']`
`\D`	Μη ψηφίο	`re.findall(r"\D", "A1B2")`	`['A', 'B']`
`\w`	Χαρακτήρας λέξης (γράμμα ή αριθμός ή κάτω παύλα)	`re.findall(r"\w+", "Hi_123!")`	`['Hi_123']`
`\W`	Όχι χαρακτήρας λέξης	`re.findall(r"\W+", "Hi_123!")`	`['!']`
`\s`	Κενό διάστημα	`re.findall(r"\s", "Hello World")`	`[' ']`
`\S`	Όχι κενό διάστημα	`re.findall(r"\S+", "Hello World")`	`['Hello', 'World']`

Πίνακας 5 – Κανονικές εκφράσεις: βασικά σύμβολα.

Μοτίβο	**Περιγραφή**	**Παράδειγμα**	**Αποτέλεσμα**

*	Μηδέν ή περισσότερες εμφανίσεις	`re.findall(r"a*", "aaab")`	`['aaa', '']`
+	Μία ή περισσότερες εμφανίσεις	`re.findall(r"a+", "aaab")`	`['aaa']`
?	Μηδέν ή μία εμφάνιση	`re.findall(r"a?", "aaab")`	`['a', 'a', 'a', '', '']`
{m}	Ακριβώς m εμφανίσεις	`re.findall(r"a{2}", "aaab")`	`['aa']`
{m,n}	Από m έως n εμφανίσεις	`re.findall(r"a{2,3}", "aaaa")`	`['aaa']`
{m,}	Τουλάχιστον m εμφανίσεις	`re.findall(r"a{2,}", "aaaa")`	`['aaaa']`
(...)	Ομάδα για αναφορά	`re.findall(r"(ab)+", "abab")`	`['ab']`

Πίνακας 6 – Κανονικές εκφράσεις: ποσοδείκτες και ομάδες.

Μοτίβο	**Περιγραφή**	**Παράδειγμα**	**Αποτέλεσμα**
[abc]	Οποιοσδήποτε χαρακτήρας από αυτούς που βρίσκονται μέσα στις αγκύλες	`re.findall(r"[aeiou]", "hello")`	`['e', 'o']`
[^abc]	Οποιοσδήποτε χαρακτήρας εκτός από αυτούς που βρίσκονται μέσα στις αγκύλες	`re.findall(r"[^0-9]", "A1B2")`	`['A', 'B']`

Πίνακας 7 – Κανονικές εκφράσεις: σύνολα χαρακτήρων.

Μοτίβο	**Περιγραφή**	**Παράδειγμα**	**Αποτέλεσμα**
\b	Όριο λέξης	`re.findall(r"\bword\b", "a word wordy")`	`['word']`
\B	Μη όριο λέξης	`re.findall(r"\Bend", "bend send")`	`['end', 'end']`

Πίνακας 8 – Κανονικές εκφράσεις: όρια λέξεων.

1.12.1. Παράδειγμα με κανονικές εκφράσεις

Ως ένα παράδειγμα χρήσης κανονικών εκφράσεων παρουσιάζεται το πρόβλημα εντοπισμού όλων των ημερομηνιών σε ένα κείμενο, δεδομένου ότι οι ημερομηνίες καταγράφονται με 1 ή 2 ψηφία για τον αριθμό ημέρας και τον αριθμό μήνα και 2 ή 4 ψηφία για το έτος ενώ το διαχωριστικό ημερών, μήνα και έτους μπορεί να είναι «/» ή «.» ή «-».

Ο κώδικας Κ. 1.12.1 επιλύει το πρόβλημα ορίζοντας αρχικά το μοτίβο `\b\d{1,2}[./-]\d{1,2}[./-](?:\d{2}|\d{4})\b` για τις θεωρούμενες ως αποδεκτές ημερομηνίες. Το αρχικό και τελικό `\b` ορίζουν όρια λέξης ώστε να αναγνωρίζονται πλήρεις ημερομηνίες και όχι τμήματα μεγαλύτερων συμβολοσειρών. Το `\d{1,2}` δηλώνει ένα ή δύο ψηφία για την ημέρα και αντίστοιχα

για τον μήνα, ενώ η κλάση χαρακτήρων `[./-]` επιτρέπει ως διαχωριστικό ένα από τα: τελεία, κάθετο / ή παύλα -. Το τμήμα `(?:\d{2}|\d{4})` αποτελεί μη αποθηκευτική ομάδα επιλογής (non-capturing group) και σημαίνει ότι το έτος μπορεί να αποτελείται είτε από δύο είτε από τέσσερα ψηφία. Συνολικά, η έκφραση αναγνωρίζει συντακτικά έγκυρες ημερομηνίες με ευελιξία στη μορφή γραφής, χωρίς όμως να ελέγχει τη λογική εγκυρότητα των τιμών (π.χ. εύρος μήνα ή ημέρας).

```python
import re

text = """
Η συνάντηση θα γίνει στις 5-9-2025 και
η επόμενη στις 12.10.25. Μια άλλη
επιλογή είναι οι συναντήσεις να γίνουν
1/10/25 και 17/10/2025 αντίστοιχα.
"""
date_pattern = r"\b\d{1,2}[./-]\d{1,2}[./-](?:\d{2}|\d{4})\b"
matches = re.findall(date_pattern, text)
for i, match in enumerate(matches):
    print(f"Εμφάνιση ημερομηνίας {i+1}: {match}")
```

Κ. 1.12.1 – Εντοπισμός ημερομηνιών με συγκεκριμένες μορφές σε κείμενο.

Η έξοδος Ε. 16 αποτελεί το αποτέλεσμα της εκτέλεσης.

```
Εμφάνιση ημερομηνίας 1: 5-9-2025
Εμφάνιση ημερομηνίας 2: 12.10.25
Εμφάνιση ημερομηνίας 3: 1/10/25
Εμφάνιση ημερομηνίας 4: 17/10/2025
```

Ε. 16 – Αποτέλεσμα εκτέλεσης κώδικα Κ. 1.12.1.

Μια παραλλαγή της παραπάνω λύσης είναι ο κώδικας Κ. 1.12.2, που με τη χρήση ομάδων επιτρέπει τη σύλληψη ξεχωριστά της ημέρας, του μήνα και του έτους για τις ημερομηνίες του κειμένου. Η έξοδος Ε. 17 είναι το αποτέλεσμα της εκτέλεσης σε αυτή την περίπτωση.

```python
import re

text = """
Η συνάντηση θα γίνει στις 5-9-2025 και
η επόμενη στις 12.10.25. Μια άλλη
επιλογή είναι οι συναντήσεις να γίνουν
1/10/25 και 17/10/2025 αντίστοιχα.
"""
date_pattern = r"\b(\d{1,2})[./-](\d{1,2})[./-](\d{2}|\d{4})\b"
matches = re.findall(date_pattern, text)
for day, month, year in matches:
    print(f"Ημέρα: {day}, Μήνας: {month}, Έτος: {year}")
```

Κ. 1.12.2 – Εντοπισμός ημερομηνιών με συγκεκριμένες μορφές σε κείμενο, με χρήση ομάδων.

```
Ημέρα: 5, Μήνας: 9, Έτος: 2025
Ημέρα: 12, Μήνας: 10, Έτος: 25
Ημέρα: 1, Μήνας: 10, Έτος: 25
Ημέρα: 17, Μήνας: 10, Έτος: 2025
```

Ε. 17 – Αποτέλεσμα εκτέλεσης κώδικα Κ. 1.12.2

1.13. Το module sys

Στην ενότητα αυτή παρουσιάζονται βασικές πληροφορίες για το module sys της Python, το οποίο παρέχει πρόσβαση σε στοιχεία του περιβάλλοντος εκτέλεσης του προγράμματος και σε μηχανισμούς αλληλεπίδρασης με το λειτουργικό σύστημα.

Το module sys διαθέτει τρία βασικά αντικείμενα που σχετίζονται με τα ρεύματα εισόδου και εξόδου:

- sys.stdin, που αντιστοιχεί στην τυπική είσοδο (standard input),
- sys.stdout, που αντιστοιχεί στην τυπική έξοδο (standard output),
- sys.stderr, που χρησιμοποιείται για την έξοδο μηνυμάτων σφαλμάτων (standard error).

Τα ρεύματα αυτά μπορούν να ανακατευθυνθούν είτε κατά την εκτέλεση του προγράμματος από τη γραμμή εντολών είτε μέσα από τον ίδιο τον κώδικα. Για παράδειγμα, ένα πρόγραμμα μπορεί να λάβει είσοδο από αρχείο και να εξάγει αποτελέσματα και σφάλματα σε διαφορετικά αρχεία με εντολή της μορφής:

```
python my_script.py < 1.in > 1.out 2 > 1.err
```

Στην περίπτωση αυτή, το stdin ανακατευθύνεται από το αρχείο 1.in, το stdout στο αρχείο 1.out και το stderr στο αρχείο 1.err.

Επιπλέον, το module sys περιλαμβάνει χρήσιμα attributes, όπως:

- sys.path, που περιέχει τη λίστα των φακέλων στους οποίους αναζητά η Python modules προς εισαγωγή (import),
- sys.argv, που αποθηκεύει τα ορίσματα γραμμής εντολών,
- sys.version, που παρέχει πληροφορία για την τρέχουσα έκδοση της Python,
- sys.platform, που δηλώνει το λειτουργικό σύστημα στο οποίο εκτελείται το πρόγραμμα.

Ιδιαίτερη σημαντικό είναι το sys.argv, το οποίο είναι λίστα συμβολοσειρών που περιέχει τα ορίσματα με τα οποία κλήθηκε το πρόγραμμα. Το πρώτο στοιχείο (sys.argv[0]) είναι το όνομα του ίδιου του προγράμματος που εκτελείται, ενώ τα υπόλοιπα στοιχεία αντιστοιχούν στα ορίσματα που δόθηκαν από τον χρήστη. Έτσι, εάν για παράδειγμα ο κώδικας Κ. 1.13.1, έχει αποθηκευτεί σε ένα αρχείο με όνομα my_script.py και εκτελεστεί με την ακόλουθη εντολή:

```
python my_script.py ένα δύο 3
```

τότε το sys.argv[0] θα είναι "my_script.py" και το sys.argv[1:] θα περιέχει τα στοιχεία ["ένα", "δύο", "3"].

```
import sys
```

```
print("Όνομα script:", sys.argv[0])
print("Ορίσματα:", sys.argv[1:])
```
Κ. 1.13.1 – Περιεχόμενα my_script.py.

Τέλος, το sys.exit() επιτρέπει τον άμεσο τερματισμό του προγράμματος. Αν δοθεί ως όρισμα η τιμή 0, δηλώνεται επιτυχής τερματισμός, ενώ οποιαδήποτε μη μηδενική τιμή υποδηλώνει κατάσταση σφάλματος. Με τον τρόπο αυτό το πρόγραμμα μπορεί να επικοινωνεί την κατάσταση εκτέλεσής του στο περιβάλλον του λειτουργικού συστήματος.

1.14. Ασκήσεις

1. Γράψτε κώδικα που να εκτυπώνει τους ακέραιους από το 1 μέχρι και το 100. Όμως, αν ο αριθμός είναι πολλαπλάσιο του 3 τότε να εκτυπώνει Fizz, αν είναι πολλαπλάσιο του 5 το Buzz, ενώ αν είναι πολλαπλάσιο του 3 και του 5 τότε να εκτυπωνει FizzBuzz.
2. Γράψτε κώδικα που να επιλέγει τυχαία αριθμούς με το random.uniform μέχρι το άθροισμά τους να γίνει μεγαλύτερο από 1. Επαναλάβατε το πείραμα 1.000.000 φορές. Κατά μέσο όρο πόσες φορές χρειάστηκε να επιλεγούν τιμές για να ξεπεράσει το άθροισμα την τιμή 1. Συγκρίνατε τον αριθμό αυτό με το math.e. Υπάρχει κάποιο όνομα για αυτό το πείραμα;
3. Το πλήθος των τρόπων που μπορούν να επιλεγούν k άτομα από n άτομα δίνεται από τον τύπο $\binom{n}{k} = \frac{n!}{k!(n-k)!}$, όπου $n! = 1 * 2 * 3 * \ldots * n$. Εμφανίστε το πληθος των διαφορετικών ομάδων που μπορούν να σχηματιστούν από 50 άτομα για μέγεθος ομάδα από 1 έως και 50. Δείτε σχετικά τις συναρτήσεις math.factorial() και math.comb() από το package math της Python.
4. Η απόσταση ανάμεσα σε 2 τοποθεσίες στην Γη δίνεται με ακρίβεια από τον τύπο του Haversine. Στη συζήτηση https://stackoverflow.com/questions/4913349/haversine-formula-in-python-bearing-and-distance-between-two-gps-points στο stackoverflow μπορείτε να βρείτε μια καλή υλοποίηση σε Python. Επίσης, υπάρχουν packages στην Python, όπως το https://pypi.org/project/haversine/ που μπορούν να εγκατασταθούν με το pip και να γίνει απευθείας υπολογισμός της απόστασης με συνάρτηση του πακέτου. Υπολογίστε την απόσταση ανάμεσα στην Αθήνα και στο Βερολίνο τόσο με την υλοποίηση της συνάρτησης από το stackoverflow όσο και με το εξωτερικό πακέτο haversine και επιβεβαιώστε ότι επιστρέφουν (περίπου) την ίδια τιμή σε χιλιόμετρα.
5. Γράψτε μια κλάση με όνομα Player που να αναπαριστά έναν παίκτη σε ένα βιντεοπαιχνίδι. Η κλάση να έχει ως χαρακτηριστικά όνομα (name) και ζωές (lives), καθώς και μία μέθοδο με όνομα die() που να μειώνει κατά μία τις ζωές του παίκτη κάθε φορά που καλείται. Δημιουργήστε ένα αντικείμενο της κλάσης Player, με όνομα "Luigi" και 3 ζωές. Καλέστε 2

φορές τη μέθοδο die() και εμφανίστε το αντικείμενο. Προσθέσετε κατάλληλες μεθόδους __init__ και __str__.

6. Γράψτε κώδικα που να συνδέεται στη βάση δεδομένων sqlite_master.db που θα κατεβάσετε τοπικά από το https://github.com/bradleygrant/sakila-sqlite3, χρησιμοποιώντας τη βιβλιοθήκη sqlite3. Στη συνέχεια να θέτει ερώτημα προς τη βάση που να εμφανίζει από τον πίνακα film όλους τους τίτλους ταινιών που ξεκινούν με το γράμμα W και περιέχουν τη λέξη Shark στην περιγραφή τους. Παρατήρηση: Το SQL ερώτημα που θα πρέπει να τεθεί είναι το ακόλουθο:

```
SELECT title, description
FROM film
WHERE title like 'W%' AND description LIKE '%Shark%';
```

1.15. Ερωτήσεις αυτοαξιολόγησης

1. Η κύρια υλοποίηση της Python ονομάζεται:
   α) Jython
   β) IronPython
   γ) PyPy
   δ) CPython

2. Η Python είναι γλώσσα με:
   α) Στατικούς τύπους
   β) Υποχρεωτική δήλωση τύπου
   γ) Δυναμικούς τύπους
   δ) Χωρίς τύπους

3. Ποια από τις παρακάτω ΔΕΝ είναι έγκυρο όνομα μεταβλητής;
   α) value_1
   β) _count
   γ) totalSum
   δ) 1value

4. Ποια από τις ακόλουθες λέξεις είναι δεσμευμένη στην Python:
   α) function
   β) def
   γ) endif
   δ) var

5. Ποιος τελεστής ελέγχει ταυτότητα αντικειμένων;
   α) ==
   β) !=
   γ) in
   δ) is

6. Τι επιστρέφει η συνάρτηση input();
   α) int
   β) float

γ) str
δ) bool

7. Στη δομή if/elif/else εκτελούνται:
α) Πάντα όλα τα μπλοκ
β) Τουλάχιστον δύο μπλοκ
γ) Το πολύ ένα μπλοκ
δ) Κανένα μπλοκ

8. Ο τριαδικός τελεστής if έχει μορφή:
α) if condition: x else y
β) condition ? x : y
γ) x else y if condition
δ) x if condition else y

9. Οι συμβολοσειρές στην Python είναι:
α) Μεταβλητές δομές
β) Μη διατεταγμένες
γ) Αμετάβλητες (immutable)
δ) Μόνο ASCII

10. Ποιο αποτέλεσμα δίνει το s[::-1];
α) Αντιγραφή της συμβολοσειράς
β) Τμήμα (slice) χωρίς το πρώτο χαρακτήρα
γ) Ταξινόμηση χαρακτήρων
δ) Αντίστροφη συμβολοσειρά

11. Ποια δομή δεδομένων αποθηκεύει ζεύγη κλειδιού–τιμής;
α) dict
β) list
γ) tuple
δ) set

12. Τι επιστρέφει η έκφραση 10 // 4;
α) 2.5
β) 2
γ) 3
δ) 4

13. Τι θα εμφανίσει;
```
s = "abcdef"
print(s[2:5])
```
α) cde
β) bcde
γ) def
δ) cd

14. Τι θα εμφανίσει;
```
d = {1: "a", 2: "b"}
print(d[2])
```
α) a
β) b

γ) 2
δ) Σφάλμα

15. Τι θα εμφανίσει;

```
s = {1, 2, 2, 3}
print(len(s))
```

α) 3
β) 4
γ) 2
δ) Σφάλμα

16. Ποιο είναι το αποτέλεσμα;

```
"hello".replace("l", "x", 1)
```

α) hexxo
β) hexlo
γ) hexxx
δ) Σφάλμα

17. Τι θα εμφανίσει;

```
print("a,b,c".split(","))
```

α) "a" "b" "c"
β) ['a', 'b', 'c']
γ) a b c
δ) ('a', 'b', 'c')

18. Τι επιστρέφει η συνάρτηση input();
α) int
β) float
γ) str
δ) bool

19. Τι θα εμφανίσει;

```
def f(x):
    x += 1
    return x

a = 41
print(f(a), a)
```

α) 42 41
β) 42 42
γ) 41 42
δ) 41 41

20. Τι θα εμφανίσει;

```
def f():
    pass

print(f())
```

α) 0
β) False
γ) Σφάλμα
δ) None

1.15.1. Απαντήσεις στις ερωτήσεις αυτοαξιολόγησης

1. δ) CPython

2. γ) Δυναμικούς τύπους

3. δ) 1value

4. β) def

5. δ) is

6. γ) str

7. γ) Το πολύ ένα μπλοκ

8. δ) x if condition else y

9. γ) Αμετάβλητες (immutable)

10. δ) Αντίστροφη συμβολοσειρά

11. α) dict

12. β) 2

13. α) cde

14. β) b

15. α) 3

16. β) hexlo

17. γ) a b c

18. γ) str

19. α) 42 41

20. δ) None

ΚΕΦΑΛΑΙΟ 2: Η ΒΙΒΛΙΟΘΗΚΗ NUMPY

Η παρούσα ενότητα στοχεύει στην παρουσίαση της βιβλιοθήκης NumPy της Python, με έμφαση στη δημιουργία εφαρμογών που επικεντρώνονται στη διαχείριση και επεξεργασία δεδομένων. Στο πλαίσιο αυτό, παρουσιάζεται η βιβλιοθήκη NumPy, τα θεμελιώδη χαρακτηριστικά της., ενώ υπάρχουν ενδεικτικές μικρές εφαρμογές που αναδεικνύουν τις δυνατότητες της βιβλιοθήκης NumPy σε θέματα διαχείρισης δεδομένων.

2.1. Εισαγωγή στη βιβλιοθήκη NumPy

Η βιβλιοθήκη NymPy αποτελεί βασική επιλογή για τον προγραμματισμό εφαρμογών ανάλυσης δεδομένων σήμερα με χρήση της γλώσσας Python για όλες τις αναγκαίες εργασίες όπως ανάκτηση, συλλογή, καθαρισμό και γενικά διαχείριση δεδομένων. Μέχρι τώρα εξετάσαμε τις βασικές βιβλιοθήκες και δομές δεδομένων της Python (λίστες, λεξικά, πλειάδες κ.α.), οι οποίες σε συνδυασμό με τις δομές ελέγχου ροής παρέχουν στον προγραμματιστή τη δυνατότητα ανάπτυξης αξιόπιστων και αποδοτικών προγραμμάτων. Οι δομές αυτές επαρκούν για την υλοποίηση αλγορίθμων και την επίλυση καθημερινών προβλημάτων ενός επιστήμονα δεδομένων.

Ωστόσο, ένα κρίσιμο ζήτημα που προκύπτει είναι ο συνδυασμός απλότητας και ταχύτητας. Σε προβλήματα μικρής κλίμακας, όπως για παράδειγμα σε πράξεις γραμμικής άλγεβρας με πίνακες μικρών διαστάσεων, ο τρόπος υλοποίησης δεν έχει σημαντική επίδραση, ακόμη και λιγότερο βέλτιστες προσεγγίσεις, που περιλαμβάνουν επαναληπτικούς βρόχους, αποδίδουν ικανοποιητικά. Όταν όμως τα δεδομένα ανέρχονται σε εκατομμύρια εγγραφές, όταν απαιτούνται πολύπλοκα ερωτήματα ή υπολογιστικά σενάρια, ο παράγοντας χρόνος εκτέλεσης αποκτά ιδιαίτερη βαρύτητα. Σε επιχειρησιακά περιβάλλοντα η καθυστέρηση ή η υπερκατανάλωση πόρων μπορεί να έχει σημαντικό κόστος.

Η ανάγκη για εργαλεία που συνδυάζουν υψηλή απόδοση μαζί με ευκολία χρήσης υπήρξε πάντα καθοριστική στην επιστημονική κοινότητα γενικά και στον προγραμματισμό ειδικά. Η διαχείριση μεγάλου όγκου δεδομένων με ταυτόχρονη ανάγκη για εκτέλεση σύνθετων υπολογισμών είναι μια ανάγκη πολλών δεκαετιών. Για αυτόν τον λόγο δημιουργήθηκαν εξειδικευμένες γλώσσες, όπως η Fortran, σχεδιασμένες αποκλειστικά για επιστημονικούς υπολογισμούς. Η κοινότητα της Python, αναγνωρίζοντας αυτήν την απαίτηση, ανέπτυξε πακέτα που διευκολύνουν την ανάλυση δεδομένων για την επιστημονική έρευνα.

2.2. Η μετάβαση από τις βασικές δομές της Python στη βιβλιοθήκη NumPy

Ένα από τα ισχυρότερα εργαλεία σήμερα είναι η βιβλιοθήκη **NumPy** (**Numerical Python**) της γλώσσας προγραμματισμού Python, η οποία αποτελεί θεμελιώδες εργαλείο για την επιστήμη ανάλυσης δεδομένων και τη μηχανική μάθηση. Η βιβλιοθήκη NumPy προσφέρει:

- **Υψηλή απόδοση**: οι λειτουργίες σε πίνακες μπορούν να εκτελεστούν 10 έως 20 φορές ταχύτερα σε σύγκριση με κώδικα που βασίζεται σε απλούς βρόχους.
- **Απλότητα**: επιτρέπει στον προγραμματιστή να περιγράφει πολύπλοκους αλγορίθμους με σύντομο και καθαρό κώδικα.
- **Δομή δεδομένων ndarray**: έναν πολυδιάστατο πίνακα με προηγμένες δυνατότητες αποθήκευσης και χειρισμού αριθμητικών δεδομένων.
- **Εργαλεία εισόδου/εξόδου**: για ανάγνωση και εγγραφή δεδομένων απευθείας σε πίνακες.
- **Διασύνδεση με C, C++ και Fortran**: που επιτρέπει την εκτέλεση υπολογιστικά απαιτητικών εργασιών χωρίς περιττές αντιγραφές δεδομένων.

Ένας από τους βασικούς ρόλους της βιβλιοθήκης NumPy είναι να λειτουργεί ως κοινός πυρήνας αποθήκευσης δεδομένων για διαφορετικούς αλγορίθμους και βιβλιοθήκες. Αυτό καθιστά τους πίνακές της πολύ πιο αποδοτικούς και αποτελεσματικούς από τις ενσωματωμένες δομές της γλώσσας προγραμματισμού Python, ενώ παράλληλα επιτρέπει τη συνεργασία με βιβλιοθήκες χαμηλότερου επιπέδου. Με λίγα λόγια, η βιβλιοθήκη NumPy γεφυρώνει το χάσμα ανάμεσα στην

απλότητα της Python και την υπολογιστική ισχύ που απαιτούν τα σύγχρονα προβλήματα ανάλυσης δεδομένων.

Η βιβλιοθήκη NumPy βρίσκει εφαρμογές σε διάφορους τομείς, συμπεριλαμβανομένων ενδεικτικά:

- **Ανάλυση Δεδομένων**: παρέχει αποτελεσματικές δομές δεδομένων και συναρτήσεις για αριθμητική ανάλυση, καθιστώντας το μια εξαιρετική επιλογή για τον χειρισμό και τον χειρισμό μεγάλων συνόλων δεδομένων.
- **Μηχανική Μάθηση**: χρησιμοποιείται εκτενώς σε εργασίες μηχανικής μάθησης, όπως γραμμικές αλγεβρικές πράξεις, υπολογισμούς πινάκων και στατιστική ανάλυση δεδομένων.
- **Επεξεργασία Εικόνας**: προσφέρει μια ισχυρή βάση για εργασίες επεξεργασίας εικόνας, επιτρέποντας λειτουργίες όπως φιλτράρισμα εικόνας, μετασχηματισμούς και χειρισμούς σε επίπεδο pixel.
- **Επιστημονική Υπολογιστική**: χρησιμεύει ως βασικό εργαλείο της επιστημονική κοινότητας παρέχοντας εργαλεία στους ερευνητές και επιστήμονες των θετικών επιστημών και όχι μόνο, με προηγμένες μαθηματικές δυνατότητες.
- **Χρηματοοικονομική Μοντελοποίηση**: διευκολύνει την οικονομική ανάλυση παρέχοντας εργαλεία για μαθηματική μοντελοποίηση, αξιολόγηση κινδύνου και βελτιστοποίηση χαρτοφυλακίου.

Η βιβλιοθήκη NumPy έχει ένα κρίσιμο ρόλο στην ανάλυση δεδομένων λόγω της ικανότητάς της να χειρίζεται αποτελεσματικά μεγάλα σύνολα δεδομένων και να εκτελεί σύνθετους υπολογισμούς. Μπορεί να χρησιμοποιηθεί πολυποίκιλα σε έργα ανάλυσης δεδομένων όπως:

- **Χειρισμός Δεδομένων**: παρέχει πολυδιάστατους πίνακες του NumPy επιτρέπουν τον εύκολο χειρισμό, την τεμαχισμό και την αναδιαμόρφωση των δεδομένων, διευκολύνοντας την εξαγωγή πολύτιμων πληροφοριών από σύνθετα σύνολα δεδομένων.
- **Στατιστική Ανάλυση**: προσφέρει ένα ευρύ φάσμα στατιστικών συναρτήσεων, όπως μέσος όρος, διάμεσος, τυπική απόκλιση, συσχέτιση και άλλα, επιτρέποντάς σας να αναλύονται και να συνοψίζονται τα δεδομένα αποτελεσματικά.
- **Μαθηματικές Λειτουργίες**: παρέχει μια πλούσια συλλογή μαθηματικών συναρτήσεων, επιτρέποντάς να εκτελούνται μαθηματικές πράξεις σε πίνακες αποτελεσματικά. Αυτές οι λειτουργίες μπορούν να εφαρμοστούν ανά στοιχείο ή σε ολόκληρους πίνακες.

- **Εκπομπή Πίνακα**: παρέχει λειτουργία μετάδοσης (broadcasting) η οποία επιτρέπει έμμεσες λειτουργίες ανά στοιχείο μεταξύ πινάκων διαφορετικών σχημάτων και μεγεθών, εξαλείφοντας την ανάγκη για χρήση βρόγχων με υψηλό υπολογιστικό κόστος, ενισχύοντας έτσι την υπολογιστική αποδοτικότητα.
- **Αριθμητικές Προσομοιώσεις**: χρησιμοποιείται στη δημιουργία προσομοιώσεων και στην ανάλυση σύνθετων συνόλων δεδομένων για τον έλεγχο υποθέσεων, την επικύρωση μοντέλων και τη διεξαγωγή πειραμάτων.

Για την χρήση της βιβλιοθήκης NumPy σε τοπικό υπολογιστή, θα πρέπει πρώτα να εγκατασταθεί με την εντολή pip σε κάποιο terminal (τερματικό) (Κ. 2.2.1 - Εγκατάσταση βιβλιοθήκης numpy

) ή άλλων τρόπων που μπορεί να παρέχει ανάλογα κάθε IDE (Integrated Development Enviroment) πρόγραμμα

```
pip install numpy
```
Κ. 2.2.1 - Εγκατάσταση βιβλιοθήκης numpy

Μόλις εγκατασταθεί, μπορείτε να εισαγάγετε το NumPy στο Python script σας (Κ. 2.2.2):

```
import numpy as np
```
Κ. 2.2.2 - Δήλωση εισαγωγής της βιβλιοθήκης Numpy στον κώδικα

Σύμφωνα με μια άτυπη σύμβαση, είναι **σύνηθες** όταν εισάγουμε τη βιβλιοθήκη NumPy στον κώδικα μας να δίνουμε το ψευδώνυμο **np** για να κάνουμε τον κώδικα πιο ευανάγνωστο και επαναχρησιμοποιήσιμο ευκολότερα.

Εάν η βιβλιοθήκη Numpy είναι ήδη εγκατεστημένη, τότε μπορεί να λάβετε μήνυμα στην έξοδο του terminal όπως στο Ε. 2.2.1 - Μήνυμα στο Terminal ότι είναι ήδη εγκατεστημένη κάποια έκδοση

όπου δείχνει ότι είναι ήδη εγκατεστημένη η έκδοση 2.3.5 της βιβλιοθήκης Numpy.

```
Requirement already satisfied: numpy in
c:\users\ezoul\appdata\local\programs\python\python313\lib\site-packages
(2.3.5)
```
Ε. 2.2.1 - Μήνυμα στο Terminal ότι είναι ήδη εγκατεστημένη κάποια έκδοση

Στην περίπτωση που υπάρχει και νεότερη διαθέσιμη έκδοση θα ακολουθείται και από μήνυμα παρόμοιο με της Ε. 2.2.2 - Διαθεσιμότητα νεότερης έκδοσης της βιβλιοθήκης NumpyΕ. 2.2.2

```
[notice] A new release of pip is available: 25.2 -> 26.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```
Ε. 2.2.2 - Διαθεσιμότητα νεότερης έκδοσης της βιβλιοθήκης Numpy

2.3. Τύποι πινάκων και χαρακτηριστικά πινάκων (shape, size)

Στην ενότητα αυτή θα δούμε τους τύπους των πινάκων της βιβλιοθήκης Numpy και τα χαρακτηριστικά τους. Στις περισσότερες των περιπτώσεων οι έννοιες και οι δυνατότητες είναι δανεισμένες από τις αντίστοιχες της θεωρίας των μαθηματικών για τους πίνακες. Υπάρχουν όμως και εξαιρέσεις που θα αναφέρουμε συγκεκριμένα. Ένας πίνακας μονοδιάστατος (1D) για την βιβλιοθήκη Numpy της Python είναι ουσιαστικά μια λίστα τιμών. Για να μπορεί κάποιος να χρησιμοποιήσει πίνακα 1D θα πρέπει να εισάγει στον κώδικα τη βιβλιοθήκη Numpy όπως μάθαμε παραπάνω:

```
import numpy as np #Εισαγωγή βιβλιοθήκης Numpy
numbers = np.array([34, 33, 29, 28, 27]) # Δημιουργία πίνακα Numpy με όνομα numbers
print(numbers) #Έξοδος: array([34, 33, 29, 28, 27])
```

Κ. 2.3.1 - Δημιουργία και εκτύπωση μονοδιάστατου πίνακα Numpy

2.3.1. Δισδιάστατοι Πίνακες

Η μαθηματική έννοια του δισδιάστατου πίνακα είναι ένα πλέγμα τιμών διατεταγμένων σε γραμμές και στήλες. Με τη βιβλιοθήκη Numpy μπορούμε να δημιουργήσουμε έναν δισδιάστατο πίνακα, θέτοντας ως όρισμα της συνάρτησης np.array() μια λίστα λιστών, όπως φαίνεται στα παραδείγματα Κ. 2.3.2 και Κ. 2.3.3.

```
import numpy as np
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(matrix) #Έξοδος: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Κ. 2.3.2 - Δημιουργία και εκτύπωση δισδιάστατου πίνακα Numpy

```
import numpy as np
measurements = np.array([[185, 75], [170, 68], [192, 82], [178, 70], [175, 77]])
print(matrix) #Έξοδος: [[185, 75], [170, 68], [192, 82], [178, 70], [175, 77]]
```

Κ. 2.3.3 - Δημιουργία και εκτύπωση δισδιάστατου πίνακα Numpy, οι γραμμές είναι διαφορετικοί άνθρωποι και οι στήλες αναπαριστούν τα ύψη και τα βάρη τους

2.3.2. Τρισδιάστατοι και περισσότερων διαστάσεων Πίνακες (Tensors)

Η μαθηματική έννοια του τρισδιάστατου πίνακα είναι μια συλλογή από δισδιάστατους πίνακες διατεταγμένους σε επίπεδα. Έτσι με τη βιβλιοθήκη Numpy, πάλι με τη χρήση της συνάρτησης np.array() ένας τρισδιάστατος πίνακας, γνωστός και ως tensor στην Python, είναι μια συλλογή από δισδιάστατους πίνακες διατεταγμένους σε επίπεδα. Η δημιουργία ενός τρισδιάστατου πίνακα, γίνεται θέτοντας ως όρισμα μια λίστα λιστών από λίστες στη συνάρτηση np.array() , όπως φαίνεται στα παραδείγματα Κ. 2.3.4 και Κ. 2.3.5

```
import numpy as np
tensor = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print(tensor) #Έξοδος: [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
```

Κ. 2.3.4 - Δημιουργία και εκτύπωση τρισδιάστατου πίνακα Numpy

```
import numpy as np
group_measurements = np.array([[[185, 75], [170, 68], [192, 82], [178, 70],
[175, 77]],
[[180, 72], [176, 78], [182, 80], [174, 72], [177, 69]],[[188, 78], [180,
71], [182, 75], [17S, 73], [17c, 70]]])
print(group_measurements)
#Έξοδος: [[[185, 75], [170, 68], [192, 82], #[178,70], [175, 77]],[[180,
72], [176, 78], [182, 80], [174, 72], [177, #69]], [[188, 78], [180, 71],
[182, 75], [17S, 73], [17c, 70]]]
```

Κ. 2.3.5 - Δημιουργία και εκτύπωση τρισδιάστατου πίνακα Numpy. Κάθε επίπεδο αναπαριστά διαφορετική ομάδα, οι γραμμές τα άτομα, η μια διάσταση το ύψος

2.3.3. Ειδικοί Πίνακες

Η βιβλιοθήκη NumPy παρέχει συναρτήσεις για τη δημιουργία ειδικών πινάκων, ώστε να καλύψει κατηγορίες ειδικών πινάκων οι οποίοι υπάρχουν στη θεωρία πινάκων των μαθηματικών αλλά και για να καλύψει άλλες προγραμματιστικές ανάγκες. Έτσι υπάρχουν στη βιβλιοθήκη Numpy υπάρχουν οι παρακάτω συναρτήσεις που δημιουργούν ειδικούς πίνακες:

• np.zeros(): Δημιουργεί έναν πίνακα γεμάτο με μηδενικά. Χρήσιμο για την αρχικοποίηση ενός πίνακα πριν από την πλήρωσή του με δεδομένα (Κ. 2.3.6).

```
import numpy as np
zeros = np.zeros((3, 3))
print("Zeros:\n", zeros) #Zeros: [[0. 0. 0.], [0. 0. 0.], [0. 0. 0.]]
```

Κ. 2.3.6 - Δημιουργία και εκτύπωση μηδενικού πίνακα 3x3

- np.ones(): Δημιουργεί έναν πίνακα γεμάτο με μονάδες. Χρήσιμο για τη δημιουργία ενός πίνακα με μια συγκεκριμένη αρχική τιμή (Κ. 2.3.7).

```
import numpy as np
ones = np.ones((3, 3))
print("Πίνακας μονάδων:", ones) #Πίνακας μονάδων: [1. 1. 1.], [1. 1. 1.],
[1. 1. 1.]]
```

Κ. 2.3.7 - Δημιουργία και εκτύπωση πίνακα μονάδων 3x3

- np.eye(): Δημιουργεί έναν μοναδιαίο ή ταυτοτικό πίνακα. Χρήσιμο για πράξεις γραμμικής άλγεβρας (Κ. 2.3.8).

```
import numpy as np
identity = np.eye(3)
print("Μοναδιαίος Πίνακας:", identity) #Μοναδιαίος Πίνακας: [[1. 0. 0.],
[0. 1. 0.], [0. 0. 1.]]
```

Κ. 2.3.8 - Δημιουργία και εκτύπωση Μοναδιαίου πίνακα 3x3

- np.full(): Δημιουργεί έναν πίνακα γεμάτο με μια καθορισμένη τιμή. Χρήσιμο για τη δημιουργία ενός πίνακα με μια συγκεκριμένη αρχική τιμή (Κ. 2.3.9).

```
import numpy as np
full = np.full((3, 3), 7)
print("Πλήρης πίνακας όμοιων αριθμών:", full) #Πλήρης πίνακας όμοιων
αριθμών: [[7 7 7], [7 7 7], [7 7 7]]
```

Κ. 2.3.9 - Δημιουργία και εκτύπωση Πλήρους Πίνακα ομοίων αριθμών, διαστάσεων 3x3

- np.arange(): Δημιουργεί έναν πίνακα με τιμές σε κανονική απόσταση μεταξύ μιας αρχικής και μιας τελικής τιμής. Η συνάρτηση είναι χρήσιμη για τη δημιουργία ακολουθιών αριθμών (Κ. 2.3.10).

```
import numpy as np
arange = np.arange(0, 10, 2)
print("Ακολουθία:", arange) #Ακολουθία: [0 2 4 6 8]
```

Κ. 2.3.10 - Δημιουργία και εκτύπωση πίνακα Ακολουθίας αριθμών

- np.linspace(): Δημιουργεί έναν πίνακα με έναν καθορισμένο αριθμό τιμών σε ίση απόσταση μεταξύ μιας αρχικής και μιας τελικής τιμής. Χρήσιμο για τη δημιουργία ισαπεχόντων διαστημάτων (Κ. 2.3.11).

```
import numpy as np
linspace = np.linspace(0, 1, 5)
print("Ισαπέχοντα διαστήματα:", linspace) #Ισαπέχοντα διαστήματα: [0. 0.25
0.5 0.75 1.]
```

Κ. 2.3.11- Δημιουργία και εκτύπωση πίνακα τιμών σε ισαπέχοντα διαστήματα

2.3.4. Βασικά Χαρακτηριστικά Πινάκων

Κάθε πίνακας της βιβλιοθήκης Numpy τύπου ndarray διαθέτει χρήσιμες ιδιότητες όπως αυτές αναφέρονται παρακάτω. Στον Κ. 2.3.12 εμφανίζεται ενδεικτικό παραδείγματα χρήσης και εκτύπωσης των επιστρεφόμενων τιμών βασικών χαρακτηριστικών ενδεικτικού πίνακα 2x3 με τιμές [[1, 2, 3], [4, 5, 6]].

- ndim → αριθμός διαστάσεων (π.χ. 1D, 2D, 3D).
- shape → το σχήμα του πίνακα (πλήθος στοιχείων σε κάθε διάσταση).
- size → το συνολικό πλήθος στοιχείων.
- dtype → ο τύπος δεδομένων των στοιχείων (π.χ. int32, float64).
- itemsize → το μέγεθος κάθε στοιχείου σε bytes.
- nbytes → συνολικό μέγεθος πίνακα σε bytes.

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print("Πίνακας:", arr)       #Πίνακας: [[1, 2, 3], [4, 5, 6]]
print("Διαστάσεις:", arr.ndim)       #Διαστάσεις: 2
print("Σχήμα:", arr.shape)           #Σχήμα: (2,3)
print("Συνολικά στοιχεία:", arr.size)  #Συνολικά στοιχεία: 6
print("Τύπος δεδομένων:", arr.dtype)   #Τύπος δεδομένων: int64 (ή ανάλογα
το σύστημα)
print("Bytes ανά στοιχείο:", arr.itemsize) #Bytes ανά στοιχείο: 8
print("Συνολικά bytes:", arr.nbytes) #Συνολικά bytes: 48
```

Κ. 2.3.12 - Δημιουργία πίνακα, υπολογισμός και εκτύπωση βασικών χαρακτηριστικών

Μια πολύ χρήσιμη συνάρτηση που λειτουργεί ως ιδιότητα των πινάκων είναι η μετατροπή πίνακα σε απλή λίστα τιμών (flattened list). Λαμβάνει ένα numpy.ndarray και το μετατρέπει σε λίστα τιμών Κ. 2.3.13.

```python
import numpy as np
# Δημιουργία πίνακα NumPy σε λίστα τιμών
A = np.array([[1, 2, 3], [4, 2, 1], [3, 1, 4]])
#Μετατροπή σε λίστα τιμών (flat)
values = A.flatten().tolist()
print("Λίστα τιμών:", values) #Λίστα τιμών: [1, 2, 3, 4, 2, 1, 3, 1, 4]
```

Κ. 2.3.13 - Μετατροπή πίνακα σε λίστα τιμών

Σε σύνδεση με την προηγούμενη ιδιότητα και επεκτείνοντας σε κάτι εξαιρετικά χρήσιμο στον προγραμματισμό η βιβλιοθήκη Numpy έχει την ιδιότητα ndenumerate με την οποία μπορούμε να λάβουμε σε ποιες θέσεις/δείκτες εμφανίζεται η κάθε τιμή. Έτσι μπορούμε να βρούμε τη θέση της κάθε τιμής (Κ. 2.3.14).

```python
import numpy as np
# Δημιουργία πίνακα NumPy σε λίστα τιμών
A = np.array([[1, 2, 3], [4, 2, 1], [3, 1, 4]])
#Μετατροπή σε λίστα τιμών (flat)
values = A.flatten().tolist()
print("Λίστα τιμών:", values) #Λίστα τιμών: [1, 2, 3, 4, 2, 1, 3, 1, 4]
# 2. Εύρεση θέσεων κάθε τιμής
positions = {}
for index, value in np.ndenumerate(A):
    if value not in positions:
        positions[value] = []
    positions[value].append(index)
print("\nΘέσεις κάθε τιμής:")
for val, inds in positions.items():
    print(f"{val}: {inds}")
```

Κ. 2.3.14 - Χρήση της ιδιότητας ndenumerate

Η έξοδος του κώδικα της Κ. 2.3.14 είναι η έξοδος Ε. 2.3.1.

```
Λίστα τιμών: [1, 2, 3, 4, 2, 1, 3, 1, 4]

Θέσεις κάθε τιμής:
1: [(0, 0), (1, 2), (2, 1)]
2: [(0, 1), (1, 1)]
3: [(0, 2), (2, 0)]
4: [(1, 0), (2, 2)]
```

Ε. 2.3.1 - Θέσεις τιμών πίνακα

2.4. Δυνατότητες Διαχείρισης πινάκων με τη Βιβλιοθήκη Numpy

2.4.1. Ευρετηρίαση (Indexing)

Η βιβλιοθήκη NumPy είναι πανίσχυρο εργαλείο και ανάμεσα στις σημαντικότερες εκ των βιβλιοθηκών της γλώσσας προγραμματισμού Python, παρέχοντας ένα ευρύ φάσμα δυνατοτήτων για αριθμητικούς υπολογισμούς. Σε αυτή την ενότητα θα δούμε τις ισχυρές τεχνικές ευρετηρίασης και τεμαχισμού που παρέχει η βιβλιοθήκη. Για να δούμε αυτές τις έννοιες θα δημιουργήσουμε απτά, καθημερινά παραδείγματα. Οι ρίζες της δημιουργίας ευρετηρίασης στη βιβλιοθήκη NumPy μπορούν να εντοπιστούν στις «ταπεινές» αλλά βασικές αρχές των λιστών της γλώσσας Python. Η πρόσβαση σε

μεμονωμένα στοιχεία ενός πίνακα είναι τόσο απλή όσο η κλήση του ευρετηρίου τους με τη σύνταξη τύπου `array[index].` Ως παράδειγμα έχουν την περίπτωση πίνακα NumPy ο οποίος περιέχει πολλαπλά ροφήματα. Ο κώδικας για τη δημιουργία και εκτύπωση όλου αυτού του πίνακα είναι στο Κ. 2.4.1.

```
import numpy as np
beverages = np.array(['Coffee', 'Tea', 'Milk', 'Juice', 'Water'])
print (beverages) # Αυτή η γραμμή είναι προαιρετική και δίνει το αποτέλεσμα εκτύπωσης του πίνακα: ['Coffee' 'Tea' 'Milk' 'Juice' 'Water']
```

Κ. 2.4.1 - Δημιουργία και εκτύπωση πίνακα με τιμές ροφήματα

Για να ανακληθεί το πρώτο στοιχείο του πίνακα, πρέπει να γίνει χρήση του δείκτη 0 και να εκτυπώσουμε την επιστρεφόμενη τιμή. Έτσι συνολικά ο κώδικας θα είναι όπως εμφανίζεται στο

```
import numpy as np
beverages = np.array(['Coffee', 'Tea', 'Milk', 'Juice', 'Water'])
print (beverages) # Αυτή η γραμμή είναι προαιρετική και δίνει το αποτέλεσμα εκτύπωσης του πίνακα: ['Coffee' 'Tea' 'Milk' 'Juice' 'Water']
first_beverage = beverages[0]
print(first_beverage) # Coffee
```

Κ. 2.4.2 - Δημιουργία πίνακα ροφημάτων, χρήση ευρετηρίασης για εύρεση πρώτου στοιχείου

Στη βιβλιοθήκη NumPy, οι πίνακες μπορούν να εκτείνονται σε πολλαπλές διαστάσεις. Για να επιλεγούν στοιχεία από αυτούς τους πολυδιάστατους πίνακες, θα πρέπει να χρησιμοποιήσουμε μια λίστα δεικτών διαχωρισμένων με κόμμα. Η βασική σύνταξη είναι `array[index1, index2, ...].` Για παράδειγμα για έναν δισδιάστατο πίνακα που αντιπροσωπεύει ένα μικρό μέρος της διάταξης των ραφιών ενός παντοπωλείου (Κ. 2.4.3), για να επιλεγεί το «Milk» από το δεύτερο ράφι, θα πρέπει να χρησιμοποιηθούν οι δείκτες 1 και 2, όπως φαίνεται στη γραμμή 6 του κώδικα Κ. 2.4.3

```
import numpy as np
shelf_layout = np.array([['Cereals', 'Pasta', 'Rice', 'Beans'],
                         ['Coffee', 'Tea', 'Milk', 'Juice'],
                         ['Apple', 'Banana', 'Mango', 'Peach']])
print(shelf_layout)
milk_position = shelf_layout[1, 2]
print(milk_position) # Milk
```

Κ. 2.4.3 - Ευρετηρίασης σε πολλές διαστάσεις

Στη βιβλιοθήκη NumPy υπάρχουν και εξελιγμένες τεχνικές ευρετηρίασης όπως η λογική και η ακέραιη ευρετηρίαση. Η λογική ευρετηρίαση επιτρέπει να επιλεγούν στοιχεία από έναν πίνακα με βάση μια συνθήκη. Επεκτείνοντας το παράδειγμα του Κ. 2.4.2 με τα ροφήματα προσθέτουμε και πίνακα με τις τιμές τους (πίνακας prices στον Κ. 2.4.4). Για να εντοπιστούν τα ροφήματα με τιμές κάτω του 1€ μπορεί να χρησιμοποιηθεί η δήλωση όπως αυτή μετά το σύμβολο = στην έκτη γραμμή του κώδικα Κ. 2.4.4.

```
import numpy as np
beverages = np.array(['Coffee', 'Tea', 'Milk', 'Juice', 'Water'])
#Αρχικοποίηση Τιμών
prices = np.array([1.5, 1.0, 0.5, 2.0, 0.7])
```

```
#Ροφήματα με αποδεκτές τιμές
cheap_beverages = beverages[prices < 1.0]
print(cheap_beverages) #['Milk' 'Water']
```
Κ. 2.4.4 - Παράδειγμα λογικής ευρετηρίασης

Άλλο ένα παράδειγμα χρήσης λογικής ευρετηρίασης με πολυδιάστατους πίνακες είναι της μορφής που παρουσιάζεται στη συνέχεια. Ας επεκτείνουμε το παράδειγμα διάταξης των ραφιών ενός παντοπωλείου (Κ. 2.4.3), ώστε να εντοπιστούν τα τρόφιμα στο ράφι του παντοπωλείου που δεν έχει κάποιος αλλεργία, στο συγκεκριμένο παράδειγμα έστω ότι εμφανίζεται αλλεργία στα 'Mango' και 'Peach' (Κ. 2.4.5). Για την επιλογή του 'Apple' και 'Banana' μπορεί να χρησιμοποιηθεί η δήλωση όπως αυτή μετά το σύμβολο = στην πέμπτη γραμμή του κώδικα Κ. 2.4.5. Για την επιλογή του 'Apple' και 'Banana' μπορεί να χρησιμοποιηθεί η δήλωση όπως αυτή μετά το σύμβολο = στην έβδομη γραμμή του κώδικα Κ. 2.4.5. Για την επιλογή του 'Apricot', που δεν υπάρχει, μπορεί να χρησιμοποιηθεί η δήλωση όπως αυτή μετά το σύμβολο = στην ένατη γραμμή του κώδικα Κ. 2.4.5. Για την επιλογή του 'Apple' μπορεί να χρησιμοποιηθεί η δήλωση όπως αυτή μετά το σύμβολο = στην ενδέκατη γραμμή του κώδικα Κ. 2.4.5.

```
import numpy as np
shelf_layout = np.array([['Cereals', 'Pasta', 'Rice', 'Beans'],
                         ['Coffee', 'Tea', 'Milk', 'Juice'],
                         ['Apple', 'Banana', 'Mango', 'Peach']])
fruit_positions1 = shelf_layout[np.isin(shelf_layout, ['Apple', 'Banana'])]
print(fruit_positions1) #Θέση φρούτων 1 - ['Apple' 'Banana']
fruit_positions2 = shelf_layout[np.isin(shelf_layout, ['Apple'])]
print(fruit_positions2) #Θέση φρούτων 2 - ['Apple']
fruit_positions3 = shelf_layout[np.isin(shelf_layout, ['Apricot'])]
print(fruit_positions3) #Θέση φρούτων 3 - []
fruit_positions4 = shelf_layout[np.isin(shelf_layout, ['Apple',
'Apricot'])]
print(fruit_positions4) #Θέση φρούτων 4 - ['Apple']
```
Κ. 2.4.5 - Δεικτιοδότηση με εύρεση συγκεκριμένων τιμών σε πολυδιάστατους πίνακες

2.4.2. Ευρετηρίαση (Indexing) – Προχωρημένες περιπτώσεις

Η χρήση ευρετηρίασης έχει και άλλες επιπλέον προχωρημένες επιλογές, όπως με τη χρήση ακέραιων αριθμών, όπου παρέχεται η επιλογή στοιχείων από έναν πίνακα χρησιμοποιώντας έναν άλλο πίνακα δεικτών. Επεκτείνοντας το παράδειγμα του Κ. 2.4.2 με τα ροφήματα προσθέτουμε δήλωση όπως αυτή μετά το σύμβολο = στην τρίτη γραμμή του κώδικα Κ. 2.4.6 ώστε να εντοπιστεί το πρώτο και το τελευταίο ρόφημα.

```
import numpy as np
beverages = np.array(['Coffee', 'Tea', 'Milk', 'Juice', 'Water'])
selected_beverages = beverages[[0, -1]]
print(selected_beverages) # ['Coffee' 'Water']
```
Κ. 2.4.6 - Ευρετηρίαση με τη χρήση ακέραιων αριθμών

Για πολυδιάστατους πίνακες, μπορεί να χρησιμοποιηθεί η ακέραιη ευρετηρίαση παρέχοντας πίνακες δεικτών για κάθε διάσταση. Αξιοποιώντας το παράδειγμα της διάταξης των ραφιών ενός

παντοπωλείου (Κ. 2.4.3) ώστε να βρούμε αυτή τη φορά το πρώτο τρόφιμο και το τελευταίο ρόφημα Κ. 2.4.7.

```
import numpy as np
shelf_layout = np.array([['Cereals', 'Pasta', 'Rice', 'Beans'],
                         ['Coffee', 'Tea', 'Milk', 'Juice'],
                         ['Apple', 'Banana', 'Mango', 'Peach']])
#Select Items
selected_items = [shelf_layout[0][0], shelf_layout[2][3]]
print(selected_items) # [np.str_('Cereals'), np.str_('Peach')]
```

Κ. 2.4.7- Ευρετηρίαση με τη χρήση ακέραιων αριθμών σε πολυδιάστατους πίνακες

2.4.3. Τεμαχισμός (Slicing)

Μια άλλη δυνατότητα της βιβλιοθήκης Numpy σε σχέση με τους πίνακες είναι ο τεμαχισμός, όπου ουσιαστικά είναι ο διακριτός ορισμός ενός πίνακα. Η άνω και κάτω τελεία (:) χρησιμοποιείται για να ορίσει την αρχή, το τέλος και το βήμα της τεμαχισμού (`array[start:end:step])`. Για παράδειγμα, αν πρέπει να εξαχθούν τα πρώτα 3 ροφήματα από τη σειρά ροφημάτων μπορεί να χρησιμοποιηθεί η δήλωση όπως αυτή μετά το σύμβολο = στην τρίτη γραμμή του κώδικα Κ. 2.4.8

```
import numpy as np
beverages = np.array(['Coffee', 'Tea', 'Milk', 'Juice', 'Water'])
top_3_beverages = beverages[:3] # Εξαγωγή των τριών πρώτων ροφημάτων από το ράφι
print(top_3_beverages) # Έξοδος:['Coffee' 'Tea' 'Milk']
```

Κ. 2.4.8 - Τεμαχισμός ραφιού προϊόντων

Ακόμη και οι πολυδιάστατοι πίνακες μπορούν να τεμαχιστούν καθορίζοντας την κάθε διάσταση, διαχωρισμένη με κόμματα σε μια σύνταξη της μορφής `array[start1:end1:step1, start2:end2:step2, ...]`. Στον κώδικα Κ. 2.4.9.παρουσιάζεται ένας πολυδιάστατος πίνακας με την πρώτη γραμμή (ράφι παντοπωλίου) να περιλαμβάνει τρόφιμα, τη δεύτερη ροφήματα και την τρίτη φρούτα. Για την επιλογή του 2^{ου} ραφιού μπορεί να χρησιμοποιηθεί η δήλωση όπως αυτή μετά το σύμβολο = στην πέμπτη γραμμή του κώδικα Κ. 2.4.9.

```
import numpy as np
shelf_layout = np.array([['Cereals', 'Pasta', 'Rice', 'Beans'],
                         ['Coffee', 'Tea', 'Milk', 'Juice'],
                         ['Apple', 'Banana', 'Mango', 'Peach']])
second_shelf = shelf_layout[1, :] #['Coffee' 'Tea' 'Milk' 'Juice']
print(second_shelf)
```

Κ. 2.4.9 - Τεμαχισμός ραφιού προϊόντων πολλών διαστάσεων

2.4.4. Αλλαγή διαστάσεων (reshaping)

Ένας πίνακας NumPy μπορεί να αναδιαμορφωθεί χρησιμοποιώντας τη συνάρτηση reshape(). Είναι σημαντικό να αναφερθεί ότι το γινόμενο των γραμμών και των στηλών στον αναδιαμορφωμένο πίνακα πρέπει να είναι ίσο με το γινόμενο των γραμμών και των στηλών στον αρχικό πίνακα. Για παράδειγμα, σε έναν αρχικό πίνακα που περιέχει τέσσερις γραμμές και έξι στήλες, δηλαδή, 4 x 6 =

24. Ένας αναδιαμορφωμένος πίνακας περιέχει τρεις γραμμές και οκτώ στήλες, δηλαδή, 3 x 8 = 24 (Κ. 2.4.10 - Αλλαγή διαστάσεων (reshaping)).

```
import numpy as np
uniform_random = np.random.rand(4, 6) #Δημιουργία πίνακα 4x6 με τυχαίους αριθμούς
print (uniform_random)
uniform_random = uniform_random.reshape(3, 8) #Εφαρμογή reshape στον πίνακα 4x6 σε πίνακα 3x8
print (uniform_random)
```

Κ. 2.4.10 - Αλλαγή διαστάσεων (reshaping)

Ένα ακόμα παράδειγμα εφαρμογής της reshape() εμφανίζεται στον κώδικα Κ. 2.4.11.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6])
reshaped_arr = np.reshape(arr, (2, 3))
print("\nReshaped array:\n", reshaped_arr) #[[1 2 3] [4 5 6]]
```

Κ. 2.4.11 - Επιπλέον παράδειγμα για την αλλαγή διαστάσεων (reshaping)

2.4.5. Συνένωση πινάκων

Σε αρκετές περιπτώσεις ανάλυσης δεδομένων μπορεί να διαθέτουμε τα δεδομένα μας σε δύο πίνακες και να θέλουμε να τα συνενώσουμε σε έναν, είτε κατά γραμμές είτε κατά στήλες. Στη συνένωση σε γραμμές μπορεί να χρησιμοποιηθεί `np.concatenate((a, b), axis=0)` ή ενναλακτικά `np.vstack((a, b))` (Κ. 2.4.12- Συνένωση πινάκων - κάθετηΚ. 2.4.12). Όποια επιλογή και να χρησιμοποιηθεί το αποτέλεσμα είναι ίδιο. Στη συνένωση σε στήλες μπορεί να χρησιμοποιηθεί `np.concatenate((a, b), axis=1)` ή ενναλακτικά `np.hstack((a, b))`, πάλι όποια επιλογή και να χρησιμοποιηθεί το αποτέλεσμα είναι ίδιο (Κ. 2.4.13).

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
# Συνένωση κατά τις γραμμές (axis=0)
ch1 = np.concatenate((a, b), axis=0)
# Εναλλακτικά np.vstack((a, b)) → κάθετη ένωση (ισοδύναμο με axis=0)
ch2 = np.vstack((a, b))
print(ch1)
print(ch2)
 #Ίδιο αποτέλεσμα για τα ch1, ch2
 #[[1 2]
      [3 4]
      [5 6]
      [7 8]]
```

Κ. 2.4.12- Συνένωση πινάκων - κάθετη

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
# Συνένωση κατά τις γραμμές (axis=1)
ch1 = np.concatenate((a, b), axis=1)
# Εναλλακτικά np.hstack((a, b)) → οριζόντια ένωση (ισοδύναμο με axis=1)
ch2 = np.hstack((a, b))
print(ch1)
print(ch2)
```

```
#Ίδιο αποτέλεσμα ch1, ch2
#[[1 2 5 6]
   [3 4 7 8]]
```
Κ. 2.4.13 - Συνένωση πινάκων - οριζόντια

2.4.6. Διαχωρισμός πινάκων

Υπάρχουν πολλοί τρόποι διαχωρισμού πινάκων με χρήση των συναρτήσεων της βιβλιοθήκης Numpy. Έτσι έχουμε διαχωρισμό σε ίσα μέρη, με χρήση της `np.split` (Κ. 2.4.14), πιο ευέλικτο, διαχωρισμό γιατί δέχεται και μη ίσο διαχωρισμό.διαχωρισμό με χρήση της `np.array_split` (Κ. 2.4.15), σε δισδιάστατους πίνακες κάθετος διαχωρισμός (κατά στήλες - Κ. 2.4.16) με χρήση της `np.vsplit`, σε δισδιάστατους πίνακες οριζόντιος διαχωρισμός (κατά γραμμές - Κ. 2.4.17) με χρήση της `np.hsplit`, και με χρήση δεικτών (π.χ. `np.split(a, [2, 4])` - Κ. 2.4.18).

```
import numpy as np
a = np.array([1, 2, 3, 4, 5, 6])
# Χωρίζουμε σε 3 ίσα μέρη
parts = np.split(a, 3)
print(parts)
#[array([1, 2]), array([3, 4]), array([5, 6])]
```
Κ. 2.4.14 - Διαχωρισμός πινάκων – σε ίσα μέρη

```
import numpy as np
a = np.array([1, 2, 3, 4, 5])
# Χωρίζουμε σε 3 μέρη (δεν είναι ίσα, αλλά το κάνει)
parts = np.array_split(a, 3)
print(parts)
#[array([1, 2]), array([3, 4]), array([5])]
```
Κ. 2.4.15 - Διαχωρισμός πινάκων – σε μη ίσα μέρη

```
import numpy as np
b = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8]])
# Οριζόντιος διαχωρισμός (κατά γραμμές)
top, bottom = np.vsplit(b, 2)
print(top)
#[[1 2 3 4]]
print(bottom)
#[[5 6 7 8]]
```
Κ. 2.4.16 - Διαχωρισμός πινάκων – Κάθετος διαχωρισμός

```
import numpy as np
b = np.array([[1, 2, 3, 4],
              [5, 6, 7, 8]])
# Κάθετος διαχωρισμός (κατά στήλες)
left, right = np.hsplit(b, 2)
print(left)
#[[1 2]
#[5 6]]
print(right)
#[[3 4]
#[7 8]]
```
Κ. 2.4.17 - Διαχωρισμός πινάκων - Οριζόντιος διαχωρισμός

```
import numpy as np
a = np.array([10, 20, 30, 40, 50, 60])
# Διαχωρισμός στους δείκτες 2 και 4
```

```
parts = np.split(a, [2, 4])
print(parts)
#[array([10, 20]), array([30, 40]), array([50, 60])]
```
Κ. 2.4.18 - Διαχωρισμός πινάκων - Με δείκτες (indices)

2.4.7. Στοίβαξη (Stacking) πινάκων

Η στοίβαξη (stacking) στη βιβλιοθήκη NumPy χρησιμοποιείται ώστε να «στοιβάζουμε» πίνακες μαζί (σαν στρώσεις πινάκων). Είναι διαφορετικό από τη συνένωση (concatenate), γιατί με τη στοίβαξη προσθέτουμε νέο άξονα. Έχουμε στοίβαξη κατά μήκος νέου οριζόντιου άξονα με χρήση του `np.stack`(Κ. 2.4.19). Επίσης με χρήση της παραμέτρου `axis=1` της `np.stack` μπορούμε να δημιουργήσουμε στοίβαξη κατά μήκος νέου κάθετου άξονα (Κ. 2.4.20). Υπάρχει δυνατότητα στοίβαξη σε έναν οριζόντιο άξονα με χρήση της `np.hstack` (Κ. 2.4.21). Ενώ τέλος η `np.dstack` στοιβάζει κατά τον τρίτο άξονα, σαν στρώσεις εικόνας (Κ. 2.4.23).

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = np.stack((a, b))
print(c)
#[[1 2 3]
#       [4 5 6]]
```
Κ. 2.4.19 - Στοίβαξη κατά μήκος νέου οριζόντιου άξονα

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
d = np.stack((a, b), axis=1)
print(d)
#[[1 4]
#       [2 5]
#       [3 6]]
```
Κ. 2.4.20- Στοίβαξη κατά μήκος νέου κάθετου άξονα

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
h = np.hstack((a, b))
print(h)
#[1 2 3 4 5 6]
```
Κ. 2.4.21 -Στοίβαξη σε έναν οριζόντιο άξονα

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
h = np.vstack((a, b))
print(h)
#[[1 2 3]
# [4 5 6]]
```
Κ. 2.4.22-Στοίβαξη (stacking) πινάκων - σε κάθετο άξονα

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
```

```
d = np.dstack((a, b))
print(d)
#[[[1 4]
#[2 5]
#[3 6]]]
```
Κ. 2.4.23 - Στοίβαξη (stacking) πινάκων – Σε βάθος (depth)

2.4.8. Μετάδοση (Broadcasting)

Το Broadcasting στη βιβλιοθήκη NumPy είναι μια ισχυρή λειτουργία που μας επιτρέπει να εκτελούμε αριθμητικές πράξεις σε πίνακες διαφορετικών σχημάτων και μεγεθών απρόσκοπτα. Όταν εκτελείτε πράξεις σε πίνακες με ασύμβατα σχήματα, το Broadcasting επεκτείνει αυτόματα τον μικρότερο πίνακα ώστε να ταιριάζει με το μέγεθος του μεγαλύτερου χωρίς να αντιγράφει στην πραγματικότητα τα δεδομένα. Αυτό καθιστά το Broadcasting έναν αποτελεσματικό και φιλικό προς τη μνήμη τρόπο για την εκτέλεση πράξεων ανά στοιχείο σε πίνακες ποικίλων διαστάσεων.

Η NumPy, είναι η βιβλιοθήκη για αριθμητικούς υπολογισμούς στην Python, έχει κερδίσει δημοτικότητα εν μέρει λόγω τέτοιων χαρακτηριστικών, συμπεριλαμβανομένης του broadcasting. Αυτή η λειτουργικότητα απλοποιεί σημαντικά τη διαδικασία χειρισμού μεγάλων συνόλων δεδομένων και εκτέλεσης σύνθετων μαθηματικών πράξεων, καθιστώντας τον κώδικα πιο ευανάγνωστο και συνοπτικό.

Η μετάδοση στο NumPy ακολουθεί δύο βασικούς κανόνες για να διασφαλίσει ότι είναι δυνατές οι λειτουργίες μεταξύ πινάκων διαφορετικών διαστάσεων:

1. Συμπλήρωση Μικρότερων Πινάκων: Εάν οι εμπλεκόμενοι πίνακες έχουν διαφορετικό αριθμό διαστάσεων, το σχήμα του μικρότερου πίνακα συμπληρώνεται αυτόματα με μονάδες στην αριστερή του πλευρά μέχρι και οι δύο πίνακες να έχουν τον ίδιο αριθμό διαστάσεων.

2. Ταίριασμα διαστάσεων: Για κάθε διάσταση, εάν τα μεγέθη των δύο πινάκων δεν ταιριάζουν, ο πίνακας με μέγεθος 1 σε αυτήν τη διάσταση τεντώνεται για να ταιριάζει με το μέγεθος του άλλου πίνακα. Αυτό λειτουργεί μόνο εάν ένας από τους πίνακες έχει μέγεθος διάστασης 1 ή και οι δύο έχουν το ίδιο μέγεθος σε αυτήν τη διάσταση.

Αυτοί οι κανόνες διασφαλίζουν ότι η μετάδοση μπορεί να επεκτείνει τους πίνακες με τρόπο που επιτρέπει ομαλές και αποτελεσματικές λειτουργίες ανά στοιχείο. Ωστόσο, εάν δεν πληρούνται αυτές οι προϋποθέσεις, το NumPy θα εμφανίσει ένα ValueError.

Ένα παράδειγμα Broadcasting από την πραγματική ζωή είναι η σύγκριση διαφόρων μετρήσεων Σκεφτείτε ένα σενάριο όπου έχουμε μετρήσεις τριών ατόμων, όπως το Ύψος (cm) και το Βάρος (kg) (Πίνακας 2.4.1 - Δεδομένα Ύψους και Βάρους

).

Person A	**170**	**72**
Person B	187	83
Person C	175	68

Πίνακας 2.4.1 - Δεδομένα Ύψους και Βάρους

Μπορούμε να αναπαραστήσουμε αυτά τα δεδομένα χρησιμοποιώντας πίνακες NumPy, όπως φαίνεται στον Κ. 2.4.24 - Κώδικας Python για δημιουργία πίνακα Υψών και Βαρών Numpy.

```
#Πίνακας Υψών και Βαρών
import numpy as np
heights = np.array([170, 187, 175])
weights = np.array([72, 83, 68])
#Οι ακόλουθες γραμμές είναι απλώς για σκοπούς εμφάνισης, επομένως η προσθήκη τους είναι εντελώς προαιρετική
print("Heights:", heights) #Heights: [170 187 175]
print("Weights:", weights) #Weights: [72 83 68]
```

Κ. 2.4.24 - Κώδικας Python για δημιουργία πίνακα Υψών και Βαρών Numpy

Μπορούμε να αξιοποιήσουμε τη μετάδοσης (Broadcasting) ώστε να υπολογίζουμε το Δείκτη Μάζας Σώματος (ΔΜΣ) και για τα τρία άτομα ταυτόχρονα, αυτό γίνεται αν στον Κ. 2.4.24 προσθέσουμε τις ακόλουθες γραμμές κώδικα (Κ. 2.4.25). Η εφαρμογή της ιδιότητας μετάδοσης της βιβλιοθήκης NumPy κάνει αυτόν τον υπολογισμό απλό. Η έξοδος που εμφανίζεται ως το τελευταίο σχόλιο του κώδικα Κ. 2.4.25 αντιπροσωπεύει τις τιμές ΔΜΣ για τα Άτομα Α, Β και Γ, αντίστοιχα.

```
#Υπολογισμός Δείκτη Μάζας Σώματος (BMI)
heights_meters = heights / 100  # Μετατρέπει τα ύψη από εκατοστά σε μέτρα
bmi = weights / (heights_meters ** 2)
print(bmi) #Υπολογισμός Δείκτη Μάζας Σώματος (BMI): [24.91349481 23.73530842 22.20408163]
```

Κ. 2.4.25 - Υπολογισμός Δείκτη Μάζας Σώματος (BMI)

Στα προηγούμενα παραδείγματα, χρησιμοποιήσαμε μερικές βασικές συναρτήσεις NumPy:

- numpy.array(): Δημιουργεί ένα πίνακα NumPy από συγκεκριμένα δεδομένα. Σύνταξη: numpy.array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)
- numpy.ndarray. truediv (other): Εκτελεί διαίρεση ανά στοιχείο μεταξύ δύο πινάκων. Σύνταξη : array1. truediv (array2)
- numpy.ndarray. pow (other): Ανεβάζει κάθε στοιχείο του πίνακα στη δύναμη του αντίστοιχου στοιχείου σε έναν άλλο πίνακα. Σύνταξη: array1. pow (array2)

Αυτές οι λειτουργίες, όταν συνδυάζονται με τη μετάδοσης (Broadcasting), επιτρέπουν αποτελεσματικούς αριθμητικούς υπολογισμούς. Ας εξερευνήσουμε μερικά επιπλέον παραδείγματα μετάδοσης για να δείξουμε την ευελιξία της. Στο παρακάτω παράδειγμα (Κ. 2.4.26) η scalar τιμή 5 μεταδίδεται/εφαρμόζεται (πρόσθεση εδώ) σε κάθε στοιχείο του arr.

```
import numpy as np
arr = np.array([1, 2, 3])
scalar = 5
result = arr + scalar
print(result)
#[6 7 8]
```

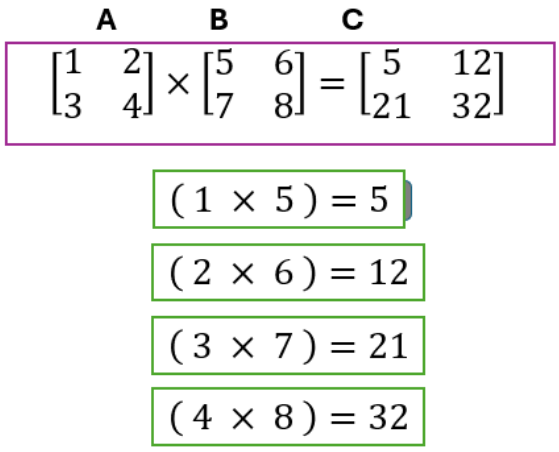
Κ. 2.4.26 - Παράδειγμα 1: Προσθήκη μιας βαθμωτής (scalar) τιμής σε έναν πίνακα τιμών

Σε αυτήν την περίπτωση, ο πίνακας arr2 «μεταδίδεται» ώστε να ταιριάζει με το σχήμα του πίνακα arr1. Προσοχή, δεν γίνεται πολλαπλασιασμός πινάκων όπως αυτός είναι γνωστός στην άλγεβρα πινάκων, αλλά γίνεται πολλαπλασιασμός μεταξύ στοιχείων (element-wise) που βρίσκονται σε αντίστοιχη θέση.

```
import numpy as np
arr1 = np.array([[1, 2, 3], [4, 5, 6]])
arr2 = np.array([10, 20, 30])
result = arr1 * arr2
print(result)
#[[ 10 40 30]
#[ 40 100 180]]
```

Κ. 2.4.27 - Παράδειγμα 2: «Πολλαπλασιασμός» πινάκων με διαφορετικά σχήματα

Στην περίπτωση που έχουμε δύο πίνακες των ίδιων διαστάσεων ο «Πολλαπλασιασμός» είναι μεταξύ στοιχείων (element-wise) που βρίσκονται στην ίδια θέση. Στην Εικόνα 2.4.1 – «Πολλαπλασιασμός» Πινάκων ίδιων διαστάσεων φαίνεται ο τρόπος που γίνεται ο «πολλαπλασιασμός» όπως ορίζεται στη βιβλιοθήκη Numpy. Επίσης ο Κ. 2.4.28 είναι ο αντίστοιχος κώδικας σε Python που υλοποιεί αυτού του τύπου τον πολλαπλασιασμό.



Εικόνα 2.4.1 – «Πολλαπλασιασμός» Πινάκων ίδιων διαστάσεων

```
import numpy as np
```

```
arr1 = np.array([[1,2],[3,4]])
arr2 = np.array([[5,6],[7,8]])
print(np.multiply(arr1,arr2))
#[[ 5 12]
#[21 32]]
```
Κ. 2.4.28 - Πολλαπλασιασμός πινάκων ίδιων διαστάσεων

2.4.9. Συνήθεις παγίδες της μετάδοσης (Broadcasting)

Ενώ η μετάδοση είναι εξαιρετικά χρήσιμη, μπορεί να οδηγήσει σε πιθανές παγίδες εάν δεν χρησιμοποιηθεί σωστά. Έτσι μια πρώτη παγίδα είναι οι ασύμβατες διαστάσεις. Εάν οι πίνακες έχουν ασύμβατες διαστάσεις, η μετάδοση θα αποτύχει, δημιουργώντας ένα `ValueError`.

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5])
result = arr1 + arr2
```
Κ. 2.4.29 - Πίνακες έχουν ασύμβατες διαστάσεις

```
ValueError                                Traceback (most recent call
last)
/tmp/ipython-input-949947271.py in <cell line: 0>()
      2 arr1 = np.array([1, 2, 3])
      3 arr2 = np.array([4, 5])
----> 4 result = arr1 + arr2

ValueError: operands could not be broadcast together with shapes (3,)
(2,)
```
Ε. 2.4.1 - Μήνυμα σφάλματος για πράξη μεταξύ πινάκων με ασύμβατες διαστάσεις

Επίσης εάν οι διαστάσεις των πινάκων δεν είναι σωστά ευθυγραμμισμένες, η μετάδοση ενδέχεται να αποφέρει αποτελέσματα που διαφέρουν από τα αναμενόμενα, όπως στο παράδειγμα του Κ. 2.4.30 - Μη αναμενόμενα αποτελέσματα εφαρμογής μετάδοσης. Εδώ, το arr2 μεταδίδεται κατά μήκος των σειρών του arr1.

```
import numpy as np
arr1 = np.array([[1, 2, 3], [4, 5, 6]])
arr2 = np.array([[10], [20]])
result = arr1 + arr2
print(result) #[[11 12 13]
                [24 25 26]]
```
Κ. 2.4.30 - Μη αναμενόμενα αποτελέσματα εφαρμογής μετάδοσης

2.5. Συναρτήσεις, Αλγεβρικές Πράξεις και Έλεγχοι Στοιχείων Πινάκων

2.5.1. Συναρτήσεις συνάθροισης

Στη NumPy υπάρχουν πολλές συναρτήσεις συνάθροισης (aggregation functions) που υπολογίζουν περιγραφικούς στατιστικούς δείκτες κεντρικής θέσεις και διασποράς από δεδομένα σε πίνακες, δείκτες όπως τα max, min, sum, mean κ.α. Η χρήση των πιο βασικών από αυτές παρουσιάζεται στο Κ. 2.5.1

```
import numpy as np
a = np.array([[1, 2, 3],
              [4, 5, 6]])
print("Sum:", np.sum(a))        #Άθροισμα: 21
print("Min:", np.min(a))        #Ελάχιστο: 1
print("Max:", np.max(a))        #Μέγιστο: 6
print("Mean:", np.mean(a))      #Μέσος Όρος: 3.5
print("Median:", np.median(a)) #Διάμεσος: 3.5
print("Std Dev:", np.std(a))    #Τυπική Απόκλιση: 1.707...
print("Variance:", np.var(a))   #Διακύμανση: 2.916...
```

Κ. 2.5.1 - Συναρτήσεις συνάθροισης της βιβλιοθήκης Numpy

Μπορούμε να υπολογίσουμε τιμές κατά γραμμές ή κατά στήλες με χρήση της παραμέτρου axis, όπως φαίνεται στο Κ. 2.5.2.

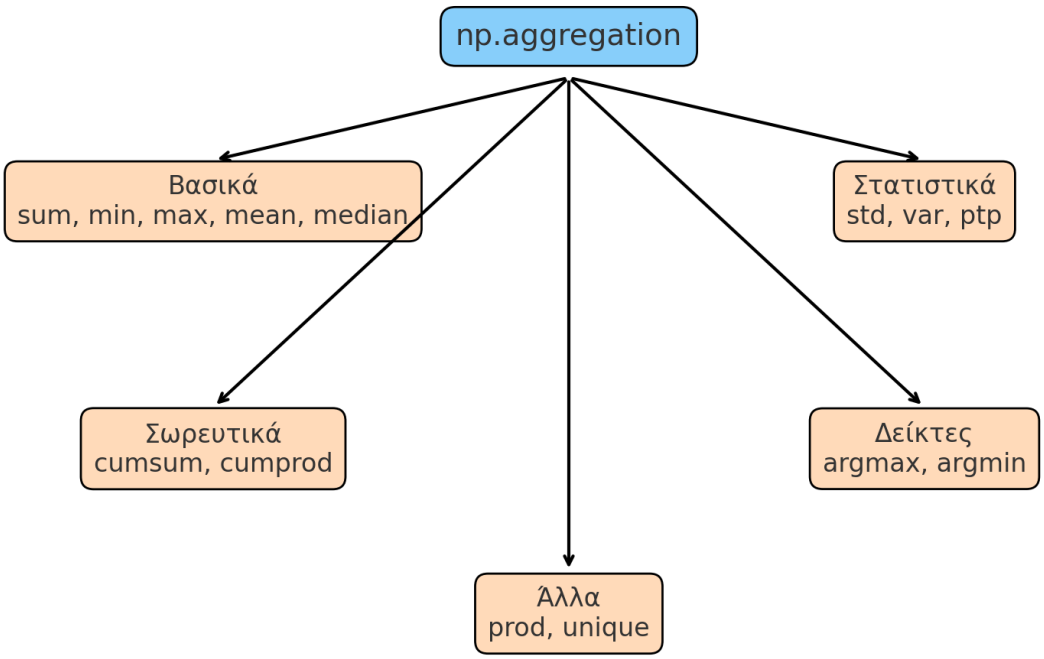
```
import numpy as np
a = np.array([[1, 2, 3],
              [4, 5, 6]])
print("Άθροισμα ανά στήλη:", np.sum(a, axis=0)) #Άθροισμα ανά στήλη: [5 7
9]
print("Άθροισμα ανά γραμμή:", np.sum(a, axis=1)) #Άθροισμα ανά γραμμή: [ 6
15]
print("Μέγιστο ανά στήλη:", np.max(a, axis=0)) #Μέγιστο ανά στήλη: [4 5 6]
print("Ελάχιστο ανά γραμμή:", np.min(a, axis=1)) #Ελάχιστο ανά γραμμή: [1
4]
```

Κ. 2.5.2 - Συναρτήσεις συνάθροισης κατά άξονα

Υπάρχουν και άλλες συναρτήσεις συνάθροισης, όπως οι cumsum, cumprod, argmax, argmin και unique. Η χρήση τους φαίνεται στον Κ. 2.5.3 - Άλλες συναρτήσεις συνάθροισης.

```
import numpy as np
a = np.array([[1, 2, 3],
              [4, 5, 6]])
print("Σωρευτικό άθροισμα:", np.cumsum(a)) #Σωρευτικό άθροισμα: [ 1 3 C
10 15 21]
print("Σωρευτικό γινόμενο:", np.cumprod(a)) #Σωρευτικό γινόμενο: [ 1 2
C 24 120 720]
print("Θέση του μέγιστου:", np.argmax(a))  #Θέση του μέγιστου (5)
print("Θέση του ελάχιστου:", np.argmin(a))  #Θέση του ελάχιστου (0)
print("Μοναδικές τιμές:", np.unique([1,2,2,3,3,3])) #Μοναδικές τιμές:
[1 2 3]
```

Κ. 2.5.3 - Άλλες συναρτήσεις συνάθροισης



Εικόνα 2.5.1 - Όλες οι συναρτήσεις συνάθροισης

2.5.2. Πολλαπλασιασμός Πινάκων

Ο τρόπος υλοποίησης του αλγεβρικού πολλαπλασιασμού πινάκων με χρήση της βιβλιοθήκης Numpy της Python είναι κάπως διαφορετικών. Αρχικά ας τον ορίσουμε σύμφωνα με την άλγεβρα πινάκων. Ο πολλαπλασιασμός πινάκων στην άλγεβρα πινάκων γίνεται όπως παρουσιάζεται στην Εικόνα 2.5.2, από την εικόνα είναι εύκολο να καταλάβει κάποιος πως γίνεται ο αλγεβρικός πολλαπλασιασμός πινάκων. Η βιβλιοθήκη Numpy της Python έχει τη μέθοδο dot () των πινάκων της βιβλιοθήκης Numpy ώστε να υλοποιεί τον αλγεβρικό πολλαπλασιασμό πινάκων. Η υλοποίηση σε κώδικα φαίνεται στο Κ. 2.5.4.

$$\underset{A}{\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}} \times \underset{B}{\begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix}} = \underset{C}{\begin{bmatrix}19 & 22\\ 43 & 50\end{bmatrix}}$$

$$(1 \times 5) + (2 \times 7) = 19$$
$$(1 \times 6) + (2 \times 8) = 22$$
$$(3 \times 5) + (4 \times 7) = 43$$
$$(3 \times 6) + (4 \times 8) = 50$$

Εικόνα 2.5.2 - Αλγεβρικός Πολλαπλασιασμός Πινάκων

```
import numpy as np
A = np.array([[1,2],[3,4]])
B = np.array([[5,6],[7,8]])
```

```
print(np.dot(A,B))
#[[19 22]
#[43 50]]
```
Κ. 2.5.4 - Αλγεβρικός Πολλαπλασιασμός Πινάκων με χρήση της βιβλιοθήκης Numpy

Επίσης υπάρχει και άλλος ένας τρόπος για να γίνει ο πολλαπλασιασμός μεταξύ δύο πινάκων, με τη χρήση της συνάρτησης multiply() (Κ. 2.5.5). Σε αυτή την περίπτωση οι διαστάσεις των δύο πινάκων πρέπει να ταιριάζουν.

```
import numpy as np
row1 = [10,12,13]
row2 = [45,32,16]
row3 = [45,32,16]
nums_2d = np.array([row1, row2, row3])
multiply = np.multiply(nums_2d, nums_2d)
print (multiply) #[[ 100  144  169]
                  [2025 1024  256]
                  [2025 1024  256]]
```
Κ. 2.5.5 - Αλγεβρικός Πολλαπλασιασμός Πινάκων με χρήση της συνάρτησης multiply της βιβλιοθήκης Numpy

Στην άλγεβρα πινάκων ορίζεται η έννοια του αναστρέψιμου πίνακα. Για να γίνει η αναστροφή του πίνακα υπολογίζουμε την ορίζουσα (determinant) και μετά προσπαθούμε να υπολογίσουμε τον ανάστροφο (inverse). Ένας πίνακας A είναι αναστρέψιμος αν και μόνο αν η ορίζουσά του det(A)≠0. Με τη βοήθεια της βιβλιοθήκης NumPy της Python μπορεί να υπολογιστεί ο ανάστροφος πίνακας με χρήση της numpy.linalg.inv(A), εφόσον υπάρχει ο ανάστροφος πίνακας. Ο κώδικας είναι στο Κ. 2.5.1.

```
import numpy as np
# Ορισμός πίνακα
A = np.array([[2, 1],[5, 3]])
# Υπολογισμός ορίζουσας
det_A = np.linalg.det(A)
print("Ορίζουσα:", det_A)
# Έλεγχος αν είναι αναστρέψιμος
if det_A != 0:
    print("Ο πίνακας είναι αναστρέψιμος.")
    A_inv = np.linalg.inv(A)
    print("ανάστροφος πίνακας:\n", A_inv)
else:
    print("Ο πίνακας ΔΕΝ είναι αναστρέψιμος.")
```
Κ. 2.5.6 Αντιστρέψιμος πίνακας

2.5.3. Διαίρεση πινάκων

Στη βιβλιοθήκη NumPy η «διαίρεση πινάκων» δεν είναι ορισμένη με την κλασική έννοια (όπως π.χ. η πρόσθεση ή ο πολλαπλασιασμός). Στην άλγεβρα πινάκων, όταν λέμε ότι «διαιρούμε» έναν πίνακα A με έναν πίνακα B, στην πραγματικότητα εννοούμε $A/B \equiv A \cdot B^{-1}$ δηλαδή πολλαπλασιασμός του A με τον αντίστροφο του B, εφόσον ο B είναι αναστρέψιμος.

Έστω λοιπόν οι πίνακες

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & 0 \\ 1 & 3 \end{pmatrix}$$

Προϋπόθεση για να γίνει ο πολλαπλασιασμός ενός πίνακα *A* με τον αντίστροφο πίνακα του *B* είναι ο πίνακας B να είναι αναστρέψιμος (διακρίνουσα του B - det(B)≠0). Επίσης, στην γλώσσα προγραμματισμού Python πάντα χρησιμοποιούμε A @ np.linalg.inv(B) και ποτέ A / B για αληθινή διαίρεση πινάκων (Κ. 2.5.7).

```
import numpy as np

# Ορισμός πινάκων
A = np.array([[1, 2],
              [3, 4]])

B = np.array([[2, 0],
              [1, 3]])

# Υπολογισμός αντιστρόφου του B
B_inv = np.linalg.inv(B) #Υπολογίζει τον αντίστροφο B^(-1)
print("Αντίστροφος B^(-1):")
print(B_inv)

# Διαίρεση: A / B ≡ A @ B^(-1) A / B με / Κάνει στοιχειοκεντρική διαίρεση — δεν είναι διαίρεση πινάκων!
result = A @ B_inv #Εκτελεί πολλαπλασιασμό πινάκων (όχι στοιχειοκεντρικό)
print("\nΑποτέλεσμα A / B = A @ B^(-1):")
print(result)
#Αντίστροφος B-1:
#[[ 0.5         0.        ]
$ [-0.16666667  0.33333333]]

#Αποτέλεσμα A / B = A @ B-1:
#[[ 0.16666667  0.66666667]
$ [ 0.83333333  1.33333333]]
```

Κ. 2.5.7 Πολλαπλασιαμός πίνακα με αντίστροφο δεύτερου πίνακα

2.5.4. Βασικοί Τελεστές - Σύγκριση στοιχείων πινάκων

Η στοιχειώδης (element-wise) λογική σύγκριση στοιχείων πινάκων ένα προς ένα στη βιβλιοθήκη NumPy γίνεται με τελεστές ή με συναρτήσεις. Στον κώδικα Κ. 2.5.8 είναι με τελεστές σύγκρισης.

```
import numpy as np
a = np.array([1, 2, 3, 4])
b = np.array([1, 0, 4, 4])
a == b # [ True False False True]
a != b # [False True True False]
a < b # [False False True False]
a >= b # [ True True False True]
```

Κ. 2.5.8 - Λογική σύγκριση στοιχείων πινάκων

Εναλλακτικά η βιβλιοθήκη Numpy μας δίνει συναρτήσεις όπως η equal, not_equal, greater, greater_equal, less, less_equal, όπου ενεργούν ως ισοδύναμες συναρτήσεις με τους λογικούς

τελεστές. Ο κώδικας χρήσης τους και τα αποτελέσματα που δίνουν σε ενδεικτικό παράδειγμα φαίνονται στον Κ. 2.5.9.

```
import numpy as np
a = np.array([1, 2, 3, 4])
b = np.array([1, 0, 4, 4])
np.equal(a, b) # a==b - array([ True, False, False, True])
np.not_equal(a, b) # a!b - array([False, True, True, False])
np.greater(a, b) # a > b - array([False, True, False, False])
np.greater_equal(a, b) # a >= b - array([ True, True, False, True])
np.less(a, b) # a < b - array([False, False, True, False])
np.less_equal(a, b) # a <= b - array([ True, False, True, True])
```

Κ. 2.5.9 - Συναρτήσεις σύγκρισης πινάκων

Εάν θέλουμε να συγκρίνουμε στοιχεία πινάκων στοιχείο προς στοιχείο αλλά με διαφορετικές διαστάσεις, τότε η δυνατότητα εφαρμογής της τεχνικής μετάδοσης (broadcasting) της βιβλιοθήκης Numpy πάλι μπορεί να μας διευκολύνει (Κ. 2.5.10, γραμμές κώδικα 4-5), ενώ σύγκριση μπορεί να γίνει και με έναν αριθμό (scalar - Κ. 2.5.10, γραμμές κώδικα 6-7).

```
import numpy as np
A = np.array([[1,2,3], [4,5,6]])
v = np.array([1,2,3])
A == v # συγκρίνει κάθε γραμμή με το v (broadcast) - array([[ True, True,
True], #[False, False, False]])
A > 3 # συγκρίνει με scalar - array([[False, False, False],
       #[ True,  True,  True]])
```

Κ. 2.5.10 - Σύγκριση στοιχείων πινάκων με διαφορετικές διαστάσεις

2.5.5. Σύγκριση πινάκων στοιχείων προς στοιχείο

Η βιβλιοθήκη Numpy διαθέτει τρόπο σύγκρισης στοιχείων των πινάκων ένα προς ένα στοιχείο και με λογικές μάσκες(Boolean masks) ή επιλογή στοιχείων. Κ. 2.5.11 δίνεται ενδεικτικό παράδειγμα διαφόρων επιλογών. Στη γραμμή 4 δημιουργείται μια μάσκα η οποία εφαρμόζεται ως φίλτρο στο πίνακα a στη γραμμή 5, με πολύ λιτό τρόπο. Στη γραμμή 6 έχουμε μια περίπτωση εφαρμογής if/else ανά στοιχείο, στη γραμμή 7 ταίριασμα θέσεων και στη γραμμή 8 την αναζήτηση σε συντεταγμένες δισδιάτες και άνω.

```
import numpy as np
a = np.array([1, 2, 3, 4])
b = np.array([1, 0, 4, 4])
mask = (a != b) & (a > 1)     # προσοχή στις παρενθέσεις με &, |, ~ -
a[mask]                       # φιλτράρισμα με μάσκα.
Απάντηση: array([2, 3])
np.where(a == b, a, -1)       # if/else ανά στοιχείο
Απάντηση: array([ 1, -1, -1,  4])
np.nonzero(a == b)            # θέσεις που ταιριάζουν
Απάντηση: (array([0, 3]),)
np.argwhere(a == b)           # συντεταγμένες (2D+) που ταιριάζουν
Απάντηση: array([[0], [3]])
```

Κ. 2.5.11 - Σύγκριση πινάκων με Λογικές Μάσκες και επιλογή στοιχείων

Υπάρχουν όμως και ακόμα περισσότερες επιλογές για σύγκριση πινάκων στοιχείων προς στοιχείο με έλεγχο τύπου “όλα/κάποιο”. Στον κώδικα Κ. 2.5.11 μπορούμε να συμπληρώσουμε τις γραμμές του Κ. 2.5.12, στον οποίο η πρώτη γραμμή κάνει έλεγχο αν είναι όλα ίδια και η δεύτερη αν υπάρχει κάποιο μεγαλύτερο

```
(np.all(a == b)) # όλα ίδια;
Απάντηση: np.False_
(np.any(a > b)) # υπάρχει κάποιο μεγαλύτερο;
Απάντηση: np.True_
```

Κ. 2.5.12 - Έλεγχος τύπου όλα/κάποιο

Στη βιβλιοθήκη Numpy υπάρχει δυνατότητα για σύγκριση πινάκων στοιχείων προς στοιχείο -σε περίπτωση αριθμούς κινητής υποδιαστολής (float) με διακριτό τρόπο με χρήση ανοχών. Συστήνεται να μην χρησιμοποιείται ποτέ μόνο το == για αριθμούς κινητής υποδιαστολής (floats), αλλά πάντα να θέτουμε ανοχές. Ο Κ. 2.5.2 δίνει ένα σχετικό παράδειγμα όπου το rtol είναι η σχετική ανοχή, αυτή η παράμετρος ορίζει το αποδεκτό σχετικό σφάλμα μεταξύ της εκτιμώμενης λύσης και της ακριβούς λύσης. Μια μικρότερη τιμή υποδεικνύει μια αυστηρότερη ανοχή. Ενώ το atol είναι η απόλυτη ανοχή, αυτή η παράμετρος καθορίζει το αποδεκτό απόλυτο σφάλμα. Ορίζει ένα όριο για την ελάχιστη αποδεκτή διαφορά μεταξύ της εκτιμώμενης λύσης και της ακριβούς λύσης.

```
import numpy as np
x = np.array([0.3 + 0.6, 1.0])
y = np.array([0.9, 1.0])
np.isclose(x, y, rtol=1e-05, atol=1e-08)   # [ True  True]
np.allclose(x, y)                           # True
```

Πίνακας 2.5.1 - Σύγκριση αριθμών κινητής υποδιαστολής

Με τη βιβλιοθήκη Numpy μπορούμε να διαχειριστούμε και την περίπτωση συγκρίσεων για τιμές μη αριθμητικές (Not a Number – NaN) ή και άλλες ειδικές περιπτώσεις. Παραδείγματα δίνονται στο Κ. 2.5.13. Η πρώτη σύγκριση (γραμμή 3η του κώδικα) αφορά σύγκριση του πίνακα με τον εαυτό του. Η isnan για έλεγχο αν κάποιο στοιχείο είναι NaN, η array_equal για ακόμα ένα τρόπο ελέγχου ισότητας του πίνακα με τον εαυτό του και array_equal με παράμετρο equal_nan=True όπου με αυτόν τον τρόπο μετριόνται τα NaN ως ίσα Κ. 2.5.13.

```
import numpy as np
z = np.array([np.nan, 1.0])
z == z # [False True] (το NaN δεν ισούται με τον εαυτό του)
np.isnan(z) # [ True False]
np.array_equal(z, z) # False (το NaN δεν ισούται με τον εαυτό του)
np.array_equal(z, z, equal_nan=True) # True (μετρώντας τα NaN ως ίσα)
```

Κ. 2.5.13- NaN και άλλες ειδικές περιπτώσεις

Γενικότερες συμβουλές για τη σύγκριση πινάκων στοιχείων προς στοιχείο είναι οι ακόλουθες:

- Οι τελεστές &, |, ~ είναι για πίνακες με λογικά στοιχεία (boolean arrays) και όχι τα and, or, not.
- Καλή πρακτική είναι η χρήση παρενθέσεων σε σύνθετες λογικές εκφράσεις.

• Για σύγκριση ολόκληρων πινάκων ως προς ισότητα στοιχείων προς-στοιχείο και μετά για έλεγχο “όλα ίσα” καλύτερα είναι να χρησιμοποιείται η np.all(a == b).

2.6. Δημιουργία Μάσκας στη βιβλιοθήκη Numpy

Μια εξαιρετικά σημαντική δυνατότητα της βιβλιοθήκης NumPy είναι η δημιουργία μάσκας (masking). Αυτό σημαίνει ότι μπορούμε να δημιουργούμε λογικούς πίνακες (boolean tables) ως μάσκες για να επιλέξουμε, τροποποιήσουμε ή διαχειριστούμε στοιχεία από έναν άλλο πίνακα. Στον Κ. 2.6.1 φαίνεται η εφαρμογή μάσκας λογικού ελέγχου για όλα τα στοιχεία ενός πίνακα και ουσιαστικά η μετατροπή του πίνακα σε λογικές τιμές από αριθμητικές. Η χρήση μάσκας βρίσκει πολλές εφαρμογές στην ψηφιακή επεξεργασία σήματος και εικόνας με εξαιρετικά μεγάλο εύρος εφαρμογών.

```
import numpy as np
a = np.array([1, 2, 3, 4, 5, 6])
mask = a > 3
print(mask) # [False False False True True True]
```

Κ. 2.6.1 - Εφαρμογή Μάσκας σε πίνακα

Με τη χρήση μάσκας μπορούμε να επιλέξουμε στοιχεία, όπως στον κώδικα Κ. 2.6.2 όπου επιλέγει μόνο τους ζυγούς αριθμούς.

```
import numpy as np
a = np.array([1, 2, 3, 4, 5, 6])
print(a[mask]) # [4 5 6]
# Απευθείας χωρίς ενδιάμεση μεταβλητή
print(a[a % 2 == 0]) # [2 4 6] (μόνο τα ζυγά)
```

Κ. 2.6.2 - Επιλογή στοιχείων με μάσκα

Επίσης με τη χρήση μάσκας μπορεί να τροποποιηθούν κατάλληλα τα στοιχεία ενός πίνακα, όπως στον κώδικα Κ. 2.6.3, όπου όσα στοιχεία του πίνακα έχουν αριθμό μικρότερο του 3 τα μετατρέπει σε 0.

```
import numpy as np
a = np.array([1, 2, 3, 4, 5,6])
a[a < 3] = 0
print(a) # [0 0 3 4 5 6]
```

Κ. 2.6.3 - Τροποποίηση με μάσκα

Με τη βιβλιοθήκη Numpy είναι εφικτό να χρησιμοποιήσουμε και λογικούς τελεστές ως μάσκες. Στον κώδικα του παραδείγματος Κ. 2.6.4 δημιουργείται ένα πίνακα από το 0 έως το δέκα και στην πρώτη περίπτωση εφαρμογής λογικού τελεστή ως μάσκα **(**`(a > 2) & (a < 8)`**)** επιλέγονται οι αριθμοί μεγαλύτεροι από το 2 και μικρότεροι από το 8 και δημιουργείται ένα πίνακας μικρότερων διαστάσεων. Στη δεύτερη περίπτωση **(**`(a % 2 == 0) | (a > 7)`**)** οι αριθμοί όπου το υπόλοιπο της διαίρεσης τους με το 2 είναι μηδέ, οι ζυγοί δηλαδή ή (|) όσοι είναι μεγαλύτεροι του 7. Στην τρίτη περίπτωση **(**`~(a > 5)`**)** όλοι όσοι δεν (~) είναι μεγαλύτεροι του 5.

```
import numpy as np
a = np.arange(10)
```

```
print(a[(a > 2) & (a < 8)]) # [3 4 5 6 7]
print(a[(a % 2 == 0) | (a > 7)]) # [0 2 4 6 8 9]
print(a[~(a > 5)]) # [0 1 2 3 4 5]
```
Κ. 2.6.4 - Λογικοί τελεστές ως μάσκα

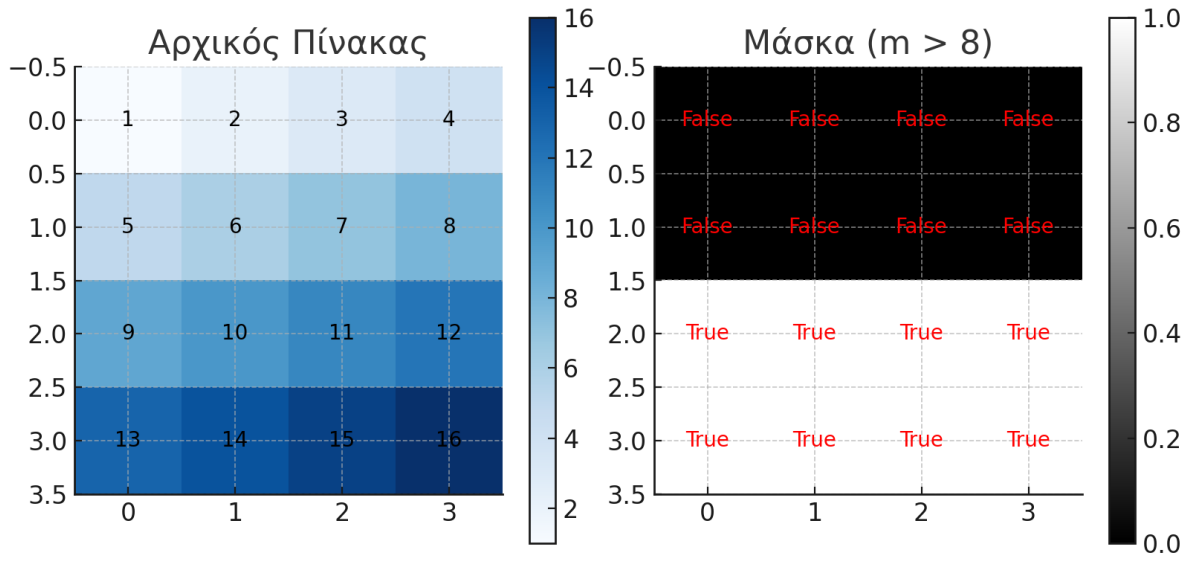
Υπάρχει δυνατότητα επιλογής και με δημιουργία λογικών τύπου με if/else ανά στοιχείο. Στον Κ. 2.6.5 βλέπουμε ένα παράδειγμα με χρήση της μεθόδου *where* η οποία έχει ως παραμέτρους τη μάσκα (`mask = a > 25`), τον πίνακα και την τιμή που θα ανατεθεί αν αληθεύει η λογική σύγκριση της μάσκας.

```
a = np.array([10, 20, 30, 40])
mask = a > 25
b = np.where(mask, a, -1)
print(b)    # [-1 -1 30 40]
```
Κ. 2.6.5 - Επιλογή με λογικό τύπου if/else

Η εφαρμογή μάσκας μπορεί να επεκταθεί και σε δισδιάστατους πίνακες (εφαρμογή στην ψηφιακής επεξεργασία εικόνας. Στον κώδικα Κ. 2.6.6, εφαρμόζεται στον πίνακα m μάσκα για να βρει τις ζυγές τιμές. Η εκτύπωση της μάσκας και η εκτύπωση εφαρμογής της μάσκας στον πίνακα φαίνονται διαδοχικά στις δύο τελευταίες γραμμές του κώδικα Κ. 2.6.6 και κάποιος μπορεί να διαπιστώσει την ευελιξία που παρέχει η χρήση μάσκας, τόσο ώστε ένας πίνακας να μετατρέπεται σε λογικές τιμές ή να επιλέγονται στοιχεία του, βάση λογικής συνάρτησης επιλογής

```
m = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
mask = m % 2 == 0 # True όπου το στοιχείο είναι ζυγό
print(mask) # [[False True False] [ True False True] [False True False]]
print(m[mask]) # [2 4 6 8]
```
Κ. 2.6.6 - Μάσκες σε 2D πίνακες

Η εφαρμογή μάσκας αν συνδυαστεί με τη βιβλιοθήκη plot της Python μπορεί να έχει εξαιρετικά χρήσιμες εφαρμογές. Για τη βιβλιοθήκη Matplotlib γίνεται εκτενείς αναφορά στην ενότητα 4.2. Περιγραφική Στατιστική: Οπτική Προσέγγιση. Στην εικόνα Εικόνα 2.6.2Εικόνα 2.6.1 αποτυπώνεται γραφικά στην αριστερή πλευρά της εικόνας η απεικόνιση του αρχικού πίνακα με τιμές 1 έως 16, ενώ στη δεξιά πλευρά η απεικόνιση της μάσκας (m > 8) όπου αποτυπώνονται True ως λευκό, για τα στοιχεία > 8 και False ως μαύρο για τα υπόλοιπα στοιχεία. Για την απεικόνιση αυτή έχει χρησιμοποιηθεί και η βιβλιοθήκη plot για την οποία υπάρχει εκτενείς αναφορά σε επόμενο κεφάλαιο. Σε αυτό το σημείο του εγχειριδίου ο κώδικας από τη βιβλιοθήκη Matplotlib θα αξιοποιηθεί μόνο για να επιδειχθεί η δυνατότητα απεικόνισης με συνδυαστική χρήση των βιβλιοθηκών Numpy και Matplotlib. Στον κώδικα Κ. 2.6.7 - Κώδικας εφαρμογής μάσκας (m > 8) όπου εμφανίζονται True (λευκό) για τα στοιχεία > 8 και False (μαύρο) και απεικόνιση υπάρχουν σχόλια για το τι κάνει κάθε τμήμα του κώδικα Δημιουργία 2D πίνακα, Ορισμός μάσκας, Σχεδίαση, Αρχικός πίνακας, Σχεδίαση Απεικόνισης Αρχικού πίνακα, Μάσκα, Σχεδίαση Απεικόνισης πίνακα Μάσκας, Εμφάνιση σχεδίων.



Εικόνα 2.6.2 - Εφαρμογή μάσκας (m > 8) όπου εμφανίζονται True (λευκό) για τα στοιχεία > 8 και False (μαύρο) και απεικόνιση

```
import numpy as np
import matplotlib.pyplot as plt
# Δημιουργία 2D πίνακα
m = np.arange(1, 17).reshape(4, 4)

# Ορισμός μάσκας: True όπου τα στοιχεία > 8
mask = m > 8

# Σχεδίαση
fig, axs = plt.subplots(1, 2, figsize=(8, 4))

# Αρχικός πίνακας
im1 = axs[0].imshow(m, cmap="Blues", interpolation="nearest")
axs[0].set_title("Αρχικός Πίνακας")

#Σχεδίαση Απεικόνισης Αρχικού πίνακα
for (i, j), val in np.ndenumerate(m):
    axs[0].text(j, i, val, ha="center", va="center", color="black")
fig.colorbar(im1, ax=axs[0])

# Μάσκα
im2 = axs[1].imshow(mask, cmap="gray", interpolation="nearest")
axs[1].set_title("Μάσκα (m > 8)")

#Σχεδίαση Απεικόνισης πίνακα Μάσκας
for (i, j), val in np.ndenumerate(mask):
    axs[1].text(j, i, str(val), ha="center", va="center", color="red")
fig.colorbar(im2, ax=axs[1])

#Εμφάνιση γραφικών
plt.tight_layout()
plt.show()
```

Κ. 2.6.7 - Κώδικας εφαρμογής μάσκας (m > 8) όπου εμφανίζονται True (λευκό) για τα στοιχεία > 8 και False (μαύρο) και απεικόνιση

2.7. Προχωρημένες τεχνικές ευρετηρίασης

Σε αυτή την ενότητα θα μελετήσουμε κάποιες περιπτώσεις προχωρημένων τεχνικών ευρετηρίασης. Μια πρώτη περίπτωση είναι η λήψη πολλαπλών στοιχείων πίνακα. Αρχικά ας κάνουμε μια μικρή επανάληψή με ένα παράδειγμα βασικής ευρετηρίασης με «τεμαχισμό» πίνακα. Στον κώδικα **Κ. 1.4.1** γίνεται «τεμαχισμός» του πίνακα a σε δύο πίνακες με χρήση ευρετηρίασης.

```
import numpy as np
a = np.arange(10)   # [0 1 2 3 4 5 6 7 8 9]
print(a[2:6])   # [2 3 4 5]
print(a[::2])   # [0 2 4 6 8]
```

Κ. 2.7.1 - Ευρετηρίασης με «τεμαχισμό» πίνακα

Ας δούμε όμως κάποιους ιδιαίτερους τρόπους χρήσης αυτής της δυνατότητας στον κώδικα Κ. 2.7.2 αρχικά η συνάρτηση `np.arange(10)` δημιουργεί έναν πίνακα ακεραίων από 0 έως 9, συμπεριλαμβανομένων. Ο πίνακας a που προκύπτει είναι ο `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`

Η γραμμή `a[[2, 4, 6]]` χρησιμοποιεί την προηγμένη λειτουργία ευρετηρίασης του NumPy για να επιλέξει συγκεκριμένα στοιχεία του πίνακα a στους δείκτες που παρέχονται στη λίστα, δηλαδή 2, 4, 6. Έτσι, η εκτυπωμένη έξοδος είναι ο πίνακας που περιέχει αυτά τα επιλεγμένα στοιχεία: [2, 4, 6]. Προσοχή, ο αριθμός του δείκτη είναι ίδιος με την τιμή του πίνακα επειδή γίνεται χρήση του συγκεκριμένου πίνακα a.

```
import numpy as np
a = np.arange(10)
print(a[[2, 4, 6]])   # [2 4 6]
```

Κ. 2.7.2 - Λίστα δεικτών - Fancy indexing

Αντίστοιχη εφαρμογή υπάρχει και σε πίνακες δύο διαστάσεων (2D). Παράδειγμα τέτοιας εφαρμογής έχουμε στον κώδικα Κ. 2.7.3.

```
import numpy as np
m = np.arange(1, 10).reshape(3, 3)
print(m) # [[1 2 3] [4 5 6] [7 8 9]]
print(m[[0, 2], [1, 2]]) # [2 9] (στοιχεία από [0,1] και [2,2])
```

Κ. 2.7.3 - Λίστα δεικτών - Fancy indexing σε πίνακες δύο διαστάσεων (2D)

Ας δούμε κάποιες περαιτέρω προχωρημένες εφαρμογές για ευρετηρίαση και τεμαχισμό σε ένα συνδυασμό χρήσης τους. Στον κώδικα Κ. 2.7.4 λαμβάνει τις 2 πρώτες γραμμές και τις στήλες 0 και 2, από τον πίνακα m.

```
import numpy as np
m = np.arange(1, 10).reshape(3, 3) #Έξοδος: [[1 3], [4 6]]
print(m[0:2, [0, 2]])
```

Κ. 2.7.4 - Συνδυασμός ευρετηρίασης με τεμαχισμό

Ένας πιο ευέλικτος τρόπος χρήσης είναι με συνδυασμό ευρετηρίασης και τεμαχισμού με χρήση της `np.take`. Ο κώδικας Κ. 2.7.5 μας δείχνει τον τρόπο εφαρμογής.

```
import numpy as np
m = np.arange(1, 10).reshape(3, 3)
print(np.take(m, [0, 3, 5])) # [1 4 6]
```
Κ. 2.7.5 - Συνδυασμός ευρετηρίασης με τεμαχισμό με την np.take()

Εάν θέλουμε να πάρουμε στοιχεία από ένα πίνακα, μπορούμε να εφαρμόσουμε κάποιο από τα παραπάνω είτε συνεχόμενα στοιχεία (slicing), μεμονωμένα indices (fancy indexing), λογικές συνθήκες (masking). Για να λάβουμε συνδυασμό αυτών των επιλογών μπορούμε να κάνουμε χρήση του np.ix_ όπως θα δούμε αμέσως στο επόμενο παράδειγμα. Με χρήση του np.ix_ γίνεται συνδυαστική επιλογή από πολλούς δείκτες γραμμών/στηλών, όπως στο παράδειγμα του κώδικα Κ. 2.7.6. Έτσι μπορούμε να επιλέξουμε ό,τι θέλουμε από έναν πίνακα: συνεχόμενα στοιχεία (slicing), μεμονωμένα indices (fancy indexing), λογικές συνθήκες (masking), ή πλέγμα επιλογών (np.ix_). Ένας πίνακας 3x3 με τιμές 1–9, όπου με fancy indexing (m[[0,2],[1,2]]) επιλέξαμε τα στοιχεία της γραμμής 0 και 2 και των στηλών 1 και 2.

```
import numpy as np
m = np.arange(1, 10).reshape(3, 3)
rows = [0, 2]
cols = [1, 2]
print(m[np.ix_(rows, cols)])
      # [[2 3]
      # [8 9]]
```
Κ. 2.7.6 - Χρήση του np.ix_ για συνδυασμό επιλογών

2.8. Ταξινόμηση πινάκων

Ας δούμε τώρα τη δυνατότητα της βιβλιοθήκης Numpy για ταξινόμηση των τιμών πινάκων. Με χρήση της `np.sort` λαμβάνουμε νέο ταξινομημένο πίνακα χωρίς να γίνει τροποποίηση στον αρχικό πίνακα (import numpy as np

a = np.array([3, 1, 4, 1, 5, 6])

```
print(np.sort(a)) # [1 1 3 4 5 6]
print(a) # [3 1 4 1 5 6] (δεν άλλαξε)
```
Κ. 2.8.1 - Ταξινόμηση δεδομένων πίνακα, αυτό είναι εξαιρετικά σημαντικό γιατί ταξινομούνται τα δεδομένα αλλά δεν χάνουμε τα αρχικά ούτε χώρο στη μνήμη ώστε να αποθηκεύσουμε τα νέα δεδομένα.

```
import numpy as np
a = np.array([3, 1, 4, 1, 5, 6])
print(np.sort(a)) # [1 1 3 4 5 6]
print(a) # [3 1 4 1 5 6] (δεν άλλαξε)
```
Κ. 2.8.1 - Ταξινόμηση δεδομένων πίνακα

Η ταξινόμηση μπορεί να λειτουργήσει και κατά άξονα, όπως στο παράδειγμα του κώδικα Κ. 2.8.2 για ταξινόμηση κατά στήλες και του Κ. 2.8.3 για ταξινόμηση κατά γραμμές.

```
import numpy as np
a = np.array([[3, 1, 2], [9, 5, 4]])
print(np.sort(a, axis=0)) # Ταξινόμηση κατά στήλες
# [[3 1 2]
# [9 5 4]]
```
Κ. 2.8.2 - Ταξινόμηση κατά στήλες

```
import numpy as np
a = np.array([[3, 1, 2], [9, 5, 4]])
print(np.sort(m, axis=1)) # Ταξινόμηση κατά γραμμές
# [[1 2 3]
# [4 5 9]]
```
Κ. 2.8.3 Ταξινόμηση κατά γραμμές

Ένα άλλος τρόπος ταξινόμησης τιμών με χρήση της Numpy είναι η ταξινόμηση πινάκων με τη συνάρτηση `np.argsort()`. Αυτή η συνάρτηση επιστρέφει τους δείκτες ταξινόμησης και όχι τις τιμές. Στον κώδικα Κ. 2.8.4 η πρώτη χρήση της συνάρτησης `np.argsort(a)` επιστρέφει τους δείκτες που θα ταξινομούν τον πίνακα a σε αύξουσα σειρά. Για τον πίνακα a = [3, 1, 4, 1, 5, 6], η έξοδος είναι [1 3 0 2 4 5] (θέσεις με αύξουσα σειρά). Δηλαδή το μικρότερο στοιχείο (1) βρίσκεται στο δείκτη 1 και 3, το επόμενο μικρότερο στοιχείο (3) βρίσκεται στο δείκτη 0, ακολουθεί το (4) στο δείκτη 2, το (5) στο δείκτη 4 και το (6) στο δείκτη 5. Στη δεύτερη χρήση του `np.argsort()`, εντολή `print(a[idx])` , ταξινομεί τον αρχικό πίνακα a σύμφωνα με τους δείκτες που επιστρέφονται από την εντολή np.argsort(). Η έξοδος είναι [1 1 3 4 5 6]. Συνοψίζοντας, στον κώδικα Κ. 2.8.4 στην πρώτη εντολή print δίνει τους δείκτες που θα ταξινομήσουν τον πίνακα. Ενώ στη δεύτερη εντολή print χρησιμοποιεί αυτούς τους δείκτες για να εμφανίσει τον ταξινομημένο πίνακα. Αυτός είναι ο λόγος που έχει και διαφορετική έξοδο.

```
import numpy as np
a = np.array([3, 1, 4, 1, 5, 6])
print(np.argsort(a))
# [1 3 0 2 4 5] (θέσεις με αύξουσα σειρά)

# Χρήση για ταξινόμηση "χειροκίνητα"
idx = np.argsort(a)
print(a[idx]) # [1 1 3 4 5 6]
```
Κ. 2.8.4 - Χρήση της συνάρτησης argsort()

Για να ταξινομήσουμε ένα πίνακα με φθίνουσα σειρά με χρήση της `np.sort` ή της `np.argsort` θα πρέπει να χρησιμοποιήσουμε ορίσματα της μορφής ::-1 ή -a, όπως φαίνεται στον κώδικα

```
import numpy as np
a = np.array([3, 1, 4, 1, 5, 6])
print(np.sort(a)[::-1]) # [6 5 4 3 1 1]
print(a[np.argsort(-a)]) # [6 5 4 3 1 1]
```
Κ. 2.8.5 - Φθίνουσα ταξινόμηση

Υπάρχει και δυνατότητα επί τόπου ταξινόμησης (In place) με την συνάρτηση `a.sort()`. Αυτή η μέθοδος ταξινομεί τον πίνακα επί τόπου, που σημαίνει ότι τροποποιεί τον αρχικό πίνακα a χωρίς να δημιουργεί νέο πίνακα. Μετά από αυτήν τη λειτουργία, ο πίνακας a περιέχει τις ταξινομημένες τιμές.

Στο κώδικα Κ. 2.8.6 το τελικό αποτέλεσμα της print(a) θα εμφανίσει τον ταξινομημένο πίνακα ως [1, 1, 3, 4, 5, 6].

```
import numpy as np
a = np.array([3, 1, 4, 1, 5, 6])
a.sort()
print(a)    # [1 1 3 4 5 6]
```

Κ. 2.8.6 - In place ταξινόμηση

Στον κώδικα Κ. 2.8.7 βλέπουμε την ταξινόμηση πινάκων για σύνθετες χρήσεις. Πιο συγκεκριμένα το `m[:, 1]` είναι για να επιλεγούν όλες οι γραμμές της δεύτερης στήλης του πίνακα m. Οι τιμές στη δεύτερη στήλη είναι [7, 9, 5]. Η συνάρτηση `np.argsort(m[:, 1])` επιστρέφει τους δείκτες που θα ταξινομούσαν τη δεύτερη στήλη σε αύξουσα σειρά. Η έξοδος είναι [0, 2, 1], υποδεικνύοντας τη σειρά των δεικτών που θα ταξινομήσουν τη δεύτερη στήλη. Η εντολή `print(m[idx])` χρησιμοποιεί τους δείκτες που επιστρέφονται από το `np.argsort(m[:, 1])` ([0, 2, 1]) για να αναδιατάξουν τις γραμμές του m. Η ταξινομημένη έξοδος με βάση τη δεύτερη στήλη θα είναι [[2 5] [3 7] [1 9]]. Δηλαδή σε αυτό το παράδειγμα η τελική έξοδος δείχνει τις γραμμές του αρχικού πίνακα m, ταξινομημένες με βάση τις τιμές στη δεύτερη στήλη.

```
import numpy as np
m = np.array([[3, 7], [1, 9], [2, 5]])
# Ταξινόμηση γραμμών με βάση τη στήλη 1
idx = np.argsort(m[:, 1])
print(m[idx])
      #Έξοδος [[2 5] [3 7] [1 9]]
```

Κ. 2.8.7 - Ταξινόμηση πινάκων - Σύνθετες χρήσεις

2.9. Επίλυση συστημάτων γραμμικών εξισώσεων

Η γραμμική άλγεβρα είναι ένας θεμελιώδης τομέας των μαθηματικών που εστιάζει σε διανύσματα, διανυσματικούς χώρους και γραμμικούς μετασχηματισμούς μεταξύ αυτών των χώρων. Η κατανόηση των βασικών εννοιών της είναι κρίσιμη για διάφορες εφαρμογές στην επιστήμη, τη μηχανική, τα γραφικά υπολογιστών, τη μηχανική μάθηση και άλλα. Αυτή η ενότητα πραγματεύεται τρία βασικά στοιχεία:

- Βαθμωτά: Μοναδικές αριθμητικές τιμές, που συχνά απεικονίζονται με πεζά γράμματα (π.χ., (x, y, z)).
- Πίνακες: Ορθογώνιοι πίνακες αριθμών διατεταγμένοι σε γραμμές και στήλες, που συνήθως αναπαρίστανται με έντονα κεφαλαία γράμματα (π.χ., (A, B)).
- Διανύσματα: Διατεταγμένοι πίνακες αριθμών που αναπαριστούν ποσότητες σε μια συγκεκριμένη κατεύθυνση, συνήθως συμβολίζονται με έντονα πεζά γράμματα (π.χ., (v, w)).

2.9.1. Πίνακες

Στη βιβλιοθήκη NumPy, αυτά τα στοιχεία αντιμετωπίζονται αποτελεσματικά χρησιμοποιώντας πίνακες, καθιστώντας τις πράξεις γραμμικής άλγεβρας τόσο απλές όσο και υπολογιστικά γρήγορες. Για το ακόλουθο σύστημα γραμμικών εξισώσεων η βιβλιοθήκη Numpy μπορεί να βοηθήσει στην επίλυση του χρησιμοποιώντας τη συνάρτηση `np.linalg.solve()`.

$$2x + 3y = 39$$

$$5x - y = -1$$

Αυτή η συνάρτηση υπολογίζει την ακριβή λύση για γραμμικές εξισώσεις της μορφής (Ax = B), όπου (A) είναι ένας πίνακας συντελεστών και (B) είναι ένα διάνυσμα σταθερών. Η χρήση της `np.linalg.solve()` γίνεται ως εξής `np.linalg.solve(A, B)`. Όπου:

- A: Ένας τετραγωνικός πίνακας (δισδιάστατος πίνακας) που περιέχει τους συντελεστές των γραμμικών εξισώσεων.
- B: Ένας μονοδιάστατος πίνακας (ή διάνυσμα στήλης) που αναπαριστά τις σταθερές στις εξισώσεις.

Η συνάρτηση επιστρέφει έναν πίνακα που περιέχει τις λύσεις για τις μεταβλητές ή αλλιώς τη λύση του συστήματος γραμμικών εξισώσεων, στο παράδειγμα του κώδικα Κ. 2.9.1 η λύση είναι `[2.11764706 11.58823529]`

```
import numpy as np
A = np.array([[2, 3], [5, -1]])
B = np.array([39, -1])
solution = np.linalg.solve(A, B)
print(solution) #[ 2.11764706 11.58823529]
```

Κ. 2.9.1 - Επίλυση γραμμικής εξίσωσης

2.9.2. Διανύσματα

Ένα διάνυσμα είναι μια διάταξη αριθμών που αντιπροσωπεύει ποσότητα με μέτρο και κατεύθυνση. Παράδειγμα: v=(2,3), w=(1,−4), v=(2,3), w=(1,−4). Συνήθως συμβολίζεται γράφοντας με έντονα γράμματα ή με βελάκι (π.χ. v, w ή $\vec{v}$ $\vec{w}$). Στη βιβλιοθήκη NumPy η υλοποίηση των διανυσμάτων γίνεται με απλούς μονοδιάστατους πίνακες (Κ. 2.9.2).

```
import numpy as np
v = np.array([2, 3])
w = np.array([1, -4])
```

Κ. 2.9.2 - Μονοδιάστατοι πίνακες Numpy

Στον κώδικα Κ. 2.9.3 εμφανίζονται μια σειρά από παραδείγματα πράξεων όπως η πρόσθεση, το Εσωτερικό γινόμενο (dot product), η Ευκλείδεια νόρμα (μήκος διανύσματος), και η κατεύθυνση (μονάδα διάνυσμα).

```python
import numpy as np
v = np.array([2, 3])
w = np.array([1, -4])
print(v + w) # Πρόσθεση: [ 3 -1]
print(np.dot(v, w)) # Εσωτερικό γινόμενο (dot product): 2*1 + 3*(-4) = -10
print(np.linalg.norm(v)) #Ευκλείδεια νόρμα (μήκος διανύσματος): sqrt(2^2 +
3^2) = sqrt(13) = 3.605551275463989
unit_v = v / np.linalg.norm(v)
print(unit_v) #Κατεύθυνση (μονάδα διάνυσμα): [0.5547002  0.83205029]
```

Κ. 2.9.3 - Πράξεις διανυσμάτων

2.10. Ερωτήσεις αυτοαξιολόγησης

1. **Ποια είναι η κύρια χρήση της βιβλιοθήκης NumPy;**

   Α) Δημιουργία ιστοσελίδων

   Β) Ανάλυση δεδομένων

   Γ) Σχεδίαση γραφικών

   Δ) Διαχείριση βάσεων δεδομένων

2. **Ποια δομή δεδομένων παρέχει η NumPy;**

   Α) DataFrame

   Β) List

   Γ) ndarray

   Δ) Dictionary

3. **Ποιες διαστάσεις πινάκων μπορεί να διαχειριστεί η βιβλιοθήκη NumPy;**

   Α) Μονοδιάστατοι πίνακες

   Β) Δισδιάστατοι πίνακες

   Γ) Τρισδιάστατοι πίνακες

   Δ) Όλες οι παραπάνω

4. **Ποια συνάρτηση δημιουργεί έναν πίνακα γεμάτο με μηδενικά;**

   Α) np.ones()

   Β) np.zeros()

   Γ) np.empty()

   Δ) np.full()

5. **Ποια συνάρτηση χρησιμοποιείται για να υπολογίσει τον μέσο όρο ενός πίνακα;**

   Α) np.sum()

   Β) np.mean()

   Γ) np.average()

   Δ) np.median()

6. **Πώς δημιουργείται ένας δισδιάστατος πίνακας με τη NumPy;**

Α) np.array([1, 2, 3])

Β) np.array([[1, 2], [3, 4]])

Γ) np.array(5)

Δ) np.array([[1, 2, 3]])

7. Ποιες είναι οι βασικές ιδιότητες ενός πίνακα ndarray;

Α) ndim, shape, size

Β) length, width, height

Γ) count, total, average

Δ) all of the above

8. Ποια συνάρτηση χρησιμοποιείται για την εκτέλεση της ανίχνευσης NaN σε πίνακες;

Α) np.isnan()

Β) np.isnull()

Γ) np.nan()

Δ) np.isnanvalue()

9. Ποιες είναι οι βασικές λειτουργίες που προσφέρει η NumPy για αριθμητικούς υπολογισμούς;

Α) Συναρτήσεις συνάθροισης

Β) Λειτουργίες μετάδοσης

Γ) Αλγεβρικές πράξεις

Δ) Όλες οι παραπάνω

10. Πώς δημιουργείται ένα διάστημα αριθμών με τη NumPy;

Α) np.linspace()

Β) np.arange()

Γ) np.sequence()

Δ) np.interval()

11. Ποια είναι η διαφορά μεταξύ np.concatenate() και np.vstack();

Α) Δεν υπάρχει διαφορά

Β) Η np.vstack() συνενώνει μόνο κατακόρυφα

Γ) Η np.concatenate() δεν επιτρέπει συνένωση

Δ) Η np.vstack() δεν δέχεται πίνακες

12. Ποιες είναι οι συνθήκες για την σωστή χρήση της μετάδοσης (broadcasting);

Α) Τα σχήματα των πινάκων πρέπει να είναι τα ίδια

Β) Ένας πίνακας πρέπει να έχει μέγεθος 1

Γ) Τα σχήματα πρέπει να είναι συμβατά

Δ) Όλες οι παραπάνω

13. Ποια συνάρτηση χρησιμοποιείται για την αναδιάρθρωση ενός πίνακα;

Α) np.resize()

Β) np.reshape()

Γ) np.restructure()

Δ) np.reformat()

14. Ποια είναι η χρήση της np.dot() στην NumPy;

Α) Για πρόσθεση πινάκων

Β) Για αλγεβρικό πολλαπλασιασμό πινάκων

Γ) Για διαίρεση πινάκων

Δ) Για αύξηση πινάκων

15. Ποιες είναι οι βασικές προκλήσεις που μπορεί να προκύψουν κατά τη χρήση της NumPy;

Α) Ασύμβατες διαστάσεις

Β) Σφάλματα κατά την εισαγωγή δεδομένων

Γ) Υψηλή κατανάλωση μνήμης

Δ) Όλες οι παραπάνω

16. Ποια συνάρτηση χρησιμοποιείται για την εύρεση μοναδικών τιμών σε έναν πίνακα;

Α) np.unique()

Β) np.distinct()

Γ) np.unique_values()

Δ) np.different()

17. **Πώς μπορεί να γίνει η επιλογή στοιχείων από έναν πίνακα με βάση μια λογική μάσκα;**

    Α) Χρησιμοποιώντας το np.select()

    Β) Με την αναφορά στον πίνακα με τις λογικές τιμές

    Γ) Με τη χρήση της np.where()

    Δ) Κανένα από τα παραπάνω

18. **Ποια είναι η χρήση της συνάρτησης np.vstack() στη NumPy;**

    Α) Δημιουργία πίνακα γεμάτου μηδενικά

    Β) Συνένωση πινάκων κατά στήλη

    Γ) Συνένωση πινάκων κατά γραμμή

    Δ) Αλλαγή διαστάσεων πίνακα

19. **Πώς μπορεί να γίνει η αναδιάρθρωση ενός πίνακα σε 2 διαστάσεις;**

    Α) Χρησιμοποιώντας την np.reshape()

    Β) Με την np.resize()

    Γ) Με την np.array()

    Δ) Με την np.transpose()

20. **Ποια είναι η λειτουργία της np.random.rand() στη NumPy;**

    Α) Δημιουργεί έναν πίνακα γεμάτο μηδενικά

    Β) Δημιουργεί έναν πίνακα με τυχαίους αριθμούς από το 0 έως το 1

    Γ) Δημιουργεί έναν πίνακα με τυχαίους ακέραιους αριθμούς

    Δ) Δημιουργεί έναν πίνακα με σταθερές τιμές

2.10.1. Απαντήσεις στις ερωτήσεις αυτοαξιολόγησης

1. Β) Ανάλυση δεδομένων

Επεξήγηση: Η NumPy είναι σχεδιασμένη για εργασίες ανάλυσης και επεξεργασίας δεδομένων, προσφέροντας εργαλεία για αριθμητικούς υπολογισμούς.

2. Γ) ndarray

Επεξήγηση: Η NumPy χρησιμοποιεί τη δομή δεδομένων ndarray για την αποθήκευση και την επεξεργασία αριθμητικών δεδομένων.

3. Δ) Όλες οι παραπάνω

Επεξήγηση: Η NumPy υποστηρίζει πίνακες πολλών διαστάσεων, συμπεριλαμβανομένων μονοδιάστατων, δισδιάστατων και τρισδιάστατων.

4. Β) np.zeros()

Επεξήγηση: Η συνάρτηση np.zeros() δημιουργεί έναν πίνακα γεμάτο με μηδενικά.

5. Β) np.mean()

Επεξήγηση: Η np.mean() υπολογίζει τον μέσο όρο των στοιχείων ενός πίνακα.

6. Β) np.array([[1, 2], [3, 4]])

Επεξήγηση: Η np.array() μπορεί να χρησιμοποιηθεί με μια λίστα λιστών για να δημιουργήσει δισδιάστατους πίνακες.

7. Α) ndim, shape, size

Επεξήγηση: Οι ιδιότητες ndim, shape και size είναι σημαντικές για την περιγραφή των πινάκων ndarray.

8. Α) np.isnan()

Επεξήγηση: Η np.isnan() ελέγχει αν τα στοιχεία ενός πίνακα είναι NaN.

9. Δ) Όλες οι παραπάνω

Επεξήγηση: Η NumPy προσφέρει πλήθος λειτουργιών για αριθμητικούς υπολογισμούς.

10. Β) np.arange()

Επεξήγηση: Η np.arange() δημιουργεί πίνακες με αριθμούς σε κανονική απόσταση.

11. Β) Η np.vstack() συνενώνει μόνο κατακόρυφα

Επεξήγηση: Η np.vstack() συνενώνει πίνακες κατά μήκος του άξονα 0.

12. Γ) Τα σχήματα πρέπει να είναι συμβατά

Επεξήγηση: Η μετάδοση λειτουργεί μόνο εάν τα σχήματα των πινάκων είναι συμβατά.

13. Β) np.reshape()

Επεξήγηση: Η np.reshape() αλλάζει τις διαστάσεις ενός πίνακα.

14. Β) Για αλγεβρικό πολλαπλασιασμό πινάκων

Επεξήγηση: Η np.dot() χρησιμοποιείται για τον αλγεβρικό πολλαπλασιασμό πινάκων.

15. Δ) Όλες οι παραπάνω

Επεξήγηση: Όλες αυτές οι προκλήσεις είναι συχνές κατά τη χρήση της NumPy.

16. Α) np.unique()

Επεξήγηση: Η np.unique() επιστρέφει έναν πίνακα που περιέχει μόνο τις μοναδικές τιμές από τον αρχικό πίνακα.

17. Β) Με την αναφορά στον πίνακα με τις λογικές τιμές

Επεξήγηση: Η επιλογή στοιχείων με λογική μάσκα γίνεται με την αναφορά στον πίνακα με τις λογικές τιμές, π.χ. array[mask]

18. Γ) Συνένωση πινάκων κατά γραμμή

Επεξήγηση: Η np.vstack() συνενώνει πίνακες κάθετα (κατά γραμμή), δημιουργώντας έναν νέο πίνακα που περιλαμβάνει τις γραμμές των συνενωμένων πινάκων.

19. Α) Χρησιμοποιώντας την np.reshape()

Επεξήγηση: Η np.reshape() χρησιμοποιείται για να αλλάξει τις διαστάσεις ενός πίνακα, επιτρέποντας την μετατροπή του σε 2 διαστάσεις.

20. Β) Δημιουργεί έναν πίνακα με τυχαίους αριθμούς από το 0 έως το 1

Επεξήγηση: Η np.random.rand() δημιουργεί έναν πίνακα γεμάτο με τυχαίους αριθμούς σε διάστημα 0, 1).

ΚΕΦΑΛΑΙΟ 3: Η ΒΙΒΛΙΟΘΗΚΗ PANDAS

Στόχοι του κεφαλαίου:

- Να γνωρίζουν που αποσκοπεί η βιβλιοθήκη Pandas και γιατί είναι δημοφιλής.
- Να κατανοούν την έννοια των Series και των DataFrames.
- Να γνωρίζουν τους τρόπους που μπορεί να γίνει δημιουργία ενός DataFrame.
- Να γνωρίζουν τους τρόπους με τους οποίους μπορεί να επιτευχθεί πρόσβαση σε γραμμές και στήλες ενός DataFrame.
- Να γνωρίζουν διάφορους τρόπους δεικτοδότησης και επιλογής δεδομένων από DataFrames.
- Να είναι σε θέση να υπολογίζουν συγκεντρωτικά στατιστικά σε DataFrames.
- Να γνωρίζουν πως να χειρίζονται δεδομένα που λείπουν.
- Να γνωρίζουν να μετατρέπουν δεδομένα που βρίσκονται σε DataFrames από έναν τύπο σε έναν άλλο.
- Να επεξεργάζονται δεδομένα DataFrames που είναι σε μορφή κειμένου.
- Να γνωρίζουν που αποσκοπεί η βιβλιοθήκη Pandas και γιατί είναι δημοφιλής.
- Να κατανοούν την έννοια των Series και των DataFrames.
- Να γνωρίζουν τους τρόπους που μπορεί να γίνει δημιουργία ενός DataFrame.
- Να γνωρίζουν τους τρόπους με τους οποίους μπορεί να επιτευχθεί πρόσβαση σε γραμμές και στήλες ενός DataFrame.
- Να γνωρίζουν διάφορους τρόπους δεικτοδότησης και επιλογής δεδομένων από DataFrames.
- Να είναι σε θέση να υπολογίζουν συγκεντρωτικά στατιστικά σε DataFrames.
- Να γνωρίζουν πως να χειρίζονται δεδομένα που λείπουν.
- Να γνωρίζουν να μετατρέπουν δεδομένα που βρίσκονται σε DataFrames από έναν τύπο σε έναν άλλο.
- Να επεξεργάζονται δεδομένα DataFrames που είναι σε μορφή κειμένου.

3.1. Εισαγωγή στη βιβλιοθήκη Pandas (pandas library)

Η **pandas** είναι μια ανοιχτού κώδικα (open-source) βιβλιοθήκη της γλώσσας προγραμματισμού Python, σχεδιασμένη ειδικά για την επεξεργασία και ανάλυση δομημένων δεδομένων (structured data), όπως πίνακες δεδομένων με γραμμές και στήλες. Αποτελεί σήμερα ένα από τα βασικότερα εργαλεία της επιστήμης δεδομένων (data science), της μηχανικής μάθησης (machine learning) και της ανάλυσης δεδομένων (data analysis), καθώς επιτρέπει την αποτελεσματική διαχείριση, μετασχηματισμό και διερεύνηση μεγάλων συνόλων δεδομένων.

Η βιβλιοθήκη αναπτύχθηκε αρχικά από τον Wes McKinney γύρω στο 2008, με στόχο να προσφέρει στην Python έναν ισχυρό και ευέλικτο τρόπο χειρισμού ταμπελωμένων δεδομένων (labeled data),

αντίστοιχο με αυτόν που υπήρχε ήδη σε άλλες πλατφόρμες (όπως το R). Σήμερα συντηρείται από μια ευρεία κοινότητα προγραμματιστών και ερευνητών και διανέμεται επίσημα μέσω του ιστότοπου του έργου pandas και του Python Package Index (PyPI).

Κεντρικός στόχος της pandas είναι να προσφέρει υψηλού επιπέδου δομές δεδομένων (high-level data structures) και εργαλεία για την εργασία με **ετικετοποιημένα δεδομένα** (labeled data) και **ετερογενή δεδομένα** (heterogeneous data). Στο κέντρο αυτής της προσέγγισης βρίσκονται δύο βασικές δομές: η **Series** (pandas.Series) και το **DataFrame** (pandas.DataFrame), οι οποίες αποτελούν τη βάση σχεδόν κάθε εργασίας με pandas.

Η pandas είναι στενά συνδεδεμένη με τη βιβλιοθήκη **NumPy**, στην οποία βασίζεται για την αναπαράσταση αριθμητικών πινάκων (ndarray) και για αποδοτικές πράξεις σε επίπεδο πινάκων. Ωστόσο, επεκτείνει τις δυνατότητες της NumPy προσθέτοντας έννοιες όπως ευέλικτο ευρετήριο (flexible index), ετικέτες γραμμών και στηλών (row and column labels), χειρισμό ελλιπών τιμών (missing values) και σύνθετες πράξεις ομαδοποίησης (groupby operations).

Μερικές από τις βασικές δυνατότητες (key features) της βιβλιοθήκης pandas είναι:

- Αποδοτική αναπαράσταση πινάκων δεδομένων (tabular data) με γραμμές και στήλες, μέσω DataFrame.
- Ευέλικτη δεικτοδότηση (flexible indexing) με βάση ετικέτες (labels) ή ακέραιες θέσεις (integer positions).
- Ενσωματωμένος χειρισμός ελλιπών τιμών (missing data), με ειδικούς δείκτες όπως NaN.
- Ισχυρές λειτουργίες επιλογής, φιλτραρίσματος, ταξινόμησης και αναδιάταξης δεδομένων (selection, filtering, sorting, reshaping).
- Συναθροίσεις και ομαδοποιήσεις δεδομένων (groupby, aggregations), χρήσιμες για στατιστική ανάλυση.
- Εργαλεία για χρονοσειρές (time series tools), όπως ημερομηνιακά ευρετήρια (DatetimeIndex), resampling και rolling υπολογισμούς.
- Εύκολη εισαγωγή και εξαγωγή δεδομένων (input/output) από/σε μορφές όπως CSV, Excel, SQL και JSON.

Τυπικές εφαρμογές της pandas συναντώνται στην προκαταρκτική ανάλυση δεδομένων (exploratory data analysis, EDA), στον καθαρισμό και προεπεξεργασία δεδομένων (data cleaning, preprocessing) και στην προετοιμασία συνόλων δεδομένων για μοντέλα μηχανικής μάθησης. Σε αυτό το πλαίσιο, η pandas λειτουργεί συχνά ως “γέφυρα” μεταξύ των αρχικών δεδομένων (raw data) και των πιο εξειδικευμένων βιβλιοθηκών ανάλυσης ή μοντελοποίησης, όπως scikit-learn, statsmodels ή βιβλιοθήκες οπτικοποίησης (visualization) τύπου Matplotlib και Seaborn.

Στις επόμενες ενότητες θα μελετηθούν αναλυτικά οι βασικές δομές δεδομένων της pandas, δηλαδή οι **σειρές** (**Series)** και τα **πλαίσια δεδομένων** (**DataFrames)**, καθώς και οι τρόποι δημιουργίας,

πρόσβασης και μετασχηματισμού τους. Με αυτόν τον τρόπο, θα τεθεί ένα σταθερό θεωρητικό και πρακτικό υπόβαθρο πάνω στο οποίο θα χτιστούν οι πιο προχωρημένες έννοιες, όπως ομαδοποιήσεις, συγχωνεύσεις, χρονοσειρές και δεδομένα με περισσότερες από δύο διαστάσεις.

3.2. Σειρές (Series) και Πλαίσια Δεδομένων (DataFrames)

Η βιβλιοθήκη pandas βασίζεται σε δύο θεμελιώδεις δομές δεδομένων: τις **Σειρές** (pandas.Series) και τα **πλαίσια δεδομένων** (pandas.DataFrame), οι οποίες παρέχουν έναν τυποποιημένο τρόπο αναπαράστασης και επεξεργασίας δεδομένων με ετικέτες (labeled data) στην Python. Η κατανόηση αυτών των δύο δομών αποτελεί προϋπόθεση για κάθε πιο προχωρημένη εργασία με pandas, από την απλή επιλογή δεδομένων μέχρι σύνθετες ομαδοποιήσεις και χρονοσειρές.

Μια **Σειρά** (**Series)** μπορεί να θεωρηθεί ως ένας μονοδιάστατος, πίνακας με ετικέτες (one-dimensional labeled array) που μπορεί να περιέχει στοιχεία οποιουδήποτε τύπου δεδομένων της Python ή της NumPy, όπως ακέραιους, πραγματικούς, λεκτικά (strings) ή ακόμη και σύνθετα αντικείμενα. Κάθε στοιχείο της Σειράς συνοδεύεται από μια ετικέτα ευρετηρίου (index label), η οποία επιτρέπει προσπέλαση όχι μόνο με ακέραιες θέσεις αλλά και με συμβολικές ετικέτες, διευκολύνοντας τη σαφή και αναγνώσιμη αναφορά σε δεδομένα.

Αντίστοιχα, ένα **Πλαίσιο Δεδομένων (DataFrame)** αποτελεί μια δισδιάστατη δομή δεδομένων (two-dimensional labeled data structure), οργανωμένη σε γραμμές και στήλες, όπου κάθε στήλη είναι ουσιαστικά μια Series με κοινό ευρετήριο γραμμών (shared index). Το DataFrame μπορεί να θεωρηθεί ως πίνακας (table) ή ως συλλογή από Series, και είναι ιδιαίτερα κατάλληλο για την αναπαράσταση πινάκων δεδομένων τύπου υπολογιστικού φύλλου (spreadsheet-like data) ή πινάκων βάσεων δεδομένων.

Η σχέση μεταξύ Series και DataFrame είναι στενή: πολλές πράξεις που εφαρμόζονται σε μια Series έχουν ανάλογη ερμηνεία σε επίπεδο στήλης DataFrame, ενώ η πρόσβαση σε μια στήλη DataFrame συνήθως επιστρέφει ένα αντικείμενο τύπου Series. Αυτός ο σχεδιασμός επιτρέπει ένα ομοιογενές μοντέλο χειρισμού δεδομένων, στο οποίο οι λειτουργίες σε επίπεδο στήλης, γραμμής ή ολόκληρου πίνακα ακολουθούν παρόμοια λογική, με συνέπεια στη σύνταξη και στη σημασιολογία.

Σε πρακτικό επίπεδο, μια Series και ένα DataFrame μπορούν να δημιουργηθούν από πληθώρα πηγών, όπως λίστες (lists), λεξικά (dictionaries), πίνακες NumPy (NumPy arrays), αρχεία CSV ή πίνακες SQL, προσφέροντας μεγάλη ευελιξία στη φόρτωση και οργάνωση δεδομένων. Στις επόμενες ενότητες θα εξεταστούν αναλυτικά οι βασικοί τρόποι δημιουργίας **DataFrame** από πίνακες NumPy, καθώς και οι μέθοδοι δημιουργίας DataFrame “κατά στήλες” και “κατά γραμμές”, ώστε να γίνει σαφές πώς οι θεωρητικές έννοιες Series και DataFrame υλοποιούνται στην πράξη.

3.3. Σειρές (Series)

Αποτελούν μια μονοδιάστατη δομή τύπου πίνακα με ομοιογενή δεδομένα, που συνοδεύεται από δείκτες (indexes), επιτρέποντας την εύκολη διαχείριση και επεξεργασία των δεδομένων.

Επίσημη σελίδα τεκμηρίωσης: https://pandas.pydata.org/docs/reference/api/pandas.Series.html

Τον κώδικας της παραγράφου μπορείτε να τον βρείτε εδώ:
https://colab.research.google.com/drive/1D8FS94GCTgN_FGVu79so6MACVuqxloPp?usp=sharing

- **Μονοδιάστατος, ετικεταρισμένος πίνακας** (1D labeled array)
- Ισοδύναμο με μια στήλη σε υπολογιστικό φύλλο ή πίνακα δεδομένων
- Κάθε στοιχείο έχει **ετικέτα (index)** και **τιμή (value)**

Μια Σειρά αποτελείται από δύο κύρια μέρη:

1. Δεδομένα (Values)

- Οι πραγματικές τιμές που αποθηκεύονται
- Μπορεί να είναι numbers, strings, booleans, κλπ.
- Στοιχίζονται σε μονοδιάστατο πίνακα

2. Ευρετήριο (Index)

- Ετικέτες για κάθε τιμή
- Προσδιορίζει μοναδικά κάθε στοιχείο

Μπορεί να είναι numbers, strings, dates, κλπ.

Βασικά Χαρακτηριστικά

Χαρακτηριστικό	Περιγραφή	Παράδειγμα
Μονοδιάστατη	Μόνο μία διάσταση	(5,)
Ετικέτες	Προσβάσιμη με ετικέτες ή θέσεις	s['a'] ή s[0]
Ένας τύπος	Όλα τα στοιχεία ίδιου τύπου	int64, float64, object
Ευέλικτη	Δέχεται διάφορους τύπους δεδομένων	numbers, strings, dates
Ισχυρή	Πλούσιο σύνολο μεθόδων	200+ μέθοδοι

Πίνακας 3.3.1: Βασικά Χαρακτηριστικά των Σειρών

Σύγκριση με Άλλες Δομές

Δομή	Διαστάσεις	Ετικέτες	Τύποι Δεδομένων
Python List	1D	✗ Όχι	✓ Μικτός
NumPy Array	N-D	✗ Όχι	✗ Ένας τύπος
Python Dict	1D	✓ Κλειδιά	✓ Μικτός
Pandas Series	1D	✓ Ευρετήριο	✗ Ένας τύπος

Πίνακας 3.3.2: Σύγκριση των Σειρών με άλλες Δομές

Βασικά Πλεονεκτήματα

Σε σχέση με τις Λίστες (Lists):

- Ετικέτες για εύκολη πρόσβαση
- Αυτόματες στατιστικές συναρτήσεις
- Εύκολο φιλτράρισμα και ταξινόμηση
- Καλύτερη απόδοση με μεγάλα δεδομένα

Σε σχέση με τους πίνακες (NumPy Arrays):

- Ευρετήριο με ετικέτες
- Εύκολη χειρισμός NaN τιμών
- Πλούσιο σύνολο μεθόδων
- Καλύτερη οπτικοποίηση

Σε σχέση με τα λεξικά (Dictionaries):

- Διατήρηση σειράς στοιχείων
- Μαθηματικές πράξεις
- Boolean indexing
- Ομαδοποίηση και συναθροίσεις

3.3.1. Δημιουργία & Βασικές Πληροφορίες

Μέθοδος	Περιγραφή	Παράδειγμα

pd.Series(data)	Δημιουργία Series	s = pd.Series([1, 2, 3])
.shape	Διάσταση (μονάδα)	s.shape → (3,)
.size	Αριθμός στοιχείων	s.size → 3
.index	Ευρετήριο	s.index → RangeIndex
.values	Τιμές ως numpy array	s.values → array([1, 2, 3])
.dtype	Τύπος δεδομένων	s.dtype → int64
.name	Όνομα Series	s.name = 'Τιμές'
.empty	Έλεγχος κενής Series	s.empty → False

Πίνακας 3.3.3: Μέθοδοι για την Δημιουργία και Βασικών Πληροφοριών των Σειρών

Αυτή η ενότητα αφορά την **«Ανατομία»** μιας Σειράς (Series). Πριν αρχίσεις να φιλτράρεις ή να υπολογίζεις μέσους όρους, πρέπει να ξέρεις τι ακριβώς κρατάς στα χέρια σου.

Αυτά τα χαρακτηριστικά (attributes) σου λένε τα πάντα για τη δομή και το περιεχόμενο των δεδομένων σου.

1. Δημιουργία και Βασική Δομή

Μια Σειρά αποτελείται από δύο βασικούς πυλώνες: τις **Τιμές** και τις **Ετικέτες** τους (Index).

- **pd.Series(data)**: Είναι ο "κατασκευαστής" (constructor). Μπορείς να μετατρέψεις λίστες, λεξικά ή NumPy arrays σε μια Σειρά.

Tip: Αν χρησιμοποιήσεις λεξικό, τα κλειδιά γίνονται αυτόματα το index της Σειράς.

2. Ιδιότητες Μεγέθους και Σχήματος

- **.shape**: Επειδή η Σειρά είναι μονοδιάστατη, σου επιστρέφει πάντα ένα tuple με έναν αριθμό, π.χ. (100,). Αυτό σημαίνει 100 γραμμές. Είναι εξαιρετικά χρήσιμο όταν θέλεις να βεβαιωθείς ότι δύο Σειρές έχουν το ίδιο μήκος πριν τις προσθέσεις.
- **.size**: Σου δίνει το συνολικό πλήθος των στοιχείων ως ακέραιο. Στις Σειρές, το .size είναι πάντα ίσο με το πρώτο στοιχείο του .shape.
- **.empty**: Μια γρήγορη "ερώτηση" Boolean. Επιστρέφει True αν η Σειρά δεν περιέχει κανένα στοιχείο. Χρήσιμο σε αυτοματισμούς για να μην "κρασάρει" ο κώδικας αν ένα φίλτρο δεν βρει αποτελέσματα.

3. Πρόσβαση στα Δεδομένα και Τύποι

- **.index**: Σου δείχνει τις "διευθύνσεις" των δεδομένων. Μπορεί να είναι απλοί αριθμοί (RangeIndex), κείμενο ή ακόμα και ημερομηνίες.

- **.values**: Σου δίνει μόνο το «ζουμί». Επιστρέφει τα δεδομένα σε μορφή **NumPy array**. Είναι χρήσιμο όταν θέλεις να κάνεις πράξεις χαμηλού επιπέδου που δεν απαιτούν την πολυπλοκότητα των Pandas.
- **.dtype**: Ίσως το πιο κρίσιμο attribute. Σου λέει αν η Σειρά περιέχει ακέραιους (int64), δεκαδικούς (float64), κείμενο (object) ή κατηγορίες.
    - *Direct correction:* Αν δεις τύπο object, συνήθως σημαίνει ότι η Σειρά περιέχει κείμενο ή ανάμεικτους τύπους δεδομένων.

4. Μεταδεδομένα

- **.name**: Μπορείς να δώσεις μια ταυτότητα στη Σειρά σου. Αυτό είναι πολύ χρήσιμο γιατί αν αργότερα ενώσεις αυτή τη Σειρά με άλλες για να φτιάξεις έναν πίνακα (DataFrame), το .name θα γίνει αυτόματα ο **τίτλος της στήλης**.

```
import pandas as pd
import numpy as np

# Δημιουργία ενός Series από λίστα
data = [10, 20, 30, 40, 50]
s = pd.Series(data, index=['a', 'b', 'c', 'd', 'e'], name='Numbers')

print("Αρχικό Series:")
print(s)
print("\n" + "="*60 + "\n")

# 1. pd.Series(data) - Δημιουργία Series
print("1. Δημιουργία Series από λίστα:")
s2 = pd.Series([1, 2, 3, 4, 5])
print(s2)
print("-"*40)

# 2. .shape - Διάσταση (tuple με μήκος)
print("2. .shape:", s.shape)  # (5,)
print("-"*40)

# 3. .size - Αριθμός στοιχείων (int)
print("3. .size:", s.size)  # 5
print("-"*40)

# 4. .index - Το ευρετήριο (Index object)
print("4. .index:")
print(s.index)
print("   Τύπος:", type(s.index))
print("   Λίστα ετικετών:", list(s.index))
print("-"*40)

# 5. .values - Οι τιμές ως numpy array
print("5. .values:")
print(s.values)
print("   Τύπος:", type(s.values))  # numpy.ndarray
print("-"*40)
```

```
# 6. .dtype - Τύπος δεδομένων
print("6. .dtype:", s.dtype)  # int64
print("-"*40)

# 7. .name - Όνομα του Series
print("7. .name (αρχικά):", s.name)  # 'Numbers'
# Αλλαγή ονόματος
s.name = 'UpdatedName'
print("   Μετά από s.name = 'UpdatedName':", s.name)
print("-"*40)

# 8. .empty - Έλεγχος αν το Series είναι κενό
print("8. .empty (για s):", s.empty)  # False
empty_series = pd.Series([])
print("   Για empty_series:", empty_series.empty)  # True
print("-"*40)

print("\n" + "="*60 + "\n")
print("Σημειώσεις:")
print("- Το .shape επιστρέφει ένα tuple (π.χ. (5,) )")
print("- Το .size είναι ακέραιος αριθμός ίσος με len(s)")
print("- Το .index περιέχει τις ετικέτες")
print("- Το .values είναι numpy array (προτιμάται .to_numpy() σε νεότερες εκδόσεις)")
print("- Το .dtype δείχνει τον τύπο δεδομένων")
print("- Το .name μπορεί να αλλάξει ή να οριστεί κατά τη δημιουργία")
print("- Το .empty είναι χρήσιμο για έλεγχο πριν από επεξεργασία")
```

Κ. 3.3.1: Παράδειγμα για τις Μεθόδους για την Δημιουργία και Βασικών Πληροφοριών των Σειρών

3.3.2. Επιλογή & Πρόσβαση Στοιχείων

Μέθοδος	Περιγραφή	Σύνταξη
[key]	Βασική επιλογή	s['a']
.loc[]	Επιλογή με ετικέτα	s.loc['a']
.iloc[]	Επιλογή με θέση	s.iloc[0]
.at[]	Γρήγορη με ετικέτα	s.at['a']
.iat[]	Γρήγορη με θέση	s.iat[0]
.get(key, default)	Ασφαλής επιλογή	s.get('x', 0)
[[key1, key2]]	Πολλαπλές ετικέτες	s[['a','b']]
[start:stop]	Φέτα (slicing)	s[1:4]

Πίνακας 3.3.4: Μέθοδοι για την Επιλογή και Πρόσβαση Στοιχείων στις Σειρές

```
import pandas as pd
s = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
```

1. [key] - Η Βασική Επιλογή

Λειτουργεί σχεδόν όπως ένα λεξικό (dictionary) στην Python. Αν του δώσεις την ετικέτα, σου επιστρέφει την τιμή.

- **Παράδειγμα:** s['a'] -> Επιστρέφει 10.
- **Σημείωση:** Αν το index ήταν νούμερα (π.χ. 0, 1, 2), θα λειτουργούσε με αυτά.

2. .loc[] - Επιλογή με Ετικέτα (Label-based)

Είναι ο πιο ρητός τρόπος για να πεις "φέρε μου το στοιχείο που έχει αυτό το όνομα".

- **Παράδειγμα:** s.loc['b'] -> Επιστρέφει 20.
- **Ιδιαιτερότητα:** Αν κάνεις slicing με .loc['a':'c'], το 'c' **συμπεριλαμβάνεται** στο αποτέλεσμα.

3. .iloc[] - Επιλογή με Θέση (Integer-based)

Εδώ δεν μας ενδιαφέρουν τα ονόματα, αλλά η σειρά. Το "i" στο .iloc σημαίνει **integer**.

- **Παράδειγμα:** s.iloc[0] -> Επιστρέφει 10 (το πρώτο στοιχείο, ανεξάρτητα αν λέγεται 'a').
- **Ιδιαιτερότητα:** Στο slicing s.iloc[0:2], η τελευταία θέση (2) **δεν συμπεριλαμβάνεται** (λειτουργεί όπως οι λίστες της Python).

4. .at[] & .iat[] - Οι "Γρήγοροι" της παρέας

Αυτές οι δύο μέθοδοι είναι βελτιστοποιημένες για να επιστρέφουν **ένα και μόνο στοιχείο** (scalar). Είναι ταχύτερες από τις .loc και .iloc γιατί δεν χρειάζεται να διαχειριστούν slicing ή arrays.

- **.at['a']:** Γρήγορη πρόσβαση με ετικέτα.
- **.iat[0]:** Γρήγορη πρόσβαση με θέση.
- **Πότε τις χρησιμοποιούμε:** Όταν θέλουμε να διαβάσουμε ή να αλλάξουμε μία συγκεκριμένη τιμή πολύ γρήγορα σε ένα μεγάλο DataFrame.

5. .get(key, default) - Η Ασφαλής Επιλογή

Αν προσπαθήσεις να ζητήσεις το s['z'], η Python θα "χτυπήσει" σφάλμα (KeyError). Η .get() σε προστατεύει.

- **Παράδειγμα:** s.get('x', 0) -> Επειδή το 'x' δεν υπάρχει, θα σου επιστρέψει 0 αντί να κρασάρει το πρόγραμμα.

6. [[key1, key2]] - Πολλαπλές Ετικέτες (Fancy Indexing)

Όταν θέλεις παραπάνω από ένα στοιχεία, αλλά όχι απαραίτητα συνεχόμενα, περνάς μια λίστα μέσα στις αγκύλες.

- **Παράδειγμα:** s[['a', 'c']] -> Θα σου επιστρέψει ένα νέο Series με τα στοιχεία 10 και 30.

7. [start:stop] - Φέτα (Slicing)

Κλασικός τρόπος για να πάρεις ένα "κομμάτι" των δεδομένων.

- **Παράδειγμα:** s[1:4] -> Επιστρέφει τα στοιχεία από τη θέση 1 έως τη θέση 3 (20, 30, 40).
- **Προσοχή:** Όταν χρησιμοποιείς ακέραιους για slicing, το Pandas συμπεριφέρεται όπως οι λίστες (το stop εξαιρείται).

```
import pandas as pd
import numpy as np

# Δημιουργία ενός Series με index από γράμματα (string) και τιμές
data = [10, 20, 30, 40, 50, 60]
index_labels = ['a', 'b', 'c', 'd', 'e', 'f']
s = pd.Series(data, index=index_labels, name='Δεδομένα')

print("Αρχικό Series:")
print(s)
print("\n" + "="*60 + "\n")

# 1. [key] - Βασική επιλογή με ετικέτα
print("1. s['c']:")
print(s['c'])
print("-"*40)

# 2. .loc[label] - Επιλογή με ετικέτα (επίσημη μέθοδος)
print("2. s.loc['d']:")
print(s.loc['d'])
print("-"*40)

# 3. .iloc[position] - Επιλογή με θέση (0-based)
print("3. s.iloc[2]:")
print(s.iloc[2])
print("-"*40)

# 4. .at[label] - Γρήγορη επιλογή με ετικέτα (μόνο για ένα στοιχείο)
print("4. s.at['e']:")
print(s.at['e'])
print("-"*40)

# 5. .iat[position] - Γρήγορη επιλογή με θέση (μόνο για ένα στοιχείο)
print("5. s.iat[4]:")
print(s.iat[4])
print("-"*40)

# 6. .get(key, default) - Ασφαλής επιλογή (επιστρέφει default αν δεν
υπάρχει το key)
print("6. s.get('c'):")
print(s.get('c'))
print("   s.get('x', 0):")
print(s.get('x', 0))
print("-"*40)

# 7. [[key1, key2]] - Επιλογή πολλαπλών ετικετών (επιστρέφει Series)
print("7. s[['a', 'c', 'e']]:")
print(s[['a', 'c', 'e']])
print("-"*40)
```

```
# 8. [start:stop] - Slicing (φέτα) με βάση τη θέση (όχι την ετικέτα)
# Σημείωση: Σε Series, το slicing με αριθμούς λειτουργεί με βάση τη θέση,
όχι την ετικέτα.
print("8. s[1:4] (slicing με βάση θέση 1 έως 3):")
print(s[1:4])  # Επιστρέφει τις θέσεις 1,2,3 (στοιχεία b, c, d)
print("-"*40)

# Slicing με ετικέτες μέσω .loc (συμπεριλαμβάνει και τα δύο άκρα)
print("   s.loc['b':'e'] (slicing με ετικέτες, συμπεριλαμβανομένου του
'e'):")
print(s.loc['b':'e'])  # Επιστρέφει b,c,d,e
print("-"*40)

# Επιπλέον: Slicing με βήμα
print("   s[::2] (κάθε δεύτερο στοιχείο):")
print(s[::2])
print("-"*40)

print("\n" + "="*60 + "\n")
print("Σημειώσεις:")
print("- Η βασική επιλογή s['a'] και η .loc['a'] είναι παρόμοιες, αλλά η
.loc είναι πιο σαφής και προτιμάται.")
print("- Η .iloc χρησιμοποιεί θέσεις, ενώ η .loc χρησιμοποιεί ετικέτες.")
print("- Οι .at και .iat είναι ταχύτερες για πρόσβαση σε ένα μόνο
στοιχείο.")
print("- Η .get είναι χρήσιμη όταν δεν είμαστε σίγουροι αν υπάρχει η
ετικέτα.")
print("- Το slicing [start:stop] λειτουργεί με βάση τη θέση (όπως σε
λίστες), ενώ .loc με ετικέτες συμπεριλαμβάνει και το stop.")
```

Κ. 3.3.2: Παράδειγμα με τις Μεθόδους για την Επιλογή και Πρόσβαση Στοιχείων στις Σειρές

3.3.3. Boolean Indexing & Φιλτράρισμα

Μέθοδος	Περιγραφή	Επιστροφή
[boolean_series]	Boolean indexing	Series
.where(cond)	Διατήρηση όπου True	Series
.mask(cond)	Διατήρηση όπου False	Series
.between(left, right)	Τιμές μεταξύ ορίων	Boolean Series
.isin(values)	Ανήκει σε λίστα	Boolean Series
.isna()	Έλεγχος για NaN	Boolean Series
.notna()	Έλεγχος για μη-NaN	Boolean Series
.isnull()	Ισοδύναμο του isna()	Boolean Series
.notnull()	Ισοδύναμο του notna()	Boolean Series

Πίνακας 3.3.5: Μέθοδοι για Boolean Indexing & Φιλτράρισμα

Εδώ περνάμε στο πιο «έξυπνο» κομμάτι των Pandas: το **φιλτράρισμα**. Αυτές οι μέθοδοι δεν επιλέγουν απλώς δεδομένα βάσει θέσης, αλλά βάσει **συνθηκών** (λογικής).

Ας χρησιμοποιήσουμε αυτό το Series για τα παραδείγματα:

```python
import pandas as pd
import numpy as np

s = pd.Series([10, 25, np.nan, 40, 55], index=['A', 'B', 'C', 'D', 'E'])
```

1. [boolean_series] - Boolean Indexing

Είναι η πιο συνηθισμένη μέθοδος. Δημιουργείς μια λίστα από True/False και την περνάς στο Series. Μόνο τα True "επιβιώνουν".

- **Παράδειγμα:** s[s > 30]
- **Αποτέλεσμα:** Θα επιστρέψει τις τιμές 40 και 55.
- **Πώς δουλεύει:** Το s > 30 δημιουργεί εσωτερικά ένα Series με True/False.

2. .where(cond) - Διατήρηση όπου True

Η .where κρατάει τις τιμές που ικανοποιούν τη συνθήκη και **αντικαθιστά τις υπόλοιπες** (συνήθως με NaN).

- **Παράδειγμα:** s.where(s > 30)
- **Αποτέλεσμα:** Θα δεις NaN, NaN, NaN, 40.0, 55.0.
- **Χρήση:** Πολύ χρήσιμο όταν θέλεις να κρατήσεις το μέγεθος του Series ίδιο, αλλά να "κρύψεις" τιμές.

3. .mask(cond) - Διατήρηση όπου False

Είναι το ακριβώς αντίθετο της .where. "Μασκάρει" (κρύβει) τις τιμές που είναι True.

- **Παράδειγμα:** s.mask(s > 30)
- **Αποτέλεσμα:** Θα κρύψει το 40 και το 55 (θα γίνουν NaN) και θα αφήσει τα υπόλοιπα.

4. .between(left, right) - Τιμές μεταξύ ορίων

Αντί να γράφεις (s > 10) & (s < 50), χρησιμοποιείς αυτή τη μέθοδο. Είναι πιο καθαρή και ευανάγνωστη.

- **Παράδειγμα:** s.between(20, 50)
- **Επιστροφή:** Ένα Boolean Series (True για τις τιμές 25 και 40).

5. .isin(values) - Ανήκει σε λίστα

Ελέγχει αν οι τιμές του Series υπάρχουν μέσα σε μια λίστα που του δίνεις εσύ.

- **Παράδειγμα:** s.isin([10, 55, 100])
- **Επιστροφή:** True για το 'A' (10) και το 'E' (55).

6. Έλεγχος για κενά (Missing Values)

Εδώ έχουμε δύο ζευγάρια μεθόδων που στην πραγματικότητα κάνουν το ίδιο πράγμα:

- **.isna() & .isnull():** Επιστρέφουν **True** αν η τιμή είναι NaN (Missing).
    - *Παράδειγμα:* s.isna() -> True μόνο για το στοιχείο 'C'.
- **.notna() & .notnull():** Επιστρέφουν **True** αν η τιμή **υπάρχει** (δεν είναι κενή).
    - *Παράδειγμα:* s.notna() -> True για όλα εκτός από το 'C'.

Note: Η χρήση των isna/notna προτιμάται συνήθως στη σύγχρονη Python, καθώς το όνομα είναι πιο ακριβές (N/A - Not Available).

```
import pandas as pd
import numpy as np

# Δημιουργία ενός Series με διάφορες τιμές, συμπεριλαμβανομένων NaN
data = [10, 25, 30, np.nan, 45, 50, 18, np.nan, 32, 27]
s = pd.Series(data, name='Τιμές')

print("Αρχικό Series:")
print(s)
print("\n" + "="*50 + "\n")

# 1. Boolean indexing (επιλογή βάσει συνθήκης)
# Επιλέγουμε μόνο τις τιμές που είναι μεγαλύτερες από 30
boolean_indexed = s[s > 30]
print("1. Boolean indexing (s > 30):")
print(boolean_indexed)
print("\n" + "="*50 + "\n")

# 2. .where(cond) – διατηρεί τις τιμές όπου η συνθήκη είναι True, αλλιώς
NaN
# Εδώ διατηρούμε τιμές > 20, αλλιώς NaN
where_result = s.where(s > 20)
print("2. .where(s > 20):")
print(where_result)
print("\n" + "="*50 + "\n")

# 3. .mask(cond) – διατηρεί τις τιμές όπου η συνθήκη είναι False, αλλιώς
NaN
# Εδώ αποκρύπτουμε (mask) τιμές > 20, δηλ. διατηρούμε τιμές ≤ 20
mask_result = s.mask(s > 20)
print("3. .mask(s > 20):")
print(mask_result)
print("\n" + "="*50 + "\n")
```

```
# 4. .between(left, right) – επιστρέφει boolean Series αν η τιμή βρίσκεται μεταξύ δύο ορίων
between_result = s.between(20, 40)  # συμπεριλαμβάνονται τα όρια
print("4. .between(20, 40):")
print(between_result)
print("\n" + "="*50 + "\n")

# 5. .isin(values) – ελέγχει αν η τιμή ανήκει σε μία λίστα/σύνολο
isin_result = s.isin([25, 45, 50, 100])
print("5. .isin([25, 45, 50, 100]):")
print(isin_result)
print("\n" + "="*50 + "\n")

# 6. .isna() – επιστρέφει True για τιμές NaN
isna_result = s.isna()
print("6. .isna():")
print(isna_result)
print("\n" + "="*50 + "\n")

# 7. .notna() – επιστρέφει True για μη-NaN τιμές
notna_result = s.notna()
print("7. .notna():")
print(notna_result)
print("\n" + "="*50 + "\n")

# 8. .isnull() – ισοδύναμο του .isna()
isnull_result = s.isnull()
print("8. .isnull() (ίδιο με .isna()):")
print(isnull_result)
print("\n" + "="*50 + "\n")

# 9. .notnull() – ισοδύναμο του .notna()
notnull_result = s.notnull()
print("9. .notnull() (ίδιο με .notna()):")
print(notnull_result)
print("\n" + "="*50 + "\n")

# Επιπλέον: Χρήση των boolean Series για φιλτράρισμα
print("Εφαρμογή boolean Series στο αρχικό Series (π.χ. s[s.between(20,40)]):")
print(s[s.between(20, 40)])
```

Κ. 3.3.3: Παράδειγμα για τις μεθόδους BOOLEAN INDEXING & ΦΙΛΤΡΑΡΙΣΜΑ

3.3.4. Ταξινόμηση & Αναδιάταξη

Μέθοδος	Περιγραφή	Παράμετροι
.sort_values()	Ταξινόμηση κατά τιμές	ascending=True

.sort_index()	Ταξινόμηση κατά ευρετήριο	ascending=True
.reindex(new_index)	Αλλαγή ευρετηρίου	fill_value=None
.reset_index()	Επαναφορά ευρετηρίου	drop=False
.reindex_like(other)	Ίδιο index με άλλη	-
.nlargest(n)	n μεγαλύτερες τιμές	keep='first'
.nsmallest(n)	n μικρότερες τιμές	keep='first'

Πίνακας 3.3.6: Μέθοδοι για ταξινόμηση & αναδιάταξη

Ενώ οι προηγούμενες μέθοδοι αφορούσαν την επιλογή, αυτές εδώ αφορούν την **οργάνωση**, τη **διάταξη** και τη **μορφή** του Series σας.

Ας χρησιμοποιήσουμε ένα Series με ονόματα προϊόντων και τις τιμές τους:

```
import pandas as pd

# Τιμές πώλησης για 4 προϊόντα (με ανακατεμένο index)
prices = pd.Series([150, 20, 300, 80], index=['Laptop', 'Mouse', 'Monitor',
'Keyboard'])
```

1. sort_values() & sort_index() - Η Ταξινόμηση

Αυτές οι μέθοδοι βάζουν τα δεδομένα σας σε σειρά.

- **.sort_values(ascending=True)**: Ταξινομεί με βάση το περιεχόμενο (τις τιμές).
    - *Παράδειγμα:* prices.sort_values() -> Θα φέρει πρώτο το Mouse (20) και τελευταίο το Monitor (300).
- **.sort_index()**: Ταξινομεί με βάση το όνομα του index (αλφαβητικά).
    - *Παράδειγμα:* prices.sort_index() -> Θα φέρει πρώτο το Keyboard και τελευταίο το Mouse.

2. reindex() & reindex_like() - Αλλαγή Δομής

- **.reindex(new_index)**: Σας επιτρέπει να αλλάξετε τη σειρά των γραμμών ή να προσθέσετε νέες. Αν το νέο index περιέχει κάτι που δεν υπήρχε πριν, τα Pandas θα βάλουν NaN.
    - *Παράδειγμα:* prices.reindex(['Laptop', 'Mouse', 'Tablet'], fill_value=0) -> Θα κρατήσει Laptop/Mouse και θα προσθέσει το 'Tablet' με τιμή 0.
- **.reindex_like(other)**: Παίρνει το index από ένα άλλο Series ή DataFrame και το επιβάλλει στο δικό σας. Είναι ο «καθρέφτης» της δομής ενός άλλου αντικειμένου.

3. reset_index() - Επαναφορά στο Μηδέν

Αυτή η μέθοδος «πετάει» το υπάρχον index (π.χ. τα ονόματα των προϊόντων) και βάζει το κλασικό αριθμητικό index (0, 1, 2...).

- **drop=False**: Το παλιό index γίνεται στήλη (μετατρέποντας το Series σε DataFrame).
- **drop=True**: Το παλιό index διαγράφεται οριστικά.
    - *Παράδειγμα:* prices.reset_index(drop=True) -> Θα έχετε μόνο τις τιμές με index 0, 1, 2, 3.

4. nlargest() & nsmallest() - Οι Πρωταθλητές

Αντί να ταξινομήσετε όλο το Series (που είναι αργό σε μεγάλα δεδομένα), αυτές οι μέθοδοι σας δίνουν γρήγορα τους "Top N" ή τους "Bottom N".

- **.nlargest(2)**: Θα επιστρέψει τις δύο πιο ακριβές τιμές (Monitor: 300, Laptop: 150).
- **.nsmallest(1)**: Θα επιστρέψει τη φθηνότερη τιμή (Mouse: 20).

```
import pandas as pd
import numpy as np

# Δημιουργία ενός Series με μη ταξινομημένες τιμές και τυχαίο index
values = [42, 15, 30, 8, 25, 50, 12, 100, 33, 7]
index_labels = ['zeta', 'alpha', 'gamma', 'beta', 'delta', 'epsilon',
'eta', 'theta', 'iota', 'kappa']
s = pd.Series(values, index=index_labels, name='Numbers')

print("Αρχικό Series (μη ταξινομημένο):")
print(s)
print("\n" + "="*60 + "\n")

# 1. .sort_values() - Ταξινόμηση κατά τιμές (ascending=True)
sorted_by_values_asc = s.sort_values(ascending=True)
print("1. .sort_values(ascending=True):")
print(sorted_by_values_asc)
print("\n" + "="*60 + "\n")

# .sort_values() με descending
sorted_by_values_desc = s.sort_values(ascending=False)
print("   .sort_values(ascending=False):")
print(sorted_by_values_desc)
print("\n" + "="*60 + "\n")

# 2. .sort_index() - Ταξινόμηση κατά ευρετήριο (ascending=True)
sorted_by_index_asc = s.sort_index(ascending=True)
print("2. .sort_index(ascending=True):")
print(sorted_by_index_asc)
print("\n" + "="*60 + "\n")
```

```
# .sort_index() με descending
sorted_by_index_desc = s.sort_index(ascending=False)
print("   .sort_index(ascending=False):")
print(sorted_by_index_desc)
print("\n" + "="*60 + "\n")

# 3. .reindex(new_index) - Αλλαγή ευρετηρίου, με fill_value για νέα
στοιχεία
new_index = ['alpha', 'beta', 'gamma', 'delta', 'epsilon', 'zeta', 'eta',
'theta', 'lambda', 'mu']
reindexed = s.reindex(new_index, fill_value=0)  # Συμπλήρωση με 0 όπου δεν
υπάρχει τιμή
print("3. .reindex(new_index, fill_value=0):")
print(reindexed)
print("\n" + "="*60 + "\n")

# 4. .reset_index() - Επαναφορά ευρετηρίου σε προκαθορισμένο ακέραιο index
# drop=False: η παλιά ετικέτα γίνεται στήλη (σε Series γίνεται Series με
index 0,1,2,... και το όνομα του index προστίθεται ως 'index')
reset_drop_false = s.reset_index(drop=False)  # drop=False -> διατηρεί το
παλιό index ως στήλη 'index' (Series -> Series με index 0..n και τα
δεδομένα? Προσοχή: reset_index σε Series επιστρέφει DataFrame αν
drop=False; Ας το διευκρινίσουμε.)
# Στην πραγματικότητα, η .reset_index() σε Series επιστρέφει DataFrame αν
drop=False, γιατί δημιουργεί στήλη με τον παλιό index. Για να το δείξουμε
σωστά, θα το διαχωρίσουμε:
print("4. .reset_index(drop=False): Επιστρέφει DataFrame:")
reset_df = s.reset_index(drop=False)
print(reset_df)
print("\nΑν drop=True, το index γίνεται default integer index, χωρίς
διατήρηση της παλιάς ετικέτας:")
reset_drop_true = s.reset_index(drop=True)
print(reset_drop_true)
print("\n" + "="*60 + "\n")

# 5. .reindex_like(other) - Προσαρμογή index σύμφωνα με άλλο Series
# Δημιουργούμε ένα άλλο Series με διαφορετικό index
other_series = pd.Series([1, 2, 3, 4], index=['alpha', 'beta', 'gamma',
'delta'])
reindex_like_result = s.reindex_like(other_series)  # Παίρνει το index του
other_series, τιμές από το s (ή NaN αν δεν υπάρχουν)
print("5. .reindex_like(other_series) με other_series index
['alpha','beta','gamma','delta']:")
print(reindex_like_result)
print("\n" + "="*60 + "\n")

# 6. .nlargest(n) - Επιστροφή των n μεγαλύτερων τιμών
n_largest_3 = s.nlargest(3)  # κρατάει default keep='first' (αν υπάρχουν
ισοπαλίες)
print("6. .nlargest(3):")
print(n_largest_3)
print("\n" + "="*60 + "\n")

# .nlargest με keep='last' (προαιρετικά)
n_largest_3_last = s.nlargest(3, keep='last')
print("   .nlargest(3, keep='last'):")
print(n_largest_3_last)
print("\n" + "="*60 + "\n")

# 7. .nsmallest(n) - Επιστροφή των n μικρότερων τιμών
n_smallest_4 = s.nsmallest(4)
```

```
print("7. .nsmallest(4):")
print(n_smallest_4)
print("\n" + "="*60 + "\n")

# .nsmallest με keep='last'
n_smallest_4_last = s.nsmallest(4, keep='last')
print("   .nsmallest(4, keep='last'):")
print(n_smallest_4_last)
print("\n" + "="*60 + "\n")
```

Κ. 3.3.4: Παράδειγμα για τις μεθόδους ταξινόμηση & αναδιάταξη

3.3.5. Επεξεργασία NaN Τιμών

Μέθοδος	Περιγραφή	Παράμετροι
.dropna()	Διαγραφή NaN τιμών	axis=0
.fillna(value)	Συμπλήρωση NaN	method=None
.ffill()	Forward fill	-
.bfill()	Backward fill	-
.isna()	Έλεγχος για NaN	-
.notna()	Έλεγχος για μη-NaN	-
.interpolate()	Παρεμβολή τιμών	method='linear'

Πίνακας 3.3.7: Μέθοδοι για επεξεργασία NaN τιμών

Τώρα μπαίνουμε στην καρδιά του **Data Cleaning**. Στον πραγματικό κόσμο, τα δεδομένα είναι «βρώμικα» και γεμάτα κενά (NaN - Not a Number). Αυτές οι μέθοδοι είναι τα εργαλεία που χρησιμοποιείς για να αποφασίσεις αν θα πετάξεις τα προβληματικά δεδομένα ή αν θα τα «μαντέψεις» για να σώσεις την παρτίδα.

Ας πάρουμε ένα Series που αντιπροσωπεύει θερμοκρασίες μέσα στην ημέρα, όπου ο αισθητήρας είχε κάποιες διακοπές:

```
import pandas as pd
import numpy as np

# Θερμοκρασίες με κενά
temp = pd.Series([20, np.nan, 22, np.nan, np.nan, 26], index=['08:00',
'09:00', '10:00', '11:00', '12:00', '13:00'])
```

1. .dropna() - Η Ριζική Λύση

Αν δεν εμπιστεύεσαι τα κενά, απλά τα διαγράφεις.

- **Παράδειγμα:** temp.dropna() -> Θα επιστρέψει μόνο τις 3 ώρες που έχουν τιμή (08:00, 10:00, 13:00).

- **Παράμετρος axis:** Στο Series είναι πάντα 0 (γραμμές). Στα DataFrames μπορείς να διαγράψεις ολόκληρες στήλες με axis=1.

2. .fillna(value) - Η Σταθερή Συμπλήρωση

Αντικαθιστάς κάθε NaN με μια συγκεκριμένη τιμή ή ένα στατιστικό στοιχείο (π.χ. τον μέσο όρο).

- **Παράδειγμα:** temp.fillna(0) -> Όλα τα κενά γίνονται 0.
- **Πιο έξυπνο:** temp.fillna(temp.mean()) -> Γεμίζει τα κενά με τη μέση θερμοκρασία των υπόλοιπων ωρών.

3. .ffill() & .bfill() - Η Λογική της Γειτονιάς

Αυτές οι μέθοδοι είναι «τεμπέλικες» αλλά αποτελεσματικές, ειδικά σε χρονοσειρές.

- **.ffill() (Forward Fill):** Παίρνει την τελευταία γνωστή τιμή και τη σπρώχνει προς τα εμπρός.
    - *Αποτέλεσμα:* Στις 09:00 θα βάλει 20 (αυτό που είχε στις 08:00).
- **.bfill() (Backward Fill):** Παίρνει την επόμενη γνωστή τιμή και τη φέρνει προς τα πίσω.
    - *Αποτέλεσμα:* Στις 09:00 θα βάλει 22 (αυτό που έχει στις 10:00).

4. .isna() & .notna() - Ο Έλεγχος

Τις είδαμε και πριν, αλλά εδώ είναι η βασική τους χρήση: ο εντοπισμός.

- **.isna():** Σου δείχνει πού έχεις «τρύπες». Αν γράψεις temp.isna().sum(), θα σου πει ακριβώς πόσες τιμές λείπουν (εδώ 3).
- **.notna():** Το αντίθετο. Χρήσιμο για να δεις πόσα δεδομένα είναι έγκυρα.

5. .interpolate() - Η «Έξυπνη» Πρόβλεψη

Αντί να γεμίσει τα κενά με μια σταθερή τιμή, η .interpolate() προσπαθεί να βρει τη διαδρομή μεταξύ δύο σημείων. Η προεπιλεγμένη μέθοδος είναι η **linear** (γραμμική).

- **Παράδειγμα:** Αν στις 08:00 έχει 20 και στις 10:00 έχει 22, η μέθοδος θα «μαντέψει» ότι στις 09:00 η θερμοκρασία ήταν 21.
- **Μαθηματικά:** Για γραμμική παρεμβολή μεταξύ δύο σημείων (x_1, y_1) και (x_2, y_2), η τιμή y στο σημείο x υπολογίζεται ως:

$$y = y_1 + (x - x_1)\frac{y_2 - y_1}{x_2 - x_1}$$

Η παράμετρος inplace=True, αν τη χρησιμοποιήσεις, η αλλαγή θα γίνει **πάνω στο αρχικό Series** και δεν θα χρειαστεί να το εκχωρήσεις ξανά (π.χ. temp.dropna(inplace=True) αντί για temp = temp.dropna()). Ωστόσο, η σύγχρονη τάση στα Pandas είναι να το αποφεύγουμε για να έχουμε πιο καθαρό κώδικα.

```
import pandas as pd
import numpy as np

# Δημιουργία Series με τιμές και NaN (με ετικέτες string)
data = [10, 25, np.nan, 30, np.nan, 45, 50, np.nan, 18, 32]
index_labels = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
s = pd.Series(data, index=index_labels, name='Δεδομένα')

print("Αρχικό Series με NaN:")
print(s)
print("\n" + "="*60 + "\n")

# 1. .dropna()
s_dropna = s.dropna()
print("1. .dropna():")
print(s_dropna)
print("\n" + "="*60 + "\n")

# 2. .fillna(value)
s_fillna_value = s.fillna(0)
print("2. .fillna(0):")
print(s_fillna_value)
print("\n" + "="*60 + "\n")

# 3. .fillna(method='ffill')
s_fillna_ffill = s.fillna(method='ffill')
print("3. .fillna(method='ffill'):")
print(s_fillna_ffill)
print("\n" + "="*60 + "\n")

# 4. .ffill()
s_ffill = s.ffill()
print("4. .ffill():")
print(s_ffill)
print("\n" + "="*60 + "\n")

# 5. .bfill()
s_bfill = s.bfill()
print("5. .bfill():")
print(s_bfill)
print("\n" + "="*60 + "\n")

# 6. .isna()
s_isna = s.isna()
print("6. .isna():")
print(s_isna)
print("\n" + "="*60 + "\n")

# 7. .notna()
s_notna = s.notna()
print("7. .notna():")
print(s_notna)
print("\n" + "="*60 + "\n")
```

```
# 8. .interpolate(method='linear') - λειτουργεί με οποιοδήποτε index
s_interpolate_linear = s.interpolate(method='linear')
print("8. .interpolate(method='linear'):")
print(s_interpolate_linear)
print("\n" + "="*60 + "\n")

# 9. .interpolate(method='polynomial', order=2) - απαιτεί αριθμητικό index
# Δημιουργούμε προσωρινό Series με αριθμητικό index (reset_index)
s_numeric_index = s.reset_index(drop=True)
s_interpolate_poly = s_numeric_index.interpolate(method='polynomial',
order=2)
print("9. .interpolate(method='polynomial', order=2) (με αριθμητικό
index):")
print(s_interpolate_poly)
print("\n" + "="*60 + "\n")

# Επιπλέον: .bfill() με limit
s_bfill_limit = s.bfill(limit=1)
print("Επιπλέον: .bfill(limit=1):")
print(s_bfill_limit)
```

Κ. 3.3.5: Παράδειγμα για τις μεθόδους NaN τιμών

3.3.6. Μετασχηματισμοί Τύπων & Τιμών

Μέθοδος	Περιγραφή	Εφαρμογή
.astype(dtype)	Αλλαγή τύπου δεδομένων	s.astype(float)
.convert_dtypes()	Μετατροπή σε βελτιστοποιημένους τύπους	-
.infer_objects()	Εκτίμηση τύπων αντικειμένων	-
.replace(to_replace)	Αντικατάσταση τιμών	{παλιά: νέα}
.map(mapper)	Αντιστοίχιση τιμών	Λεξικό/συνάρτηση
.apply(func)	Εφαρμογή συνάρτησης	-
.rename(name)	Αλλαγή ονόματος Series	-
.copy(deep)	Δημιουργία αντιγράφου	deep=True

Πίνακας 3.3.8: Μέθοδοι για μετασχηματισμούς τύπων και τιμών

Αυτές οι μέθοδοι δεν αλλάζουν απλώς τη διάταξη των δεδομένων, αλλά τη **φύση** τους (τον τύπο τους) και το **περιεχόμενό** τους. Είναι τα εργαλεία που χρησιμοποιούμε για να κάνουμε τα δεδομένα μας «καθαρά» και συμβατά με μαθηματικά μοντέλα.

Ας χρησιμοποιήσουμε ένα Series που περιέχει «ακατάστατα» δεδομένα:

```
import pandas as pd
import numpy as np

# Δεδομένα που μοιάζουν με αριθμούς αλλά είναι strings, και μερικά codes
s = pd.Series(['1', '2', '3.5', '4'], name='Prices')
codes = pd.Series(['A', 'B', 'A', 'C'], name='Categories')
```

1. astype() & convert_dtypes() - Η Αλλαγή Τύπου

- **.astype(dtype)**: Αναγκάζει το Series να μετατραπεί σε έναν συγκεκριμένο τύπο. Πολύ χρήσιμο όταν φορτώνεις δεδομένα και οι αριθμοί διαβάζονται ως κείμενο.
    - **Παράδειγμα:** s.astype(float) -> Μετατρέπει τα strings σε δεκαδικούς αριθμούς.
- **.convert_dtypes()**: Η πιο «έξυπνη» έκδοση. Το Pandas εξετάζει τα δεδομένα και επιλέγει μόνο του τον καλύτερο δυνατό τύπο (π.χ. χρησιμοποιεί τους νέους nullable integer τύπους αν υπάρχουν κενά).

2. .infer_objects() - Η Εκτίμηση

Χρησιμοποιείται κυρίως όταν έχεις ένα Series με τύπο object (γενικό κείμενο/μείγμα). Η μέθοδος προσπαθεί να καταλάβει αν τα αντικείμενα μέσα είναι στην πραγματικότητα κάτι πιο συγκεκριμένο (όπως integers ή floats) και τα μετατρέπει αναλόγως.

3. .replace() & .map() - Η Αντικατάσταση

Εδώ αλλάζουμε το περιεχόμενο των κελιών.

- **.replace({παλιό: νέο})**: Ιδανικό για να διορθώσεις συγκεκριμένα λάθη.
    - **Παράδειγμα:** s.replace({'1': '10'}) -> Όπου έβλεπε '1', τώρα βάζει '10'.
- **.map(mapper)**: Πιο ισχυρό. Μπορεί να πάρει ένα λεξικό ή μια συνάρτηση και να μετασχηματίσει **κάθε** τιμή.
    - **Παράδειγμα:** codes.map({'A': 'High', 'B': 'Medium', 'C': 'Low'}).

4. .apply(func) - Η Εφαρμογή Συναρτήσεων

Αν οι έτοιμες μέθοδοι δεν σου φτάνουν, μπορείς να γράψεις τη δική σου συνάρτηση και να την εφαρμόσεις σε κάθε στοιχείο.

- **Παράδειγμα:** s.astype(float).apply(lambda x: x**2) -> Υπολογίζει το τετράγωνο κάθε τιμής.

5. .rename() & .copy() - Ταυτότητα και Ασφάλεια

- **.rename('NewName')**: Αλλάζει το όνομα του ίδιου του Series (όχι των περιεχομένων). Είναι χρήσιμο όταν προετοιμάζεις ένα Series για να το ενώσεις (merge) με ένα άλλο DataFrame.

- **.copy(deep=True)**: **Προσοχή εδώ.** Αν γράψεις s2 = s, τότε όποια αλλαγή κάνεις στο s2 θα αλλάξει και το αρχικό s. Με το .copy(), δημιουργείς ένα εντελώς ανεξάρτητο αντίγραφο στη μνήμη.

```
import pandas as pd
import numpy as np

# Δημιουργία Series με μικτούς τύπους δεδομένων
data = ['1', '2', '3.5', '4.0', '5', '6', np.nan, '8', '9', '10']
s = pd.Series(data, index=range(10, 20), name='original_series')
print("Αρχικό Series (με strings και NaN):")
print(s)
print("Τύπος δεδομένων:", s.dtype)
print("\n" + "="*60 + "\n")

# 1. .astype(dtype) - Αλλαγή τύπου σε float (αγνοεί τα NaN; τα NaN
μετατρέπονται σε float NaN)
s_float = s.astype(float)
print("1. .astype(float):")
print(s_float)
print("Νέος τύπος:", s_float.dtype)
print("\n" + "="*60 + "\n")

# 2. .convert_dtypes() - Μετατροπή σε βελτιστοποιημένους τύπους pandas
# (π.χ. Int64 για ακέραιους που υποστηρίζουν NaN, string τύπο κλπ.)
s_converted = s.convert_dtypes()
print("2. .convert_dtypes():")
print(s_converted)
print("Νέος τύπος:", s_converted.dtype)  # Πιθανά string ή Int64 ανάλογα
print("\n" + "="*60 + "\n")

# 3. .infer_objects() - Εκτίμηση τύπων για object columns
# Δημιουργούμε ένα Series με τύπο object αλλά αριθμητικά δεδομένα ως
strings
s_object = pd.Series(['1', '2.2', '3', '4.5'], dtype='object')
print("Πριν .infer_objects():")
print(s_object)
print("dtype:", s_object.dtype)
s_inferred = s_object.infer_objects()
print("3. .infer_objects():")
print(s_inferred)
print("dtype μετά:", s_inferred.dtype)  # Πρέπει να γίνει float64 αν όλα τα
strings μπορούν να γίνουν αριθμοί
print("\n" + "="*60 + "\n")

# 4. .replace(to_replace) - Αντικατάσταση τιμών
# Αντικατάσταση συγκεκριμένων τιμών, π.χ. το '3.5' με 999, το NaN με -1
# Μπορούμε να χρησιμοποιήσουμε λεξικό {παλιά: νέα}
s_replaced = s.replace({'3.5': 999, np.nan: -1})
print("4. .replace({'3.5': 999, np.nan: -1}):")
print(s_replaced)
print("\n" + "="*60 + "\n")

# 5. .map(mapper) - Αντιστοίχιση τιμών με λεξικό ή συνάρτηση
# Π.χ. αντιστοίχιση αριθμητικών τιμών (μετά από μετατροπή) σε κατηγορίες
# Πρώτα μετατρέπουμε σε float για να έχουμε αριθμούς
s_num = s.astype(float)
# Λεξικό αντιστοίχισης: τιμές < 5 -> 'μικρό', >=5 -> 'μεγάλο'
```

```
# Προσοχή: .map() με λεξικό θα αντικαταστήσει μόνο όσες τιμές υπάρχουν ως κλειδιά, οι υπόλοιπες γίνονται NaN
# Γι' αυτό καλύτερα με συνάρτηση:
def categorize(val):
    if pd.isna(val):
        return 'άγνωστο'
    elif val < 5:
        return 'μικρό'
    else:
        return 'μεγάλο'

s_mapped = s_num.map(categorize)
print("5. .map(categorize) (με συνάρτηση):")
print(s_mapped)
print("\n" + "="*60 + "\n")

# 6. .apply(func) - Εφαρμογή συνάρτησης σε κάθε στοιχείο
# Μπορούμε να εφαρμόσουμε οποιαδήποτε συνάρτηση, π.χ. τετράγωνο
s_squared = s_num.apply(lambda x: x**2 if not pd.isna(x) else np.nan)
print("6. .apply(lambda x: x**2) (τετράγωνο, αγνοώντας NaN):")
print(s_squared)
print("\n" + "="*60 + "\n")

# 7. .rename(name) - Αλλαγή ονόματος Series
s_renamed = s.rename('renamed_series')
print("7. .rename('renamed_series'):")
print(s_renamed)
print("Όνομα:", s_renamed.name)
print("\n" + "="*60 + "\n")

# 8. .copy(deep=True) - Δημιουργία αντιγράφου
s_copy = s.copy(deep=True)
print("8. .copy(deep=True):")
print(s_copy)
print("Είναι το ίδιο αντικείμενο;", s_copy is s)  # False
print("Τροποποιούμε το αντίγραφο - προσθέτουμε 100 στα πρώτα 5 στοιχεία")
s_copy.iloc[:5] = s_copy.iloc[:5].astype(float) + 100
print("Τροποποιημένο αντίγραφο:")
print(s_copy)
print("Αρχικό παραμένει ίδιο:")
print(s.head())
print("\n" + "="*60 + "\n")
```

Κ. 3.3.6: Παράδειγμα για τις μεθόδους μετασχηματισμούς τύπων και τιμών

3.3.7. Στατιστικές Μέθοδοι

Μέθοδος	Περιγραφή	Επιστροφή
.count()	Πλήθος μη-NaN	Ακέραιος
.sum()	Άθροισμα	Αριθμός
.mean()	Μέσος όρος	Αριθμός
.median()	Διάμεσος	Αριθμός
.mode()	Επικρατούσα τιμή	Series
.min()	Ελάχιστη τιμή	Αριθμός
.max()	Μέγιστη τιμή	Αριθμός

Μέθοδος	Περιγραφή	Επιστροφή
.std()	Τυπική απόκλιση	Αριθμός
.var()	Διακύμανση	Αριθμός
.sem()	Σφάλμα μέσου όρου	Αριθμός
.quantile(q)	Ποσοστημόριο	Αριθμός
.skew()	Λοξότητα	Αριθμός
.kurtosis()	Κύρτωση	Αριθμός
.describe()	Πλήρης στατιστική σύνοψη	

Πίνακας 3.3.9: Μέθοδοι για Στατιστική επεξεργασία

Αυτές οι μέθοδοι ονομάζονται **Περιγραφική Στατιστική (Descriptive Statistics)** και μας επιτρέπουν να καταλάβουμε την «προσωπικότητα» των δεδομένων μας με μερικά μόνο κλικ.

Ας χρησιμοποιήσουμε ένα Series που αντιπροσωπεύει τους βαθμούς μιας τάξης (με ένα κενό για έναν μαθητή που έλειπε):

```
import pandas as pd
import numpy as np

grades = pd.Series([12, 15, 15, 18, 20, np.nan, 14], name="Grades")
```

1. Κεντρική Τάση (Πού «γέρνουν» τα δεδομένα;)

Αυτές οι τιμές μας δείχνουν το κέντρο των δεδομένων μας.

- **.mean() (Μέσος Όρος):** Το άθροισμα όλων των τιμών δια το πλήθος τους.

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

- **.median() (Διάμεσος):** Η μεσαία τιμή αν βάλουμε τα νούμερα στη σειρά. Είναι πιο «ανθεκτική» σε ακραίες τιμές (outliers) από τον μέσο όρο.
- **.mode() (Επικρατούσα τιμή):** Η τιμή που εμφανίζεται πιο συχνά. Στο παράδειγμά μας είναι το 15.

Σημείωση: Επιστρέφει **Series**, γιατί μπορεί να έχουμε δύο ή περισσότερες τιμές που ισοψηφούν στην πρώτη θέση.

2. Διασπορά (Πόσο «απλωμένα» είναι τα δεδομένα;)

Μας λένε αν οι βαθμοί είναι όλοι κοντά στο 15 ή αν έχουμε μεγάλες αποκλίσεις.

- **.std() (Τυπική Απόκλιση):** Μας δείχνει πόσο απέχουν οι τιμές από τον μέσο όρο. Υψηλό std σημαίνει μεγάλη ανομοιομορφία.

$$\sigma = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n-1}}$$

- **.var() (Διακύμανση):** Το τετράγωνο της τυπικής απόκλισης (σ^2).
- **.sem() (Standard Error of the Mean):** Μετράει την ακρίβεια με την οποία ο μέσος όρος του δείγματος αντιπροσωπεύει τον μέσο όρο του πληθυσμού.

3. Σχήμα Κατανομής & Θέση

- **.quantile(q):** Επιστρέφει την τιμή κάτω από την οποία βρίσκεται ένα ποσοστό των δεδομένων. Π.χ. το .quantile(0.5) είναι η διάμεσος.
- **.skew() (Λοξότητα):** Δείχνει αν η κατανομή είναι συμμετρική. Αν είναι θετική, έχουμε «ουρά» προς τις μεγάλες τιμές.
- **.kurtosis() (Κύρτωση):** Δείχνει πόσο «αιχμηρή» ή «πλατιά» είναι η κατανομή (αν έχουμε πολλές ακραίες τιμές).

4. Οι Βασικοί Μετρητές

- **.count():** Μετράει πόσες τιμές υπάρχουν (εξαιρεί τα NaN). Στο παράδειγμά μας θα επιστρέψει 6, ενώ το len(grades) θα επέστρεφε 7.
- **.sum():** Το άθροισμα όλων των βαθμών.
- **.min() / .max():** Ο χαμηλότερος και ο υψηλότερος βαθμός.

5. .describe() – Συγκεντρωτικά στοιχεία περιγραφικής στατιστικής

.describe() σου δίνει τα σημαντικότερα (count, mean, std, min, 25%, 50%, 75%, max) με μία κίνηση. Είναι το πρώτο πράγμα που τρέχει κάθε Data Analyst μόλις πάρει νέα δεδομένα στα χέρια του.

```
import pandas as pd
import numpy as np

# Δημιουργία ενός Series με αριθμητικά δεδομένα και μερικές NaN τιμές
data = [15, 22, 31, 44, np.nan, 28, 35, 41, np.nan, 19, 27, 33]
index_labels = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L']
s = pd.Series(data, index=index_labels, name='Δεδομένα')
```

```
print("Αρχικό Series:")
print(s)
print("\n" + "="*60 + "\n")

# 1. .count() - Πλήθος μη-NaN τιμών
count_val = s.count()
print("1. .count():", count_val)
print("Τύπος:", type(count_val))
print("-"*40)

# 2. .sum() - Άθροισμα (αγνοεί NaN)
sum_val = s.sum()
print("2. .sum():", sum_val)

# 3. .mean() - Μέσος όρος
mean_val = s.mean()
print("3. .mean():", mean_val)

# 4. .median() - Διάμεσος
median_val = s.median()
print("4. .median():", median_val)

# 5. .mode() - Επικρατούσα τιμή (επιστρέφει Series)
mode_val = s.mode()
print("5. .mode():")
print(mode_val)
print("Τύπος επιστροφής:", type(mode_val))

# 6. .min() - Ελάχιστη τιμή
min_val = s.min()
print("6. .min():", min_val)

# 7. .max() - Μέγιστη τιμή
max_val = s.max()
print("7. .max():", max_val)

# 8. .std() - Τυπική απόκλιση (δείγματος)
std_val = s.std()
print("8. .std():", std_val)

# 9. .var() - Διακύμανση (δείγματος)
var_val = s.var()
print("9. .var():", var_val)

# 10. .sem() - Τυπικό σφάλμα του μέσου όρου
sem_val = s.sem()
print("10. .sem():", sem_val)

# 11. .quantile(q) - Ποσοστημόριο (εδώ 0.25, 0.5, 0.75)
q25 = s.quantile(0.25)
q50 = s.quantile(0.5)   # ίδιο με .median()
q75 = s.quantile(0.75)
print("11. .quantile(0.25):", q25)
print("    .quantile(0.50):", q50)
print("    .quantile(0.75):", q75)

# 12. .skew() - Λοξότητα (ασυμμετρία)
skew_val = s.skew()
print("12. .skew():", skew_val)
```

```
# 13. .kurtosis() - Κύρτωση
kurt_val = s.kurtosis()
print("13. .kurtosis():", kurt_val)

print("\n" + "="*60 + "\n")

# 14. .describe() - Πλήρης στατιστική σύνοψη
desc = s.describe()
print("14. .describe():")
print(desc)
print("\n" + "="*60 + "\n")

# Επιπλέον: Χρήση παραμέτρων (π.χ. skipna=False για να ληφθούν υπόψη τα
NaN)
print("Με skipna=False (αν υπάρχουν NaN, οι περισσότερες επιστρέφουν
NaN):")
print("sum (skipna=False):", s.sum(skipna=False))
print("mean (skipna=False):", s.mean(skipna=False))
# Η .quantile() δεν έχει skipna, πάντα αγνοεί NaN (ή αν όλες NaN επιστρέφει
NaN)
print("quantile (πάντα αγνοεί NaN):", s.quantile(0.5))
```

Κ. 3.3.7: Παράδειγμα για τις στατιστικές μεθόδους

3.3.8. Σωρευτικοί Υπολογισμοί και Μεταβολές

Μέθοδος	Περιγραφή	Επιστροφή
.cumsum()	Σωρευτικό άθροισμα	Series
.cumprod()	Σωρευτικό γινόμενο	Series
.cummax()	Σωρευτικό μέγιστο	Series
.cummin()	Σωρευτικό ελάχιστο	Series
.diff(periods)	Διαφορές από προηγούμενο	Series
.pct_change(periods)	Ποσοστιαίες αλλαγές	Series

Πίνακας 3.3.10: Μέθοδοι Σωρευτικών Υπολογισμών και Μεταβολών

Αυτές οι μέθοδοι είναι εξαιρετικά χρήσιμες στα χρηματοοικονομικά δεδομένα και στις χρονοσειρές, καθώς μας επιτρέπουν να δούμε την εξέλιξη ενός μεγέθους στον χρόνο.

Ας υποθέσουμε ότι έχουμε τις ημερήσιες εισπράξεις ενός καταστήματος για μια εβδομάδα:

```
import pandas as pd

# Ημερήσιες εισπράξεις
sales = pd.Series([100, 120, 110, 150, 140], index=['Δευ', 'Τρι', 'Τετ',
'Πεμ', 'Παρ'])
```

1. Σωρευτικοί Υπολογισμοί (Cumulative)

Αυτές οι μέθοδοι "χτίζουν" πάνω στις προηγούμενες τιμές, δημιουργώντας μια εικόνα της συνολικής πορείας.

- **.cumsum() (Cumulative Sum):** Προσθέτει κάθε νέα τιμή στο τρέχον άθροισμα.
    - *Παράδειγμα:* Τη Δευτέρα έχουμε 100, την Τρίτη 100+120=220, την Τετάρτη 220+110=330 κ.ο.κ.
- **.cumprod() (Cumulative Product):** Πολλαπλασιάζει διαδοχικά τις τιμές. Χρησιμοποιείται κυρίως σε ανατοκισμούς ή σύνθετες αποδόσεις.
- **.cummax() / .cummin():** Κρατούν το "ρεκόρ" υψηλότερης ή χαμηλότερης τιμής που έχει σημειωθεί μέχρι εκείνη τη στιγμή.
    - *Παράδειγμα:* Αν οι τιμές είναι [100, 120, 110], το .cummax() θα δώσει [100, 120, 120].

2. Υπολογισμοί Μεταβολών (Differences)

Εδώ δεν μας ενδιαφέρει το σύνολο, αλλά η σύγκριση μεταξύ των ημερών.

- **.diff(periods=1):** Υπολογίζει τη διαφορά της τρέχουσας τιμής από την προηγούμενη.
    - *Παράδειγμα:* Την Τρίτη θα έχουμε 120 - 100 = 20. Την Τετάρτη 110 - 120 = -10.
- **.pct_change(periods=1):** Υπολογίζει το ποσοστό μεταβολής. Είναι το "Α" και το "Ω" για να δούμε αν μια μετοχή ή μια πωλήση ανέβηκε ή έπεσε ποσοστιαία.
    - *Τύπος:* $\frac{V_{now} - V_{prev}}{V_{prev}}$
    - *Παράδειγμα:* Από Δευτέρα σε Τρίτη η μεταβολή είναι $+20\%$.

```
import pandas as pd
import numpy as np

# Δημιουργία Series με αριθμητικά δεδομένα και ένα NaN
data = [10, 25, 30, 45, 12, 8, 33, np.nan, 27, 50]
index_labels = [f'item_{i}' for i in range(10)]
s = pd.Series(data, index=index_labels, name='Original')

print("Αρχικό Series:")
print(s)
print("\n" + "="*60 + "\n")

# 1. .cumsum() - Σωρευτικό άθροισμα
cumsum_result = s.cumsum()
print("1. .cumsum() (σωρευτικό άθροισμα):")
```

```
print(cumsum_result)
print("\n" + "="*60 + "\n")

# 2. .cumprod() - Σωρευτικό γινόμενο
cumprod_result = s.cumprod()
print("2. .cumprod() (σωρευτικό γινόμενο):")
print(cumprod_result)
print("\n" + "="*60 + "\n")

# 3. .cummax() - Σωρευτικό μέγιστο
cummax_result = s.cummax()
print("3. .cummax() (σωρευτικό μέγιστο):")
print(cummax_result)
print("\n" + "="*60 + "\n")

# 4. .cummin() - Σωρευτικό ελάχιστο
cummin_result = s.cummin()
print("4. .cummin() (σωρευτικό ελάχιστο):")
print(cummin_result)
print("\n" + "="*60 + "\n")

# 5. .diff(periods=1) - Διαφορές από προηγούμενο (προεπιλογή periods=1)
diff_1 = s.diff()  # ίδιο με s.diff(1)
print("5. .diff() (διαφορά από προηγούμενο στοιχείο, periods=1):")
print(diff_1)
print("\n" + "="*60 + "\n")

# Διαφορές με periods=2
diff_2 = s.diff(periods=2)
print("   .diff(periods=2) (διαφορά από στοιχείο δύο θέσεις πριν):")
print(diff_2)
print("\n" + "="*60 + "\n")

# 6. .pct_change(periods=1) - Ποσοστιαία μεταβολή
pct_1 = s.pct_change()  # προεπιλογή periods=1
print("6. .pct_change() (ποσοστιαία μεταβολή από προηγούμενο):")
print(pct_1)
print("\n" + "="*60 + "\n")

# Ποσοστιαία μεταβολή με periods=2
pct_2 = s.pct_change(periods=2)
print("   .pct_change(periods=2) (ποσοστιαία μεταβολή από δύο θέσεις
πριν):")
print(pct_2)
print("\n" + "="*60 + "\n")

# Επιπλέον: συμπεριφορά με fill_method (για χειρισμό NaN)
# Μπορούμε να ορίσουμε πώς θα αντιμετωπιστούν τα NaN πριν τον υπολογισμό
(π.χ. .cumsum(skipna=False))
# Ας δούμε τη διαφορά αν δεν αγνοηθούν τα NaN στο σωρευτικό άθροισμα
cumsum_nan = s.cumsum(skipna=False)
print("Με skipna=False στο cumsum() (σταματάει στο NaN):")
print(cumsum_nan)
```

Κ. 3.3.8:Παράδειγμα Μεθόδων Σωρευτικών Υπολογισμών και Μεταβολών

3.3.9. Μαθηματικές Πράξεις – Αριθμητικές Μέθοδοι

Μέθοδος	Περιγραφή	Ισοδύναμο
.add(other)	Πρόσθεση	s + other
.sub(other)	Αφαίρεση	s - other
.mul(other)	Πολλαπλασιασμός	s * other
.div(other)	Διαίρεση	s / other
.floordiv(other)	Ακέραια διαίρεση	s // other
.mod(other)	Υπόλοιπο διαίρεσης	s % other
.pow(exponent)	Δύναμη	s ** exponent
.abs()	Απόλυτη τιμή	-
.round(decimals)	Στρογγυλοποίηση	-
.clip(lower, upper)	Περικοπή τιμών	-

Πίνακας 3.3.11: Μέθοδοι Αριθμητικών Πράξεων

Ίσως αναρωτηθείς: *«Γιατί να χρησιμοποιήσω τη μέθοδο .add() όταν μπορώ απλά να γράψω +;»*. Η απάντηση κρύβεται στην ευελιξία: οι μέθοδοι των Pandas (όπως η .add, .sub, κ.λπ.) σου επιτρέπουν να ορίσεις πώς θα χειριστείς τις τιμές που λείπουν (NaN) μέσω της παραμέτρου fill_value, κάτι που οι απλοί τελεστές δεν μπορούν να κάνουν.

Ας δούμε τις μεθόδους με ένα παράδειγμα με αποθέματα δύο αποθηκών:

```
import pandas as pd
import numpy as np

store_a = pd.Series([10, 20, np.nan, 40], index=['Apples', 'Oranges',
'Bananas', 'Pears'])
store_b = pd.Series([5, 10, 15, 20], index=['Apples', 'Oranges', 'Bananas',
'Pears'])
```

1. Βασικές Πράξεις (Arithmetic)

Αυτές οι μέθοδοι εκτελούν πράξεις στοιχείο προς στοιχείο (element-wise).

- **.add() / .sub()**: Πρόσθεση και Αφαίρεση.
    - *Pro Tip:* store_a.add(store_b, fill_value=0) -> Αν λείπουν οι μπανάνες από την αποθήκη Α, θα τις θεωρήσει 0 και θα προσθέσει το 15 της αποθήκης Β.
- **.mul() / .div()**: Πολλαπλασιάζει ή διαιρεί τις τιμές.
- **.floordiv()**: Επιστρέφει μόνο το ακέραιο μέρος της διαίρεσης (π.χ. $7 // 2 = 3$).
- **.mod()**: Επιστρέφει το υπόλοιπο της διαίρεσης (π.χ. $7 \% 2 = 1$).
- **.pow()**: Υψώνει τις τιμές σε μια δύναμη.

2. Μαθηματική Μορφοποίηση

- **.abs()**: Μετατρέπει όλες τις αρνητικές τιμές σε θετικές. Χρήσιμο όταν υπολογίζεις αποκλίσεις ή σφάλματα.
- **.round(decimals)**: Στρογγυλοποιεί τις τιμές στα δεκαδικά ψηφία που θες.
    - *Παράδειγμα:* pd.Series([3.14159]).round(2) -> 3.14.

3. .clip(lower, upper) - Ο «Κόφτης»

Αυτή η μέθοδος είναι εξαιρετικά χρήσιμη για τον περιορισμό των ακραίων τιμών (outliers).

- Αν μια τιμή είναι μικρότερη από το lower, γίνεται ίση με το lower.
- Αν μια τιμή είναι μεγαλύτερη από το upper, γίνεται ίση με το upper.
- *Παράδειγμα:* Αν έχεις βαθμολογίες και θέλεις να σιγουρευτείς ότι καμία δεν ξεπερνά το 20 ή πέφτει κάτω από το 0: grades.clip(0, 20).

```
import pandas as pd
import numpy as np

# Δημιουργία αρχικού Series s
data_s = [10, -5, 7.5, 12, 0, 8, 15, -3.2, 9, 4.8]
index_s = [f's{i}' for i in range(10)]
s = pd.Series(data_s, index=index_s, name='s')

# Δημιουργία Series other για πράξεις
data_other = [2, 3, 1.5, 4, 2, 1, 3, 2, 5, 2]
index_other = [f's{i}' for i in range(10)]  # ίδιο index για ευθυγράμμιση
other = pd.Series(data_other, index=index_other, name='other')

print("Αρχικό Series s:")
print(s)
print("\nSeries other:")
print(other)
print("\n" + "="*70 + "\n")

# 1. Πρόσθεση .add()
print("1. .add(other):")
print(s.add(other))
print("Ισοδύναμο s + other:")
print(s + other)
print("-"*50)

# 2. Αφαίρεση .sub()
print("2. .sub(other):")
print(s.sub(other))
print("Ισοδύναμο s - other:")
print(s - other)
print("-"*50)

# 3. Πολλαπλασιασμός .mul()
print("3. .mul(other):")
```

```
print(s.mul(other))
print("Ισοδύναμο s * other:")
print(s * other)
print("-"*50)

# 4. Διαίρεση .div()
print("4. .div(other):")
print(s.div(other))
print("Ισοδύναμο s / other:")
print(s / other)
print("-"*50)

# 5. Ακέραια διαίρεση .floordiv()
print("5. .floordiv(other):")
print(s.floordiv(other))
print("Ισοδύναμο s // other:")
print(s // other)
print("-"*50)

# 6. Υπόλοιπο .mod()
print("6. .mod(other):")
print(s.mod(other))
print("Ισοδύναμο s % other:")
print(s % other)
print("-"*50)

# 7. Δύναμη .pow()
print("7. .pow(2) (τετράγωνο):")
print(s.pow(2))
print("Ισοδύναμο s ** 2:")
print(s ** 2)
print("\n   .pow(other) (s σε δύναμη other):")
print(s.pow(other))
print("Ισοδύναμο s ** other:")
print(s ** other)
print("-"*50)

# 8. Απόλυτη τιμή .abs()
print("8. .abs():")
print(s.abs())
print("-"*50)

# 9. Στρογγυλοποίηση .round()
s_decimal = pd.Series([1.234, 2.345, 3.456, 4.567, 5.678])
print("9. .round(2) σε δεκαδικό Series:")
print(s_decimal)
print("Αποτέλεσμα:")
print(s_decimal.round(2))
print("-"*50)

# 10. Περικοπή .clip()
print("10. .clip(lower=0, upper=10):")
print(s.clip(lower=0, upper=10))
print("-"*50)

print("\n" + "="*70 + "\n")

# Πράξεις με σταθερά
print("Πράξεις με σταθερά (π.χ. 3):")
print("s.add(3):")
print(s.add(3))
```

```
print("s.sub(3):")
print(s.sub(3))
print("s.mul(3):")
print(s.mul(3))
print("s.div(3):")
print(s.div(3))
print("s.floordiv(3):")
print(s.floordiv(3))
print("s.mod(3):")
print(s.mod(3))
print("s.pow(3):")
print(s.pow(3))
print("\n" + "="*70 + "\n")

# Διαχείριση NaN με fill_value
s_nan = pd.Series([1, 2, np.nan, 4, 5])
other_nan = pd.Series([10, np.nan, 30, 40, 50])
print("Series με NaN:")
print(s_nan)
print(other_nan)
print("Πρόσθεση χωρίς fill_value (NaN επικρατεί):")
print(s_nan.add(other_nan))
print("Πρόσθεση με fill_value=0:")
print(s_nan.add(other_nan, fill_value=0))
print("Πολλαπλασιασμός με fill_value=1:")
print(s_nan.mul(other_nan, fill_value=1))
```

Κ. 3.3.9: Παράδειγμα για τις Μεθόδους Αριθμητικών Πράξεων

3.3.10. Σχέσεις Σύγκρισης (Boolean Output)

Μέθοδος	Περιγραφή	Επιστροφή
.eq(other)	Ισούται με	Boolean Series
.ne(other)	Διαφορετικό από	Boolean Series
.gt(other)	Μεγαλύτερο από	Boolean Series
.lt(other)	Μικρότερο από	Boolean Series
.ge(other)	Μεγαλύτερο ή ίσο	Boolean Series
.le(other)	Μικρότερο ή ίσο	Boolean Series
.compare(other)	Σύγκριση με άλλη Series	DataFrame
.equals(other)	Έλεγχος ισότητας	Boolean

Πίνακας 3.3.12: Μέθοδοι για τις Σχέσεις Σύγκρισης (Boolean Output)

Όπως και στις αριθμητικές πράξεις, έτσι και εδώ, οι μέθοδοι των Pandas προσφέρουν μεγαλύτερη ευελιξία από τους απλούς τελεστές (όπως ==, >, <), ειδικά όταν έχουμε να διαχειριστούμε διαφορετικά indices ή κενές τιμές.

Ας δούμε τις μεθόδους χρησιμοποιώντας δύο Series με τιμές μετοχών:

```
import pandas as pd
import numpy as np

stock_a = pd.Series([100, 150, 200], index=['AAPL', 'GOOG', 'MSFT'])
```

```
stock_b = pd.Series([100, 160, np.nan], index=['AAPL', 'GOOG', 'MSFT'])
```

1. Σχέσεις Σύγκρισης (Boolean Output)

Αυτές οι μέθοδοι συγκρίνουν κάθε στοιχείο και επιστρέφουν ένα **Series από True/False**.

- **.eq() (Equal):** Ελέγχει αν οι τιμές είναι ίσες. stock_a.eq(100) -> True για την 'AAPL'.
- **.ne() (Not Equal):** Ελέγχει αν οι τιμές είναι διαφορετικές.
- **.gt() (Greater Than) / .lt() (Less Than):** Μεγαλύτερο ή μικρότερο από.
- **.ge() (Greater or Equal) / .le() (Less or Equal):** Μεγαλύτερο/Μικρότερο ή ίσο.

Γιατί να τις χρησιμοποιήσω; Γιατί μπορείς να συγκρίνεις ένα Series με ένα άλλο Series ή με έναν μεμονωμένο αριθμό (scalar) πολύ εύκολα.

2. Βαθιά Σύγκριση (Deep Comparison)

Εδώ περνάμε σε μεθόδους που μας δείχνουν **πού** και **πώς** διαφέρουν τα δεδομένα μας.

- **.equals(other)**: Αυτή η μέθοδος επιστρέφει **έναν και μόνο** Boolean (True ή False). Ελέγχει αν τα δύο Series είναι πανομοιότυπα (ίδια στοιχεία, ίδιο index, ίδιος τύπος δεδομένων).
    - *Προσοχή:* Το stock_a.equals(stock_b) θα επιστρέψει False γιατί διαφέρουν στην τιμή της 'GOOG' και της 'MSFT'.
- **.compare(other)**: Η πιο χρήσιμη μέθοδος για τον εντοπισμό διαφορών. Επιστρέφει ένα **DataFrame** που δείχνει μόνο τις τιμές που διαφέρουν, δίπλα-δίπλα.
    - *Παράδειγμα:* stock_a.compare(stock_b) θα μας δείξει έναν πίνακα με τις διαφορές στις 'GOOG' και 'MSFT'.

```
import pandas as pd
import numpy as np

# Δημιουργία αρχικού Series s
data_s = [10, -5, 7.5, 12, 0, 8, 15, -3.2, 9, 4.8]
index_s = [f's{i}' for i in range(10)]
s = pd.Series(data_s, index=index_s, name='s')

# Δημιουργία Series other για πράξεις
data_other = [2, 3, 1.5, 4, 2, 1, 3, 2, 5, 2]
index_other = [f's{i}' for i in range(10)]  # ίδιο index για ευθυγράμμιση
other = pd.Series(data_other, index=index_other, name='other')

print("Αρχικό Series s:")
print(s)
print("\nSeries other:")
```

```
print(other)
print("\n" + "="*70 + "\n")

# 1. Πρόσθεση .add()
print("1. .add(other):")
print(s.add(other))
print("Ισοδύναμο s + other:")
print(s + other)
print("-"*50)

# 2. Αφαίρεση .sub()
print("2. .sub(other):")
print(s.sub(other))
print("Ισοδύναμο s - other:")
print(s - other)
print("-"*50)

# 3. Πολλαπλασιασμός .mul()
print("3. .mul(other):")
print(s.mul(other))
print("Ισοδύναμο s * other:")
print(s * other)
print("-"*50)

# 4. Διαίρεση .div()
print("4. .div(other):")
print(s.div(other))
print("Ισοδύναμο s / other:")
print(s / other)
print("-"*50)

# 5. Ακέραια διαίρεση .floordiv()
print("5. .floordiv(other):")
print(s.floordiv(other))
print("Ισοδύναμο s // other:")
print(s // other)
print("-"*50)

# 6. Υπόλοιπο .mod()
print("6. .mod(other):")
print(s.mod(other))
print("Ισοδύναμο s % other:")
print(s % other)
print("-"*50)

# 7. Δύναμη .pow()
print("7. .pow(2) (τετράγωνο):")
print(s.pow(2))
print("Ισοδύναμο s ** 2:")
print(s ** 2)
print("\n   .pow(other) (s σε δύναμη other):")
print(s.pow(other))
print("Ισοδύναμο s ** other:")
print(s ** other)
print("-"*50)

# 8. Απόλυτη τιμή .abs()
print("8. .abs():")
print(s.abs())
print("-"*50)
```

```
# 9. Στρογγυλοποίηση .round()
s_decimal = pd.Series([1.234, 2.345, 3.456, 4.567, 5.678])
print("9. .round(2) σε δεκαδικό Series:")
print(s_decimal)
print("Αποτέλεσμα:")
print(s_decimal.round(2))
print("-"*50)

# 10. Περικοπή .clip()
print("10. .clip(lower=0, upper=10):")
print(s.clip(lower=0, upper=10))
print("-"*50)

print("\n" + "="*70 + "\n")

# Πράξεις με σταθερά
print("Πράξεις με σταθερά (π.χ. 3):")
print("s.add(3):")
print(s.add(3))
print("s.sub(3):")
print(s.sub(3))
print("s.mul(3):")
print(s.mul(3))
print("s.div(3):")
print(s.div(3))
print("s.floordiv(3):")
print(s.floordiv(3))
print("s.mod(3):")
print(s.mod(3))
print("s.pow(3):")
print(s.pow(3))
print("\n" + "="*70 + "\n")

# Διαχείριση NaN με fill_value
s_nan = pd.Series([1, 2, np.nan, 4, 5])
other_nan = pd.Series([10, np.nan, 30, 40, 50])
print("Series με NaN:")
print(s_nan)
print(other_nan)
print("Πρόσθεση χωρίς fill_value (NaN επικρατεί):")
print(s_nan.add(other_nan))
print("Πρόσθεση με fill_value=0:")
print(s_nan.add(other_nan, fill_value=0))
print("Πολλαπλασιασμός με fill_value=1:")
print(s_nan.mul(other_nan, fill_value=1))
```

Κ. 3.3.10: Παράδειγμα για τις Μεθόδους για τις Σχέσεις Σύγκρισης

3.3.11. Μοναδικές Τιμές & Συχνότητες

Μέθοδος	Περιγραφή	Επιστροφή
.unique()	Μοναδικές τιμές	ndarray
.nunique()	Αριθμός μοναδικών τιμών	Integer
.value_counts()	Συχνότητες τιμών	Series
.duplicated()	Επανάληψη τιμών	Boolean Series

Μέθοδος	Περιγραφή	Επιστροφή
.drop_duplicates()	Διαγραφή διπλοεγγραφών	Series
.is_unique	Όλες οι τιμές μοναδικές	Boolean
.is_monotonic_increasing	Αύξουσες τιμές	Boolean
.is_monotonic_decreasing	Φθίνουσες τιμές	Boolean

Πίνακας 3.3.13: Μέθοδοι για την Ανάλυση Μοναδικότητας και Συχνότητας

Αυτή η ομάδα μεθόδων αφορά την **Ανάλυση Μοναδικότητας και Συχνότητας**. Είναι τα εργαλεία που χρησιμοποιείς για να καταλάβεις πόσο "ποικιλόμορφα" είναι τα δεδομένα σου και αν υπάρχουν επαναλήψεις που ίσως αποτελούν σφάλματα ή σημαντικά μοτίβα.

Ας δούμε τις μεθόδους με ένα παράδειγμα από μια λίστα παραγγελιών προϊόντων:

```
import pandas as pd

orders = pd.Series(['Apple', 'Orange', 'Apple', 'Banana', 'Orange',
'Apple'], name="Fruits")
```

1. Εντοπισμός Μοναδικών Τιμών

- **.unique()**: Σου επιστρέφει ποιες είναι οι διαφορετικές τιμές που υπάρχουν, χωρίς να τον νοιάζει πόσες φορές εμφανίζονται.
    - *Αποτέλεσμα:* ['Apple', 'Orange', 'Banana'] (ως NumPy array).
- **.nunique()**: Σου δίνει τον **αριθμό** των μοναδικών τιμών.
    - *Αποτέλεσμα:* 3.
- **.is_unique**: Μια γρήγορη ερώτηση: "Είναι όλα τα στοιχεία διαφορετικά μεταξύ τους;". Επιστρέφει True ή False. Στο παράδειγμά μας: False.

2. Συχνότητες & Κατανομή

- **.value_counts()**: Ίσως η πιο χρήσιμη μέθοδος στα Pandas. Μετράει πόσες φορές εμφανίζεται η κάθε τιμή και τις ταξινομεί από τη συχνότερη στη λιγότερο συχνή.
    - *Αποτέλεσμα:* Apple: 3, Orange: 2, Banana: 1.
    - *Pro Tip:* Με normalize=True σου δίνει τα ποσοστά (π.χ. 0.5 για το Apple).

3. Διαχείριση Διπλοεγγραφών (Duplicates)

- **.duplicated()**: Μαρκάρει με True κάθε τιμή που έχει εμφανιστεί ξανά νωρίτερα.

- *Παράμετρος keep='first'/'last'/False*: Ορίζει ποια εμφάνιση θα θεωρηθεί η "πρωτότυπη".

- **.drop_duplicates()**: Καθαρίζει το Series κρατώντας μόνο την πρώτη εμφάνιση κάθε στοιχείου και διαγράφοντας τις επαναλήψεις.

4. Έλεγχος Τάσης (Monotonicity)

Αυτές οι ιδιότητες (properties) ελέγχουν αν τα δεδομένα ακολουθούν μια συγκεκριμένη κατεύθυνση χωρίς "πισωγυρίσματα".

- **.is_monotonic_increasing**: Επιστρέφει True αν κάθε τιμή είναι ίση ή μεγαλύτερη από την προηγούμενη (π.χ. μια σειρά ημερομηνιών ή αυξανόμενων IDs).

- **.is_monotonic_decreasing**: Επιστρέφει True αν κάθε τιμή είναι ίση ή μικρότερη από την προηγούμενη.

```
import pandas as pd
import numpy as np

# Δημιουργία Series με επαναλαμβανόμενες τιμές, NaN, και διάταξη
data = [5, 3, 8, 3, np.nan, 7, 5, 2, 8, 3, np.nan, 1]
index_labels = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L']
s = pd.Series(data, index=index_labels, name='Δεδομένα')

print("Αρχικό Series:")
print(s)
print("\n" + "="*70 + "\n")

# 1. .unique() - Μοναδικές τιμές (επιστρέφει numpy ndarray)
unique_values = s.unique()
print("1. .unique():")
print(unique_values)
print("Τύπος:", type(unique_values))
print("-"*50)

# 2. .nunique() - Αριθμός μοναδικών τιμών (προεπιλογή dropna=True)
nunique_val = s.nunique()
print("2. .nunique() (dropna=True):", nunique_val)
nunique_val_false = s.nunique(dropna=False)
print("   .nunique(dropna=False) (συμπεριλαμβάνει NaN):",
nunique_val_false)
print("-"*50)

# 3. .value_counts() - Συχνότητες τιμών
value_counts_default = s.value_counts()  # dropna=True, ταξινόμηση φθίνουσα
print("3. .value_counts() (dropna=True):")
print(value_counts_default)
print("\n   .value_counts(dropna=False) (συμπεριλαμβάνει NaN):")
value_counts_nan = s.value_counts(dropna=False)
print(value_counts_nan)
print("-"*50)

# 4. .duplicated() - Boolean Series για διπλότυπες τιμές
```

```
duplicated_default = s.duplicated()  # keep='first'
print("4. .duplicated() (keep='first'):")
print(duplicated_default)
print("\n   .duplicated(keep='last'):")
print(s.duplicated(keep='last'))
print("\n   .duplicated(keep=False) (όλες οι διπλότυπες):")
print(s.duplicated(keep=False))
print("-"*50)

# 5. .drop_duplicates() - Διαγραφή διπλοεγγραφών
drop_duplicates_default = s.drop_duplicates()  # keep='first'
print("5. .drop_duplicates() (keep='first'):")
print(drop_duplicates_default)
print("\n   .drop_duplicates(keep='last'):")
print(s.drop_duplicates(keep='last'))
print("\n   .drop_duplicates(keep=False) (κρατά μόνο τις μοναδικές):")
print(s.drop_duplicates(keep=False))
print("-"*50)

# 6. .is_unique - Ιδιότητα (property) που ελέγχει αν όλες οι τιμές είναι
μοναδικές
print("6. .is_unique:", s.is_unique)
# Δημιουργία Series με όλες μοναδικές τιμές για σύγκριση
s_unique = pd.Series([1,2,3,4,5])
print("   Σε μοναδικό Series:", s_unique.is_unique)
print("-"*50)

# 7. .is_monotonic_increasing - Ιδιότητα ελέγχου μονοτονίας (αύξουσα)
print("7. .is_monotonic_increasing:", s.is_monotonic_increasing)
s_inc = pd.Series([1, 2, 3, 5, 8])
print("   Σε αύξον Series:", s_inc.is_monotonic_increasing)
print("-"*50)

# 8. .is_monotonic_decreasing - Ιδιότητα ελέγχου μονοτονίας (φθίνουσα)
print("8. .is_monotonic_decreasing:", s.is_monotonic_decreasing)
s_dec = pd.Series([9, 7, 5, 3, 1])
print("   Σε φθίνον Series:", s_dec.is_monotonic_decreasing)
print("\n" + "="*70 + "\n")

# Επιπλέον: Μονοτονία με NaN (επιστρέφει False αν υπάρχουν NaN)
s_with_nan = pd.Series([1, 2, np.nan, 4])
print("Μονοτονία με NaN:")
print(s_with_nan.is_monotonic_increasing)  # False
print(s_with_nan.is_monotonic_decreasing)  # False
```

Κ. 3.3.11: Παράδειγμα Μεθόδων για την Ανάλυση Μοναδικότητας και Συχνότητας

3.3.12. Μέθοδοι Κειμένου (str accessor)

Μέθοδος	Περιγραφή	Παράδειγμα
.str.upper()	Κεφαλαία	s.str.upper()
.str.lower()	Πεζά	s.str.lower()
.str.title()	Τίτλος	s.str.title()
.str.len()	Μήκος	s.str.len()
.str.strip()	Αφαίρεση κενών	s.str.strip()

Μέθοδος	Περιγραφή	Παράδειγμα
.str.contains(pat)	Περιέχει μοτίβο	s.str.contains('a')
.str.replace(pat, repl)	Αντικατάσταση	s.str.replace('a','b')
.str.split(pat)	Διαχωρισμός	s.str.split(' ')
.str.startswith(pat)	Ξεκινάει με	s.str.startswith('A')
.str.endswith(pat)	Τελειώνει με	s.str.endswith('Z')
.str.find(sub)	Εύρεση θέσης	s.str.find('a')
.str.extract(pat)	Εξαγωγή με regex	s.str.extract('(\d+)')

Πίνακας 3.3.14: Μέθοδοι επεξεργασίας αλφαριθμητικών (Κειμένου)

Στα Pandas, όλες οι μέθοδοι κειμένου βρίσκονται κάτω από τον "accessor" **.str**. Αυτό είναι απαραίτητο γιατί ένα Series μπορεί να περιέχει διάφορους τύπους δεδομένων, και το .str λέει στα Pandas: «Μετάφερέ με στο περιβάλλον επεξεργασίας κειμένου».

Ας δούμε τις μεθόδους χρησιμοποιώντας μια λίστα με "ακατάστατα" ονόματα χρηστών:

```
import pandas as pd

users = pd.Series([' giannis papas ', 'maria_01', 'eleni konsta',
'nikos.p'], name="Usernames")
```

1. Βασική Μορφοποίηση

Αλλάζουν την εμφάνιση του κειμένου.

- **.str.upper() / .str.lower()**: Μετατρέπει σε ΚΕΦΑΛΑΙΑ ή πεζά.
- **.str.title()**: Κάνει το πρώτο γράμμα κάθε λέξης κεφαλαίο (ιδανικό για ονοματεπώνυμα).
- **.str.strip()**: Η πιο σημαντική μέθοδος για καθαρισμό! Αφαιρεί τα κενά διαστήματα στην αρχή και στο τέλος.
    - *Παράδειγμα:* Το ' giannis papas ' γίνεται 'giannis papas'.

2. Αναζήτηση & Φιλτράρισμα

Επιστρέφουν συνήθως Boolean (True/False), οπότε είναι τέλειες για φιλτράρισμα.

- **.str.contains('pat')**: Ελέγχει αν ένα κομμάτι κειμένου υπάρχει μέσα στη συμβολοσειρά.
- **.str.startswith() / .str.endswith()**: Ελέγχει αν ξεκινάει ή τελειώνει με συγκεκριμένο χαρακτήρα.
    - *Χρήση:* users[users.str.startswith('m')] -> Θα σου φέρει τη 'maria_01'.

- **.str.find()**: Επιστρέφει τη θέση (index) που ξεκινάει το μοτίβο. Αν δεν το βρει, επιστρέφει -1.

3. Μετασχηματισμός & Εξαγωγή

- **.str.replace('old', 'new')**: Αντικαθιστά χαρακτήρες.
    - *Παράδειγμα:* users.str.replace('_', ' ').
- **.str.split(' ')**: "Σπάει" το κείμενο σε λίστα με βάση έναν χαρακτήρα.
    - *Παράδειγμα:* Το 'eleni konsta' γίνεται ['eleni', 'konsta'].
- **.str.extract(pat)**: Χρησιμοποιεί **Regular Expressions (Regex)** για να "τραβήξει" συγκεκριμένα κομμάτια.
    - *Παράδειγμα:* users.str.extract('(\d+)') -> Θα βρει το '01' από το 'maria_01'.
- **.str.len()**: Μετράει πόσους χαρακτήρες έχει το κάθε κείμενο.

```
import pandas as pd
import numpy as np

# Δημιουργία ενός Series με δεδομένα κειμένου (strings) και μερικά NaN
data = [
    ' hello world ',
    'Python programming',
    'DATA SCIENCE',
    np.nan,
    'apple,banana,orange',
    'Hello123',
    '   trim me   ',
    'PYTHON',
    'python',
    '123-456-7890'
]
index_labels = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
s = pd.Series(data, index=index_labels, name='Κείμενο')

print("Αρχικό Series με strings και NaN:")
print(s)
print("\n" + "="*70 + "\n")

# 1. .str.upper() - Μετατροπή σε κεφαλαία
print("1. .str.upper():")
print(s.str.upper())
print("-"*50)

# 2. .str.lower() - Μετατροπή σε πεζά
print("2. .str.lower():")
print(s.str.lower())
print("-"*50)

# 3. .str.title() - Μετατροπή σε τίτλο (πρώτο γράμμα κάθε λέξης κεφαλαίο)
print("3. .str.title():")
print(s.str.title())
print("-"*50)
```

```
# 4. .str.len() - Μήκος κάθε string (σε χαρακτήρες, NaN γίνεται NaN)
print("4. .str.len():")
print(s.str.len())
print("-"*50)

# 5. .str.strip() - Αφαίρεση κενών από την αρχή και το τέλος
print("5. .str.strip():")
print(s.str.strip())
print("-"*50)

# 6. .str.contains(pat) - Έλεγχος αν περιέχει το υπό-στιγμιότυπο 'Py'
(επιστρέφει boolean Series)
print("6. .str.contains('Py'):")
print(s.str.contains('Py'))
print("-"*50)

# 7. .str.replace(pat, repl) - Αντικατάσταση 'apple' με 'fruit'
print("7. .str.replace('apple', 'fruit'):")
print(s.str.replace('apple', 'fruit'))
print("-"*50)

# 8. .str.split(pat) - Διαχωρισμός σε λίστες (με βάση το κόμμα για το
στοιχείο Ε)
print("8. .str.split(','):")
print(s.str.split(','))
print("-"*50)

# 9. .str.startswith(pat) - Ελέγχει αν ξεκινά με 'P' (True/False)
print("9. .str.startswith('P'):")
print(s.str.startswith('P'))
print("-"*50)

# 10. .str.endswith(pat) - Ελέγχει αν τελειώνει με 'n'
print("10. .str.endswith('n'):")
print(s.str.endswith('n'))
print("-"*50)

# 11. .str.find(sub) - Εύρεση θέσης του 'a' (επιστρέφει -1 αν δεν βρεθεί,
NaN για NaN)
print("11. .str.find('a'):")
print(s.str.find('a'))
print("-"*50)

# 12. .str.extract(pat) - Εξαγωγή τμήματος με κανονική έκφραση (π.χ.
αριθμοί)
# Προσοχή: επιστρέφει DataFrame αν υπάρχουν πολλές ομάδες, αλλιώς Series
print("12. .str.extract('(\d+)') - εξαγωγή πρώτων ψηφίων:")
print(s.str.extract('(\d+)'))  # Επιστρέφει DataFrame με μία στήλη
print("-"*50)

# Επιπλέον: .str.extract με named groups για σαφήνεια
print("   .str.extract('(?P<numbers>\d+)') - με όνομα:")
print(s.str.extract('(?P<numbers>\d+)'))

print("\n" + "="*70 + "\n")

# Επίδειξη συμπεριφοράς με NaN (όλες οι μέθοδοι επιστρέφουν NaN)
print("Όλες οι string μέθοδοι διατηρούν τα NaN (όπου υπήρχε NaN, το
αποτέλεσμα είναι NaN).")
```

Κ. 3.3.12: Παράδειγμα για τις Μεθόδους επεξεργασίας αλφαριθμητικών (Κειμένου)

3.3.13. Χρονικές Σειρές (dt accessor)

Μέθοδος	Περιγραφή	Επιστροφή
.dt.year	Έτος	Series
.dt.month	Μήνας	Series
.dt.day	Ημέρα	Series
.dt.hour	Ώρα	Series
.dt.minute	Λεπτά	Series
.dt.second	Δευτερόλεπτα	Series
.dt.dayofweek	Ημέρα εβδομάδας	Series
.dt.dayofyear	Ημέρα έτους	Series
.dt.is_month_start	Αρχή μήνα	Boolean Series
.dt.is_month_end	Τέλος μήνα	Boolean Series
.dt.strftime(format)	Μορφοποίηση	Series

Πίνακας 3.3.15: Μέθοδοι επεξεργασίας ημερομηνιών και ώρας

Όπως το .str ξεκλειδώνει τις μεθόδους κειμένου, έτσι και το **.dt** (Datetime accessor) ξεκλειδώνει τις δυνατότητες διαχείρισης ημερομηνιών και ώρας.

Για να δουλέψουν αυτές οι μέθοδοι, το Series σου πρέπει να έχει τύπο datetime64. Αν έχεις strings, τα μετατρέπεις πρώτα με την pd.to_datetime().

Ας δούμε ένα παράδειγμα με ημερομηνίες παραγγελιών:

```
import pandas as pd

# Δημιουργία Series με ημερομηνίες
dates = pd.to_datetime(pd.Series(['2024-01-01 10:30:00', '2024-05-15
14:45:00', '2024-12-31 23:59:59']))
```

1. Εξαγωγή Στοιχείων (Components)

Αυτές οι ιδιότητες "σπάνε" την ημερομηνία στα μέρη της.

- **.dt.year, .dt.month, .dt.day**: Σου δίνουν το έτος, τον μήνα ή την ημέρα ως αριθμό.
- **.dt.hour, .dt.minute, .dt.second**: Σου δίνουν την ακριβή ώρα της καταγραφής.
- **.dt.dayofweek**: Επιστρέφει έναν αριθμό από 0 έως 6 (0 = Δευτέρα, 6 = Κυριακή).
- **.dt.dayofyear**: Σου λέει ποια μέρα του χρόνου είναι (από 1 έως 366).

2. Λογικοί Έλεγχοι (Properties)

Επιστρέφουν True/False και είναι εξαιρετικά χρήσιμοι για λογιστικές αναφορές.

- **.dt.is_month_start**: Είναι η 1η του μηνός;
- **.dt.is_month_end**: Είναι η τελευταία μέρα του μηνός;

3. Μορφοποίηση με .dt.strftime()

Αυτή η μέθοδος μετατρέπει την ημερομηνία σε κείμενο (string) με τη μορφή που θέλεις εσύ, χρησιμοποιώντας ειδικούς κωδικούς (format codes).

- **%d**: Ημέρα (01-31)
- **%m**: Μήνας (01-12)
- **%Y**: Έτος (2024)
- **%A**: Όνομα ημέρας (π.χ. Monday)

Παράδειγμα: dates.dt.strftime('%d/%m/%Y') -> Μετατρέπει το '2024-01-01' σε '01/01/2024'.

```
import pandas as pd
import numpy as np

# Δημιουργία ενός Series με ημερομηνίες (datetime)
# Χρήση pd.date_range για δημιουργία ημερομηνιών
date_rng = pd.date_range(start='2023-01-15', periods=8, freq='M')  # 8 μήνες, τέλος μήνα
# Προσθήκη μερικών τυχαίων ωρών για να έχουμε και ώρες
timestamps = [
    '2023-01-15 10:30:00',
    '2023-02-20 14:45:30',
    '2023-03-10 08:15:00',
    '2023-04-05 23:59:59',
    '2023-05-25 00:00:01',
    '2023-06-30 12:00:00',
    '2023-07-14 18:30:00',
    '2023-08-01 05:05:05'
]
# Μετατροπή σε datetime Series
s = pd.Series(pd.to_datetime(timestamps), name='Ημερομηνίες')

print("Αρχικό Series με datetime:")
print(s)
print("\n" + "="*70 + "\n")

# 1. .dt.year - Έτος
print("1. .dt.year:")
print(s.dt.year)
print("-"*50)

# 2. .dt.month - Μήνας (1-12)
print("2. .dt.month:")
print(s.dt.month)
print("-"*50)
```

```
# 3. .dt.day - Ημέρα του μήνα
print("3. .dt.day:")
print(s.dt.day)
print("-"*50)

# 4. .dt.hour - Ώρα (0-23)
print("4. .dt.hour:")
print(s.dt.hour)
print("-"*50)

# 5. .dt.minute - Λεπτά
print("5. .dt.minute:")
print(s.dt.minute)
print("-"*50)

# 6. .dt.second - Δευτερόλεπτα
print("6. .dt.second:")
print(s.dt.second)
print("-"*50)

# 7. .dt.dayofweek - Ημέρα εβδομάδας (Δευτέρα=0, Κυριακή=6)
print("7. .dt.dayofweek:")
print(s.dt.dayofweek)
print("-"*50)

# 8. .dt.dayofyear - Ημέρα του έτους (1-366)
print("8. .dt.dayofyear:")
print(s.dt.dayofyear)
print("-"*50)

# 9. .dt.is_month_start - Αν η ημερομηνία είναι αρχή μήνα
print("9. .dt.is_month_start:")
print(s.dt.is_month_start)
print("-"*50)

# 10. .dt.is_month_end - Αν η ημερομηνία είναι τέλος μήνα
print("10. .dt.is_month_end:")
print(s.dt.is_month_end)
print("-"*50)

# 11. .dt.strftime(format) - Μορφοποίηση ημερομηνίας σε string
print("11. .dt.strftime('%Y-%m-%d'):")
print(s.dt.strftime('%Y-%m-%d'))
print("    .dt.strftime('%A %d %B %Y'):")
print(s.dt.strftime('%A %d %B %Y'))  # Π.χ. Monday 15 January 2023
print("-"*50)

print("\n" + "="*70 + "\n")

# Επιπλέον: Άλλες χρήσιμες ιδιότητες (quarter, week, is_leap_year κλπ.)
print("Επιπλέον ιδιότητες .dt:")
print("s.dt.quarter (τρίμηνο):")
print(s.dt.quarter)
print("s.dt.week (εβδομάδα του έτους):")
print(s.dt.isocalendar().week)  # .week έχει καταργηθεί, χρήση
.isocalendar().week
print("s.dt.is_leap_year (δισέκτο έτος;):")
print(s.dt.is_leap_year)
```

Κ. 3.3.13: Παράδειγμα για τις Μεθόδους επεξεργασίας ημερομηνιών και ώρας

3.3.14. Κατηγορικά Δεδομένα (cat accessor)

Μέθοδος	Περιγραφή	Εφαρμογή
.cat.categories	Κατηγορίες	Index
.cat.codes	Κωδικοί κατηγοριών	Series
.cat.add_categories()	Προσθήκη κατηγοριών	-
.cat.remove_categories()	Αφαίρεση κατηγοριών	-
.cat.set_categories()	Ορισμός κατηγοριών	-
.cat.as_ordered()	Διατεταγμένες κατηγορίες	-
.cat.as_unordered()	Μη διατεταγμένες κατηγορίες	-

Πίνακας 3.3.16: Μέθοδοι για Χρήση Κατηγοριών

Αυτές οι μέθοδοι είναι ο κρυφός άσος στο μανίκι σου όταν διαχειρίζεσαι τεράστια δεδομένα με επαναλαμβανόμενες τιμές (π.χ. "Small", "Medium", "Large").

Όπως το .str και το .dt, έτσι και το **.cat** ξεκλειδώνει λειτουργίες ειδικά για τον τύπο category. Η χρήση κατηγοριών αντί για απλό κείμενο (strings) κάνει τον κώδικά σου **πολύ πιο γρήγορο** και μειώνει δραματικά τη χρήση μνήμης.

Ας δούμε ένα παράδειγμα με μεγέθη ρούχων:

```
import pandas as pd

# Δημιουργία κατηγορικού Series
sizes = pd.Series(['Small', 'Large', 'Medium', 'Small'], dtype='category')
```

1. Πληροφορίες & Κωδικοποίηση

Στο παρασκήνιο, τα Pandas δεν αποθηκεύουν τη λέξη "Small" πολλές φορές, αλλά έναν αριθμό (κωδικό).

- **.cat.categories**: Σου δείχνει ποιες είναι οι επιτρεπτές τιμές (το "λεξιλόγιο" της στήλης).
    - *Έξοδος:* Index(['Large', 'Medium', 'Small'], dtype='object').
- **.cat.codes**: Σου δείχνει την αριθμητική αναπαράσταση κάθε τιμής.
    - *Παράδειγμα:* Το 'Small' μπορεί να είναι το 2, το 'Large' το 0. Αυτό είναι έτοιμη τροφή για μοντέλα Machine Learning!

2. Διαχείριση "Λεξιλογίου"

Μπορείς να τροποποιήσεις τις διαθέσιμες επιλογές χωρίς να πειράξεις τα δεδομένα.

- **.cat.add_categories(['XL'])**: Επιτρέπει στο Series να δεχτεί και την τιμή 'XL' στο μέλλον.
- **.cat.remove_categories(['Small'])**: Αφαιρεί την επιλογή. Αν υπήρχαν 'Small' στα δεδομένα σου, θα γίνουν NaN.
- **.cat.set_categories(['Small', 'Medium', 'Large'])**: Ορίζει από την αρχή όλο το σύνολο των επιτρεπτών τιμών.

3. Διάταξη (Ordering)

Αυτό είναι το πιο ισχυρό σημείο. Οι κατηγορίες μπορούν να έχουν **ιεραρχία**.

- **.cat.as_ordered()**: Λέει στα Pandas ότι οι τιμές έχουν σειρά (π.χ. Small < Medium < Large).
    - *Γιατί βοηθάει:* Μπορείς να κάνεις s.min() και να σου επιστρέψει το 'Small', ή να ταξινομήσεις τα δεδομένα σου όχι αλφαβητικά, αλλά βάσει μεγέθους!
- **.cat.as_unordered()**: Αφαιρεί την ιεραρχία, θεωρώντας όλες τις τιμές ισότιμες.

```
import pandas as pd
import numpy as np

# Δημιουργία ενός Series με κατηγορικά δεδομένα (categorical)
data = ['μικρό', 'μεσαίο', 'μεγάλο', 'μικρό', 'μεγάλο', 'μεσαίο', np.nan]
s = pd.Series(data, name='Μέγεθος')
print("Αρχικό Series (object):")
print(s)

# Μετατροπή σε categorical (χωρίς διάταξη)
s_cat = s.astype('category')
print("\nΜετατροπή σε category:")
print(s_cat)
print("Κατηγορίες:", s_cat.cat.categories)
print("Διατεταγμένο;", s_cat.cat.ordered)
print("\n" + "="*70 + "\n")

# 1. .cat.categories - Οι κατηγορίες (Index)
print("1. .cat.categories:")
print(s_cat.cat.categories)
print("Τύπος:", type(s_cat.cat.categories))
print("-"*50)

# 2. .cat.codes - Κωδικοί κατηγοριών (οι NaN κωδικοποιούνται ως -1)
print("2. .cat.codes:")
print(s_cat.cat.codes)
print("-"*50)

# 3. .cat.add_categories() - Προσθήκη νέων κατηγοριών
s_cat_new = s_cat.cat.add_categories(['πολύ μικρό', 'πολύ μεγάλο'])
print("3. .cat.add_categories(['πολύ μικρό', 'πολύ μεγάλο']):")
print(s_cat_new.cat.categories)
print("-"*50)
```

```
# 4. .cat.remove_categories() - Αφαίρεση κατηγοριών (όσες τιμές ανήκουν σε αυτές γίνονται NaN)
s_cat_removed = s_cat.cat.remove_categories(['μεσαίο'])
print("4. .cat.remove_categories(['μεσαίο']):")
print(s_cat_removed)
print("Κατηγορίες:", s_cat_removed.cat.categories)
print("-"*50)

# 5. .cat.set_categories() - Ορισμός νέων κατηγοριών (αντικαθιστά πλήρως)
s_cat_set = s_cat.cat.set_categories(['μικρό', 'μεγάλο'])  # αφαιρεί το 'μεσαίο' και όσες τιμές ήταν 'μεσαίο' γίνονται NaN
print("5. .cat.set_categories(['μικρό', 'μεγάλο']):")
print(s_cat_set)
print("Κατηγορίες:", s_cat_set.cat.categories)
print("-"*50)

# 6. .cat.as_ordered() - Ορισμός των κατηγοριών ως διατεταγμένες (ordered)
# Πρέπει να έχουμε μια λογική σειρά, π.χ. μικρό < μεσαίο < μεγάλο
# Για να γίνει αυτό, πρέπει πρώτα να ορίσουμε τις κατηγορίες με τη σωστή σειρά
s_cat_ordered = s_cat.cat.set_categories(['μικρό', 'μεσαίο', 'μεγάλο'], ordered=True)
print("6. .cat.as_ordered() (μέσω set_categories με ordered=True):")
print(s_cat_ordered)
print("Διατεταγμένο;", s_cat_ordered.cat.ordered)
# Μπορούμε τώρα να συγκρίνουμε
print("Σύγκριση: s_cat_ordered > 'μεσαίο':")
print(s_cat_ordered > 'μεσαίο')  # Θα δώσει True για 'μεγάλο', False για 'μικρό' και NaN
print("-"*50)

# 7. .cat.as_unordered() - Αφαίρεση διάταξης (κάνει unordered)
s_cat_unordered = s_cat_ordered.cat.as_unordered()
print("7. .cat.as_unordered():")
print(s_cat_unordered.cat.ordered)
print("Κατηγορίες:", s_cat_unordered.cat.categories)
print("-"*50)

print("\n" + "="*70 + "\n")
print("Σημείωση: Οι NaN τιμές διατηρούνται ως NaN στις περισσότερες πράξεις.")
```

Κ. 3.3.14: Παράδειγμα των Μεθόδων για Χρήση Κατηγοριών

3.3.15. Ομαδοποίηση & Κινητοί Μέσοι

Μέθοδος	Περιγραφή	Εφαρμογή
.groupby()	Ομαδοποίηση	s.groupby(key)
.rolling(window)	Κινητός μέσος	s.rolling(3).mean()
.expanding()	Επεκτεινόμενος	s.expanding().sum()
.ewm(span)	Εκθετικά σταθμισμένος	s.ewm(span=3).mean()
.aggregate(func)	Συναθροίσεις	s.agg(['mean','sum'])
.transform(func)	Μετασχηματισμός	s.transform('sqrt')

Πίνακας 3.3.17: Μέθοδοι Προχωρημένης Ανάλυσης και Παραθύρων

Μέθοδοι **Προχωρημένης Ανάλυσης και Παραθύρων (Window & Aggregation Functions)**. Αυτές οι μέθοδοι είναι που μετατρέπουν τα Pandas από μια απλή βιβλιοθήκη δεδομένων σε ένα πανίσχυρο εργαλείο στατιστικής και χρηματοοικονομικής ανάλυσης.

Ας χρησιμοποιήσουμε ένα Series με ημερήσιες τιμές μιας μετοχής ή θερμοκρασίες:

```
import pandas as pd

data = pd.Series([10, 20, 15, 30, 45, 25], index=['M1', 'M1', 'M2', 'M2',
'M3', 'M3'])
```

1. .groupby() - Ομαδοποίηση

Αν και χρησιμοποιείται συχνότερα στα DataFrames, στο Series χρησιμεύει για να ομαδοποιήσεις τις τιμές με βάση το index ή ένα άλλο Series και να εφαρμόσεις μια πράξη ανά ομάδα.

- **Παράδειγμα:** data.groupby(level=0).mean()
- **Αποτέλεσμα:** Θα σου δώσει τον μέσο όρο για κάθε "M" (M1: 15, M2: 22.5, M3: 35).

2. Οι "Window" Συναρτήσεις (Rolling & Expanding)

Αυτές οι μέθοδοι εξετάζουν τα δεδομένα σε "παράθυρα" χρόνου, κάτι απαραίτητο για να εξομαλύνουμε τις απότομες αλλαγές (θόρυβο).

- **.rolling(window) (Κινητός Μέσος):** Κοιτάζει τις X προηγούμενες τιμές.
    - *Παράδειγμα:* s.rolling(3).mean() -> Υπολογίζει τον μέσο όρο των τελευταίων 3 ημερών για κάθε σημείο.
- **.expanding():** Ξεκινάει από την αρχή και "μεγαλώνει" το παράθυρο μέχρι να συμπεριλάβει όλα τα δεδομένα.
    - *Παράδειγμα:* s.expanding().sum() -> Στο 4ο στοιχείο, θα έχει το άθροισμα των στοιχείων 1 έως 4.
- **.ewm(span) (Exponential Weighted Moving):** Δίνει μεγαλύτερη βαρύτητα στις **πρόσφατες** τιμές και μικρότερη στις παλιές. Χρησιμοποιείται κατά κόρον στα χρηματιστήρια.

3. .aggregate() & .transform() - Μαζική Επεξεργασία

- **.aggregate() (ή .agg()):** Σου επιτρέπει να ζητήσεις πολλά στατιστικά μαζί.
    - *Παράδειγμα:* data.agg(['min', 'max', 'std']).

- **.transform()**: Εφαρμόζει μια συνάρτηση αλλά **διατηρεί το αρχικό σχήμα** (length) του Series. Είναι εξαιρετικά χρήσιμο για κανονικοποίηση (normalization).
    - *Παράδειγμα:* data.transform(lambda x: (x - x.mean()) / x.std()).

```
import pandas as pd
import numpy as np

# Δημιουργία DataFrame με δεδομένα για groupby
np.random.seed(42)
dates = pd.date_range('2023-01-01', periods=10, freq='D')
categories = ['A', 'B', 'A', 'B', 'C', 'A', 'C', 'C', 'B', 'A']
values = np.random.randint(10, 50, size=10)

df = pd.DataFrame({'date': dates, 'category': categories, 'value': values})
print("Αρχικό DataFrame:")
print(df)
print("\n" + "="*70 + "\n")

# Δημιουργία Series για rolling/expanding/ewm
s = pd.Series([10, 20, 15, 30, 25, 40, 35, 50, 45, 60], index=dates,
name='values')
print("Αρχικό Series για rolling/expanding/ewm:")
print(s)
print("\n" + "="*70 + "\n")

# 1. .groupby() - Ομαδοποίηση
# Εφαρμόζεται σε DataFrame ή Series, εδώ στο DataFrame για να δούμε
ομαδοποίηση
grouped = df.groupby('category')['value'].mean()
print("1. .groupby('category')['value'].mean():")
print(grouped)
print("\nΕναλλακτικά, groupby σε Series με άλλο Series ως κλειδί:")
s_grouped = s.groupby(categories).mean()
print(s_grouped)
print("-"*50)

# 2. .rolling(window) - Κινητός μέσος
rolling_mean = s.rolling(window=3).mean()
print("2. .rolling(3).mean():")
print(rolling_mean)
print("-"*50)

# 3. .expanding() - Επεκτεινόμενος (συνήθως άθροισμα, μέσος κλπ.)
expanding_sum = s.expanding().sum()
print("3. .expanding().sum():")
print(expanding_sum)
print("-"*50)

# 4. .ewm(span) - Εκθετικά σταθμισμένος μέσος
ewm_mean = s.ewm(span=3).mean()
print("4. .ewm(span=3).mean():")
print(ewm_mean)
print("-"*50)

# 5. .aggregate(func) - Συναθροίσεις (πολλαπλές συναρτήσεις)
agg_result = s.aggregate(['mean', 'std', 'min', 'max'])
print("5. .aggregate(['mean', 'std', 'min', 'max']):")
print(agg_result)
```

```
print("\nΜε ομαδοποίηση και aggregate:")
grouped_agg = df.groupby('category')['value'].agg(['mean', 'sum', 'count'])
print(grouped_agg)
print("-"*50)

# 6. .transform(func) - Μετασχηματισμός (επιστρέφει Series ίδιου μήκους)
# Παράδειγμα: αφαίρεση του μέσου όρου της ομάδας
df['value_normalized'] = df.groupby('category')['value'].transform(lambda
x: x - x.mean())
print("6. .transform() - Κανονικοποίηση εντός ομάδας (αφαίρεση μέσου όρου
ομάδας):")
print(df[['category', 'value', 'value_normalized']])
print("\nΜε transform για τετραγωνική ρίζα (στο Series s):")
sqrt_transformed = s.transform(np.sqrt)
print(sqrt_transformed)
print("-"*50)

print("\n" + "="*70 + "\n")
print("Σημείωση: Οι .rolling(), .expanding(), .ewm() εφαρμόζονται απευθείας
σε Series/DataFrame.")
print("Οι .aggregate() και .transform() μπορούν να συνδυαστούν με groupby
για πιο σύνθετες αναλύσεις.")
```

Κ. 3.3.15: Παράδειγμα των Μεθόδων Προχωρημένης Ανάλυσης και Παραθύρων

3.3.16. Εισαγωγή & Εξαγωγή

Μέθοδος	Περιγραφή	Μορφή
.to_list()	Μετατροπή σε λίστα	List
.to_dict()	Μετατροπή σε λεξικό	Dict
.to_frame()	Μετατροπή σε DataFrame	DataFrame
.to_csv(path)	Εξαγωγή CSV	.csv
.to_json()	Εξαγωγή JSON	JSON string
.to_string()	Μετατροπή σε string	String
.to_numpy()	Μετατροπή σε numpy array	ndarray

Πίνακας 3.3.18: Μέθοδοι Εξαγωγής και Μετατροπής (Exporting & Conversion)

Αυτές είναι οι μέθοδοι που χρησιμοποιείς όταν τελειώσεις την ανάλυσή σου και θέλεις να στείλεις τα δεδομένα σου κάπου αλλού: σε μια άλλη βιβλιοθήκη (όπως η NumPy), σε έναν συνάδελφο (ως CSV) ή σε μια εφαρμογή (ως JSON).

Ας δούμε πώς το Series μας «αλλάζει μορφές»:

```
import pandas as pd

s = pd.Series([10, 20, 30], index=['A', 'B', 'C'], name="Data")
```

1. Μετατροπή σε Δομές της Python & NumPy

Όταν θέλεις να βγεις από το οικοσύστημα των Pandas:

- **.to_list()**: Μετατρέπει μόνο τις τιμές σε μια κλασική λίστα Python [10, 20, 30].
- **.to_dict()**: Δημιουργεί ένα λεξικό όπου τα keys είναι το index και values οι τιμές: {'A': 10, 'B': 20, 'C': 30}.
- **.to_numpy()**: Μετατρέπει το Series σε έναν πίνακα NumPy (ndarray). Είναι ο πιο γρήγορος τρόπος για να περάσεις δεδομένα σε βιβλιοθήκες όπως η Scikit-Learn για Machine Learning.

2. Αναβάθμιση σε DataFrame

- **.to_frame()**: Μετατρέπει το Series (που είναι μια στήλη) σε DataFrame (πίνακα).
    - *Γιατί βοηθάει:* Πολλές μέθοδοι των Pandas δουλεύουν μόνο σε DataFrames. Αν του δώσεις όνομα, π.χ. .to_frame(name='Sales'), αυτό θα είναι το όνομα της στήλης στον πίνακα.

3. Αποθήκευση & Ανταλλαγή Δεδομένων (I/O)

- **.to_csv('file.csv')**: Δημιουργεί ένα αρχείο κειμένου που ανοίγει στο Excel. Είναι ο standard τρόπος ανταλλαγής δεδομένων.
- **.to_json()**: Μετατρέπει τα δεδομένα σε μορφή κειμένου JSON, που είναι το "standard" για το Web και τα APIs.
- **.to_string()**: Σου επιστρέφει μια καλαίσθητη αναπαράσταση του Series σε απλό κείμενο, χρήσιμο για logs ή εκτυπώσεις σε κονσόλα.

```
import pandas as pd
import numpy as np
import io  # για προσομοίωση εγγραφής CSV σε string buffer

# Δημιουργία ενός Series με διάφορες τιμές
data = [10, 25, 30, 45, 12, 8, 33, 27, 50, 18]
index_labels = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
s = pd.Series(data, index=index_labels, name='Δεδομένα')

print("Αρχικό Series:")
print(s)
print("\n" + "="*60 + "\n")

# 1. .to_list() - Μετατροπή σε λίστα Python
to_list_result = s.to_list()
print("1. .to_list():")
print(to_list_result)
print("Τύπος:", type(to_list_result))
print("-"*40)

# 2. .to_dict() - Μετατροπή σε λεξικό (index -> τιμή)
to_dict_result = s.to_dict()
print("2. .to_dict():")
print(to_dict_result)
```

```
print("Τύπος:", type(to_dict_result))
print("-"*40)

# 3. .to_frame() - Μετατροπή σε DataFrame
to_frame_result = s.to_frame()
print("3. .to_frame():")
print(to_frame_result)
print("Τύπος:", type(to_frame_result))
print("-"*40)

# 4. .to_csv() - Εξαγωγή σε CSV (εδώ σε string buffer)
# Δημιουργία buffer για να λάβουμε το CSV ως string χωρίς αποθήκευση σε αρχείο
csv_buffer = io.StringIO()
s.to_csv(csv_buffer)  # γράφει στον buffer
csv_string = csv_buffer.getvalue()
print("4. .to_csv() (ως string):")
print(csv_string)
print("-"*40)

# Εναλλακτικά, με παραμέτρους (π.χ. χωρίς header)
csv_buffer2 = io.StringIO()
s.to_csv(csv_buffer2, header=False)  # χωρίς γραμμή ονόματος
print("   .to_csv(header=False):")
print(csv_buffer2.getvalue())
print("-"*40)

# 5. .to_json() - Εξαγωγή σε JSON string
to_json_result = s.to_json()  # προεπιλογή orient='index'
print("5. .to_json() (orient='index'):")
print(to_json_result)
print("\n   .to_json(orient='split'):")
print(s.to_json(orient='split'))
print("-"*40)

# 6. .to_string() - Αναπαράσταση ως string (όπως το print)
to_string_result = s.to_string()
print("6. .to_string():")
print(to_string_result)
print("Τύπος:", type(to_string_result))
print("-"*40)

# 7. .to_numpy() - Μετατροπή σε NumPy array
to_numpy_result = s.to_numpy()
print("7. .to_numpy():")
print(to_numpy_result)
print("Τύπος:", type(to_numpy_result))
print("-"*40)

print("\n" + "="*60 + "\n")
print("Σημείωση: Οι μέθοδοι .to_csv() και .to_json() γράφουν σε αρχείο αν δοθεί μονοπάτι, αλλά εδώ χρησιμοποιήσαμε StringIO για να λάβουμε τα δεδομένα ως string.")
```

Κ. 3.3.16: Παράδειγμα για Μεθόδους Εξαγωγής και Μετατροπής (Exporting & Conversion)

3.3.17. Διάφορες Μέθοδοι

Μέθοδος	Περιγραφή	Χρήση
.apply(func)	Εφαρμογή συνάρτησης	s.apply(np.sqrt)
.pipe(func)	Συνέχεια λειτουργιών	s.pipe(func1).pipe(func2)
.update(other)	Ενημέρωση τιμών	s.update(other_series)
.combine(other, func)	Συνδυασμός με άλλη	s.combine(other, max)
.combine_first(other)	Συνδυασμός με προτεραιότητα	-
.memory_usage()	Χρήση μνήμης	Integer
.equals(other)	Έλεγχος ισότητας	Boolean
.all()	Όλες οι τιμές True	Boolean
.any()	Οποιαδήποτε τιμή True	Boolean

Πίνακας 3.3.19: Μέθοδοι για Προχωρημένες Λειτουργίες και τη Διαχείριση Συστήματος

Παρουσιάζονται οι **Προχωρημένες Λειτουργίες και τη Διαχείριση Συστήματος**. Αυτές οι μέθοδοι αφορούν κυρίως τη ροή εργασίας (workflow), τη βελτιστοποίηση και τον συνδυασμό διαφορετικών πηγών δεδομένων.

Ας δούμε πώς λειτουργούν στην πράξη:

1. Εφαρμογή και Αλυσιδωτές Λειτουργίες (Pipelines)

- **.apply(func)**: Χρησιμοποιείται για τον μετασχηματισμό δεδομένων. Εφαρμόζει μια συνάρτηση σε κάθε στοιχείο του Series.
- **.pipe(func)**: Χρησιμοποιείται για να "καθαρίσει" τον κώδικα όταν θέλεις να εφαρμόσεις πολλές συναρτήσεις τη μία μετά την άλλη. Αντί να γράφεις func3(func2(func1(s))), γράφεις s.pipe(func1).pipe(func2).pipe(func3). Αυτό κάνει τον κώδικα πολύ πιο αναγνώσιμο.

2. Συνδυασμός και Ενημέρωση (Merging & Updating)

Εδώ βλέπουμε πώς μπορούμε να "συγχωνεύσουμε" δύο Series:

- **.update(other)**: Τροποποιεί το αρχικό Series **στη θέση του (in-place)** χρησιμοποιώντας τιμές από ένα άλλο Series. Αν υπάρχουν NaN στο other, δεν θα επηρεάσουν το αρχικό Series.
- **.combine(other, func)**: Παίρνει δύο Series και μια συνάρτηση (π.χ. max) και δημιουργεί ένα νέο Series επιλέγοντας την τιμή που ορίζει η συνάρτηση για κάθε θέση.
- **.combine_first(other)**: Η μέθοδος "μπαλώματος". Αν το πρώτο Series έχει NaN, παίρνει την τιμή από το δεύτερο. Είναι ιδανικό για να γεμίζεις κενά από μια εναλλακτική πηγή.

3. Έλεγχοι και Απόδοση (Validation & Performance)

- **.memory_usage()**: Σου λέει πόσα bytes καταλαμβάνει το Series στη μνήμη RAM. Πολύ σημαντικό όταν δουλεύεις με μεγάλα δεδομένα (Big Data) για να ξέρεις αν πρέπει να αλλάξεις τους τύπους δεδομένων (π.χ. σε category ή int8).
- **.all() / .any()**:
    - Το .all() επιστρέφει True μόνο αν **όλα** τα στοιχεία είναι True.
    - Το .any() επιστρέφει True αν **τουλάχιστον ένα** στοιχείο είναι True.
    - *Παράδειγμα:* (s > 0).all() ελέγχει αν όλες οι τιμές είναι θετικές.

```
import pandas as pd
import numpy as np

# Δημιουργία αρχικών Series
s = pd.Series([1, 4, 9, 16, 25], index=['a', 'b', 'c', 'd', 'e'],
name='original')
s2 = pd.Series([3, 6, 9, 12, 15], index=['a', 'b', 'c', 'd', 'e'],
name='other')
s_with_nan = pd.Series([1, np.nan, 3, np.nan, 5], index=['a', 'b', 'c',
'd', 'e'], name='with_nan')
s_fill = pd.Series([10, 20, 30, 40, 50], index=['a', 'b', 'c', 'd', 'e'],
name='fill_source')

print("Αρχικό Series s:")
print(s)
print("\nSeries s2:")
print(s2)
print("\nSeries with NaN (s_with_nan):")
print(s_with_nan)
print("\nSeries s_fill (για combine_first):")
print(s_fill)
print("\n" + "="*60 + "\n")

# 1. .apply(func) - Εφαρμογή συνάρτησης σε κάθε στοιχείο
s_sqrt = s.apply(np.sqrt)
print("1. .apply(np.sqrt):")
print(s_sqrt)
print("-"*50)

# 2. .pipe(func) - Αλυσιδωτή εφαρμογή συναρτήσεων
def add_ten(series):
    return series + 10

def multiply_two(series):
    return series * 2

# Εφαρμογή pipe: πρώτα add_ten, μετά multiply_two
s_piped = s.pipe(add_ten).pipe(multiply_two)
print("2. .pipe(add_ten).pipe(multiply_two):")
print(s_piped)
print("-"*50)
```

```
# 3. .update(other) - Ενημέρωση τιμών in-place (τροποποιεί το αρχικό
Series)
print("3. .update() - Πριν την ενημέρωση:")
print(s)
s_update_source = pd.Series([99, 100], index=['b', 'd'])  # θα ενημερώσει
μόνο τα index b και d
s.update(s_update_source)
print("Μετά την ενημέρωση με s.update(s_update_source):")
print(s)
print("-"*50)

# Επαναφορά του s για τις επόμενες μεθόδους (για να μην μείνει
τροποποιημένο)
s = pd.Series([1, 4, 9, 16, 25], index=['a', 'b', 'c', 'd', 'e'])

# 4. .combine(other, func) - Συνδυασμός με άλλη Series μέσω συνάρτησης
# Π.χ. παίρνουμε το max κάθε ζεύγους
s_combined_max = s.combine(s2, max)
print("4. .combine(s2, max):")
print(s_combined_max)
# Συνδυασμός με min
s_combined_min = s.combine(s2, min)
print("   .combine(s2, min):")
print(s_combined_min)
print("-"*50)

# 5. .combine_first(other) - Συμπληρώνει τις NaN τιμές από άλλη Series
s_filled = s_with_nan.combine_first(s_fill)
print("5. .combine_first(s_fill) (συμπλήρωση NaN από s_fill):")
print(s_filled)
print("-"*50)

# 6. .memory_usage() - Μνήμη που καταλαμβάνει το Series (σε bytes)
mem_usage = s.memory_usage()
print("6. .memory_usage():", mem_usage, "bytes")
# Με index=True (προεπιλογή) περιλαμβάνει και το index. Αν deep=True,
υπολογίζει και τη μνήμη των αντικειμένων.
mem_usage_deep = s.memory_usage(deep=True)
print("   .memory_usage(deep=True):", mem_usage_deep, "bytes (με deep για
object types)")
print("-"*50)

# 7. .equals(other) - Έλεγχος αν δύο Series είναι ίδια (τιμές και index)
s_equal = pd.Series([1, 4, 9, 16, 25], index=['a', 'b', 'c', 'd', 'e'])
print("7. .equals(s_equal):", s.equals(s_equal))
s_not_equal = pd.Series([1, 4, 9, 16, 26], index=['a', 'b', 'c', 'd', 'e'])
print("   .equals(s_not_equal):", s.equals(s_not_equal))
print("-"*50)

# 8. .all() - Επιστρέφει True αν όλες οι τιμές είναι True (ή μη μηδενικές)
# Δημιουργούμε boolean Series
bool_series = pd.Series([True, True, True, True])
print("8. Boolean Series [True, True, True, True] .all():",
bool_series.all())
bool_series_false = pd.Series([True, False, True])
print("   Boolean Series [True, False, True] .all():",
bool_series_false.all())
# Σε αριθμητικό Series, το 0 θεωρείται False, τα άλλα True
num_series = pd.Series([1, 2, 3, 4])
print("   Αριθμητικό Series [1,2,3,4] .all():", num_series.all())
num_series_zero = pd.Series([1, 2, 0, 4])
```

```
print("   Αριθμητικό Series [1,2,0,4] .all():", num_series_zero.all())
print("-"*50)

# 9. .any() - Επιστρέφει True αν τουλάχιστον μία τιμή είναι True
print("9. Boolean Series [False, False, False] .any():", pd.Series([False,
False, False]).any())
print("   Boolean Series [False, True, False] .any():", pd.Series([False,
True, False]).any())
print("   Αριθμητικό Series [0,0,0] .any():", pd.Series([0, 0, 0]).any())
print("   Αριθμητικό Series [0,1,0] .any():", pd.Series([0, 1, 0]).any())
print("-"*50)
```

Κ. 3.3.17: Παράδειγμα Μεθόδων για Προχωρημένες Λειτουργίες και τη Διαχείριση Συστήματος

3.3.18. Πίνακας Κοινών Παραμέτρων για Σειρες

Παράμετρος	Περιγραφή	Τιμές	Εφαρμογή
axis	Άξονας λειτουργίας	0/'index'	s.sum(axis=0)
skipna	Παράλειψη NaN	True/False	s.sum(skipna=True)
level	Επίπεδο για πολυ-index	Integer	s.sum(level=0)
inplace	Τροποποίηση inplace	True/False	s.fillna(0, inplace=True)
ascending	Φθίνουσα/αύξουσα	True/False	s.sort_values(ascending=True)

Πίνακας 3.3.20: Παράμετροι των Μεθόδων

Ολοκληρώνουμε την «ανατομία» των Pandas Series εξετάζοντας τις **βασικές παραμέτρους** που ελέγχουν πώς συμπεριφέρονται οι μέθοδοι. Αν οι μέθοδοι είναι τα εργαλεία, οι παράμετροι είναι οι «ρυθμίσεις» τους που καθορίζουν την ακρίβεια και το αποτέλεσμα της εργασίας μας.

1. axis – Ο Άξονας

Στα **Series**, ο άξονας είναι σχεδόν πάντα 0 (ή 'index'), αφού έχουμε μόνο μία στήλη δεδομένων. Η παράμετρος αυτή γίνεται εξαιρετικά σημαντική στα DataFrames, όπου επιλέγεις αν μια πράξη (π.χ. το άθροισμα) θα γίνει κατά μήκος των γραμμών ή των στηλών.

2. skipna – Η Διαχείριση των Κενών

Αυτή η παράμετρος καθορίζει αν οι στατιστικοί υπολογισμοί θα αγνοούν τα NaN.

- **skipna=True (Default):** Το Pandas υπολογίζει το αποτέλεσμα χρησιμοποιώντας μόνο τις υπάρχουσες τιμές.
- **skipna=False:** Αν υπάρχει έστω και ένα NaN, το αποτέλεσμα ολόκληρης της πράξης θα είναι NaN. Είναι χρήσιμο όταν θέλεις να είσαι σίγουρος ότι δεν λείπει ούτε ένα δεδομένο.

3. inplace – Μόνιμη Αλλαγή ή Αντίγραφο;

Αυτή είναι μια από τις πιο κρίσιμες παραμέτρους για τη διαχείριση της μνήμης και τη ροή του κώδικα.

- **inplace=False (Default):** Η μέθοδος επιστρέφει ένα **νέο** Series. Το αρχικό μένει ανέπαφο.
- **inplace=True:** Η αλλαγή εφαρμόζεται **απευθείας** στο αρχικό Series και η μέθοδος δεν επιστρέφει τίποτα (None).

Συμβουλή: Η χρήση του inplace=True τείνει να καταργηθεί στις νεότερες εκδόσεις των Pandas. Προτιμάται η εκχώρηση: s = s.fillna(0).

4. ascending – Η Κατεύθυνση της Ταξινόμησης

Χρησιμοποιείται στις μεθόδους sort_values και sort_index.

- **True:** Από το μικρότερο στο μεγαλύτερο (Α-Ω).
- **False:** Από το μεγαλύτερο στο μικρότερο (Ω-Α).
- **5. level – Πολυ-επίπεδα (Multi-index)**

Όταν ένα Series έχει παραπάνω από ένα index (ιεραρχικό indexing), η παράμετρος level σου επιτρέπει να ορίσεις σε ποιο επίπεδο θα γίνει η πράξη. Για παράδειγμα, αν έχεις πωλήσεις ανά «Έτος» και «Μήνα», μπορείς να ζητήσεις το άθροισμα μόνο στο επίπεδο του «Έτους».

```
import pandas as pd
import numpy as np

# Δημιουργία Series με MultiIndex
index = pd.MultiIndex.from_tuples([('A', 1), ('A', 2), ('B', 1), ('B', 2),
('B', 3)],
                                  names=['group', 'sub'])
data = [10, 20, 30, 40, 50]
s_multi = pd.Series(data, index=index, name='values')

print("Series με MultiIndex:")
print(s_multi)
print("\n" + "="*60 + "\n")

# 1. axis - Άξονας λειτουργίας (μόνο axis=0 για Series)
print("1. axis (μόνο axis=0):")
print("s_multi.sum(axis=0):", s_multi.sum(axis=0))
print("Ισοδύναμα χωρίς axis:", s_multi.sum())
print("-"*40)

# 2. skipna - Παράλειψη NaN
s_with_nan = pd.Series([1, 2, np.nan, 4, 5])
print("2. skipna - Series με NaN:")
print(s_with_nan)
print("sum() skipna=True (default):", s_with_nan.sum())
print("sum() skipna=False:", s_with_nan.sum(skipna=False))
print("-"*40)

# 3. level - Συνάθροιση ανά επίπεδο (μέσω groupby)
print("3. level - Συνάθροιση ανά επίπεδο (groupby):")
print("s_multi.groupby(level='group').sum():")
print(s_multi.groupby(level='group').sum())
```

```
print("\ns_multi.groupby(level=0).mean():")
print(s_multi.groupby(level=0).mean())
print("-"*40)

# 4. inplace - Τροποποίηση in-place
s_inplace = pd.Series([1, np.nan, 3, np.nan, 5])
print("4. inplace - Πριν από fillna:")
print(s_inplace)
s_inplace.fillna(0, inplace=True)
print("Μετά από fillna(0, inplace=True):")
print(s_inplace)
print("-"*40)

# 5. ascending - Ταξινόμηση αύξουσα/φθίνουσα
s_unsorted = pd.Series([3, 1, 4, 2, 5], index=['c', 'a', 'd', 'b', 'e'])
print("5. ascending - sort_values(ascending=True):")
print(s_unsorted.sort_values(ascending=True))
print("\n   sort_values(ascending=False):")
print(s_unsorted.sort_values(ascending=False))
print("-"*40)
```

Κ. 3.3.18: Παράδειγμα για Παράμετροι των Μεθόδων

3.3.19. Ολοκληρωμένο Παράδειγμα στις Σειρές: Διαχείριση Προϊόντων

Το πρόγραμμα διαχειρίζεται τα προϊόντα ενός καταστήματος. Τα στοιχεία που κρατάμε στα δεδομένα είναι όνομα προϊόντος, ποσότητα και τιμή.

Οι λειτουργίες που υλοποιεί το πρόγραμμα είναι:

ΣΥΣΤΗΜΑ ΔΙΑΧΕΙΡΙΣΗΣ ΠΡΟΪΟΝΤΩΝ (Series)

1. Προσθήκη προϊόντος
2. Διόρθωση προϊόντος
3. Διαγραφή προϊόντος
4. Κατάσταση προϊόντων
5. Συνολική αξία προϊόντων
6. Ταξινόμηση προϊόντων
7. Αποθήκευση σε αρχείο
8. Φόρτωση από αρχείο
9. Έξοδος

```
import pandas as pd
import os

class ProductManager:
    def __init__(self):
        # Δημιουργία κενών Series για κάθε πεδίο
```

```
        self.names = pd.Series([], dtype='object', name='Όνομα')
        self.quantities = pd.Series([], dtype='int64', name='Ποσότητα')
        self.prices = pd.Series([], dtype='float64', name='Τιμή')

    def display_menu(self):
        """Εμφάνιση μενού επιλογών"""
        print("\n" + "="*50)
        print("ΣΥΣΤΗΜΑ ΔΙΑΧΕΙΡΙΣΗΣ ΠΡΟΪΟΝΤΩΝ (Series)")
        print("="*50)
        print("1. Προσθήκη προϊόντος")
        print("2. Διόρθωση προϊόντος")
        print("3. Διαγραφή προϊόντος")
        print("4. Κατάσταση προϊόντων")
        print("5. Συνολική αξία προϊόντων")
        print("6. Ταξινόμηση προϊόντων")
        print("7. Αποθήκευση σε αρχείο")
        print("8. Φόρτωση από αρχείο")
        print("9. Έξοδος")
        print("="*50)

    def add_product(self):
        """Προσθήκη νέου προϊόντος"""
        print("\n--- ΠΡΟΣΘΗΚΗ ΝΕΟΥ ΠΡΟΪΟΝΤΟΣ ---")

        name = input("Δώστε όνομα προϊόντος: ").strip()

        # Έλεγχος αν υπάρχει ήδη το προϊόν
        if not self.names.empty and name in self.names.values:
            print(f"Το προϊόν '{name}' υπάρχει ήδη στη βάση δεδομένων!")
            return

        # Εισαγωγή ποσότητας
        while True:
            try:
                quantity = int(input("Δώστε ποσότητα: "))
                if quantity < 0:
                    print("Η ποσότητα πρέπει να είναι θετικός αριθμός!")
                    continue
                break
            except ValueError:
                print("Παρακαλώ εισάγετε έγκυρο αριθμό!")

        # Εισαγωγή τιμής
        while True:
            try:
                price = float(input("Δώστε τιμή: "))
                if price < 0:
                    print("Η τιμή πρέπει να είναι θετικός αριθμός!")
                    continue
                break
            except ValueError:
                print("Παρακαλώ εισάγετε έγκυρο αριθμό!")

        # Προσθήκη στα Series
        self.names = pd.concat([self.names, pd.Series([name])],
ignore_index=True)
        self.quantities = pd.concat([self.quantities,
pd.Series([quantity])], ignore_index=True)
        self.prices = pd.concat([self.prices, pd.Series([price])],
ignore_index=True)
```

```
        print(f"Το προϊόν '{name}' προστέθηκε επιτυχώς!")

    def edit_product(self):
        """Διόρθωση υπάρχοντος προϊόντος"""
        if self.names.empty:
            print("Δεν υπάρχουν προϊόντα στη βάση δεδομένων!")
            return

        print("\n--- ΔΙΟΡΘΩΣΗ ΠΡΟΪΟΝΤΟΣ ---")
        self.show_products()

        product_name = input("Δώστε όνομα προϊόντος για διόρθωση:
").strip()

        if product_name not in self.names.values:
            print(f"Το προϊόν '{product_name}' δεν βρέθηκε!")
            return

        # Εύρεση του δείκτη του προϊόντος
        product_index = self.names[self.names == product_name].index[0]

        print(f"\nΤρέχοντα στοιχεία για '{product_name}':")
        print(f"Ποσότητα: {self.quantities.iloc[product_index]}")
        print(f"Τιμή: {self.prices.iloc[product_index]:.2f} €")

        # Επιλογή πεδίου για διόρθωση
        print("\nΤι θέλετε να διορθώσετε;")
        print("1. Ποσότητα")
        print("2. Τιμή")
        print("3. Και τα δύο")

        choice = input("Επιλογή (1-3): ").strip()

        if choice == '1' or choice == '3':
            while True:
                try:
                    new_quantity = int(input("Νέα ποσότητα: "))
                    if new_quantity < 0:
                        print("Η ποσότητα πρέπει να είναι θετικός
αριθμός!")
                        continue
                    self.quantities.iloc[product_index] = new_quantity
                    break
                except ValueError:
                    print("Παρακαλώ εισάγετε έγκυρο αριθμό!")

        if choice == '2' or choice == '3':
            while True:
                try:
                    new_price = float(input("Νέα τιμή: "))
                    if new_price < 0:
                        print("Η τιμή πρέπει να είναι θετικός αριθμός!")
                        continue
                    self.prices.iloc[product_index] = new_price
                    break
                except ValueError:
                    print("Παρακαλώ εισάγετε έγκυρο αριθμό!")

        print(f"Το προϊόν '{product_name}' ενημερώθηκε επιτυχώς!")

    def delete_product(self):
```

```
        """Διαγραφή προϊόντος"""
        if self.names.empty:
            print("Δεν υπάρχουν προϊόντα στη βάση δεδομένων!")
            return

        print("\n--- ΔΙΑΓΡΑΦΗ ΠΡΟΪΟΝΤΟΣ ---")
        self.show_products()

        product_name = input("Δώστε όνομα προϊόντος για διαγραφή:
").strip()

        if product_name not in self.names.values:
            print(f"Το προϊόν '{product_name}' δεν βρέθηκε!")
            return

        # Εύρεση του δείκτη του προϊόντος
        product_index = self.names[self.names == product_name].index[0]

        # Επιβεβαίωση διαγραφής
        confirm = input(f"Είστε σίγουρος ότι θέλετε να διαγράψετε το
'{product_name}'; (ναι/όχι): ").strip().lower()

        if confirm == 'ναι' or confirm == 'ν':
            # Διαγραφή από όλα τα Series
            self.names =
self.names.drop(product_index).reset_index(drop=True)
            self.quantities =
self.quantities.drop(product_index).reset_index(drop=True)
            self.prices =
self.prices.drop(product_index).reset_index(drop=True)

            print(f"Το προϊόν '{product_name}' διαγράφηκε επιτυχώς!")
        else:
            print("Η διαγραφή ακυρώθηκε.")

    def show_products(self):
        """Εμφάνιση όλων των προϊόντων"""
        if self.names.empty:
            print("Δεν υπάρχουν προϊόντα στη βάση δεδομένων!")
            return

        print("\n--- ΚΑΤΑΣΤΑΣΗ ΠΡΟΪΟΝΤΩΝ ---")
        print(f"{'Α/Α':<4} {'Όνομα':<20} {'Ποσότητα':<10} {'Τιμή':<10}
{'Σύνολο':<12}")
        print("-" * 60)

        for i in range(len(self.names)):
            total = self.quantities.iloc[i] * self.prices.iloc[i]
            print(f"{i+1:<4} {self.names.iloc[i]:<20}
{self.quantities.iloc[i]:<10} {self.prices.iloc[i]:<10.2f} {total:<12.2f}")

    def total_value(self):
        """Υπολογισμός συνολικής αξίας προϊόντων"""
        if self.names.empty:
            print("Δεν υπάρχουν προϊόντα στη βάση δεδομένων!")
            return

        total_value = (self.quantities * self.prices).sum()
        print(f"\n--- ΣΥΝΟΛΙΚΗ ΑΞΙΑ ΠΡΟΪΟΝΤΩΝ ---")
        print(f"Συνολική αξία: {total_value:.2f} €")
```

```
        # Αναλυτική παρουσίαση
        print("\nΑναλυτικά:")
        for i in range(len(self.names)):
            product_total = self.quantities.iloc[i] * self.prices.iloc[i]
            print(f"{self.names.iloc[i]}: {product_total:.2f} €")

    def sort_products(self):
        """Ταξινόμηση προϊόντων"""
        if self.names.empty:
            print("Δεν υπάρχουν προϊόντα στη βάση δεδομένων!")
            return

        print("\n--- ΤΑΞΙΝΟΜΗΣΗ ΠΡΟΪΟΝΤΩΝ ---")
        print("1. Ταξινόμηση κατά όνομα (αύξουσα)")
        print("2. Ταξινόμηση κατά όνομα (φθίνουσα)")
        print("3. Ταξινόμηση κατά ποσότητα (αύξουσα)")
        print("4. Ταξινόμηση κατά ποσότητα (φθίνουσα)")
        print("5. Ταξινόμηση κατά τιμή (αύξουσα)")
        print("6. Ταξινόμηση κατά τιμή (φθίνουσα)")

        choice = input("Επιλογή (1-6): ").strip()

        # Δημιουργία προσωρινού DataFrame για ταξινόμηση
        temp_df = pd.DataFrame({
            'Όνομα': self.names,
            'Ποσότητα': self.quantities,
            'Τιμή': self.prices
        })

        sorting_options = {
            '1': ('Όνομα', True),
            '2': ('Όνομα', False),
            '3': ('Ποσότητα', True),
            '4': ('Ποσότητα', False),
            '5': ('Τιμή', True),
            '6': ('Τιμή', False)
        }

        if choice in sorting_options:
            column, ascending = sorting_options[choice]
            temp_df = temp_df.sort_values(by=column,
ascending=ascending).reset_index(drop=True)

            # Ενημέρωση των Series με τα ταξινομημένα δεδομένα
            self.names = temp_df['Όνομα']
            self.quantities = temp_df['Ποσότητα']
            self.prices = temp_df['Τιμή']

            print(f"Τα προϊόντα ταξινομήθηκαν κατά {column} ({'αύξουσα' if
ascending else 'φθίνουσα'} σειρά)!")
            self.show_products()
        else:
            print("Μη έγκυρη επιλογή!")

    def save_to_file(self):
        """Αποθήκευση δεδομένων σε αρχείο κειμένου"""
        if self.names.empty:
            print("Δεν υπάρχουν δεδομένα για αποθήκευση!")
            return
```

```
        filename = input("Δώστε όνομα αρχείου για αποθήκευση (π.χ.
products.txt): ").strip()

        if not filename:
            filename = "products.txt"

        try:
            # Δημιουργία DataFrame προσωρινά για αποθήκευση
            temp_df = pd.DataFrame({
                'Όνομα': self.names,
                'Ποσότητα': self.quantities,
                'Τιμή': self.prices
            })

            # Αποθήκευση με κωδικοποίηση UTF-8 για ελληνικούς χαρακτήρες
            temp_df.to_csv(filename, index=False, encoding='utf-8')
            print(f"Τα δεδομένα αποθηκεύτηκαν επιτυχώς στο αρχείο
'{filename}'!")
        except Exception as e:
            print(f"Σφάλμα κατά την αποθήκευση: {e}")

    def load_from_file(self):
        """Φόρτωση δεδομένων από αρχείο κειμένου"""
        filename = input("Δώστε όνομα αρχείου για φόρτωση: ").strip()

        if not os.path.exists(filename):
            print(f"Το αρχείο '{filename}' δεν βρέθηκε!")
            return

        try:
            # Φόρτωση με κωδικοποίηση UTF-8 για ελληνικούς χαρακτήρες
            temp_df = pd.read_csv(filename, encoding='utf-8')

            # Έλεγχος ότι το αρχείο έχει τις σωστές στήλες
            expected_columns = ['Όνομα', 'Ποσότητα', 'Τιμή']
            if all(col in temp_df.columns for col in expected_columns):
                self.names = temp_df['Όνομα']
                self.quantities = temp_df['Ποσότητα'].astype('int64')
                self.prices = temp_df['Τιμή'].astype('float64')

                print(f"Τα δεδομένα φορτώθηκαν επιτυχώς από το αρχείο
'{filename}'!")
                print(f"Φορτώθηκαν {len(self.names)} προϊόντα.")
            else:
                print("Το αρχείο δεν έχει τη σωστή μορφή!")

        except Exception as e:
            print(f"Σφάλμα κατά τη φόρτωση: {e}")

    def get_products_count(self):
        """Επιστρέφει τον αριθμό των προϊόντων"""
        return len(self.names)

    def run(self):
        """Κύρια λειτουργία του προγράμματος"""
        print("Καλώς ήρθατε στο Σύστημα Διαχείρισης Προϊόντων (Series
Edition)!")

        while True:
            self.display_menu()
            choice = input("Επιλέξτε λειτουργία (1-9): ").strip()
```

```
            if choice == '1':
                self.add_product()
            elif choice == '2':
                self.edit_product()
            elif choice == '3':
                self.delete_product()
            elif choice == '4':
                self.show_products()
            elif choice == '5':
                self.total_value()
            elif choice == '6':
                self.sort_products()
            elif choice == '7':
                self.save_to_file()
            elif choice == '8':
                self.load_from_file()
            elif choice == '9':
                print("Ευχαριστούμε που χρησιμοποιήσατε το πρόγραμμα!
Αντίο!")
                break
            else:
                print("Μη έγκυρη επιλογή! Παρακαλώ επιλέξτε από 1 έως 9.")

# Εκτέλεση του προγράμματος
if __name__ == "__main__":
    manager = ProductManager()
    manager.run()
```

3.4. Πλαίσια Δεδομένων (Data Frames)

Αναλογία με Άλλες Δομές

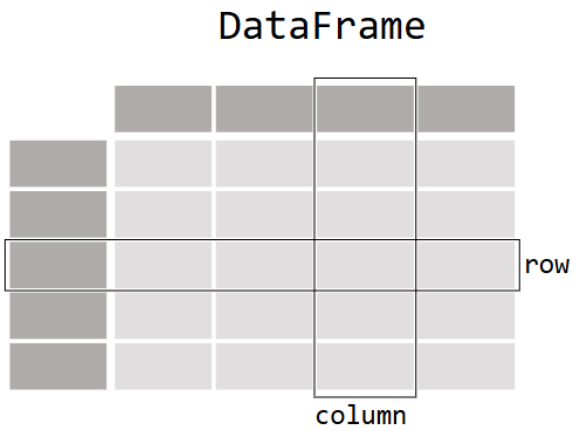
Δομή	Διαστάσεις	Ομογένεια	Ιδιότητες
DataFrame	2D	Όχι (μixed types)	Ετικέτες, ισχυρές λειτουργίες
NumPy Array	N-D	Ναι (ίδιος τύπος)	Γρήγορη αριθμητική
Python List	1D	Ναι	Απλή, ευέλικτη
Dictionary	1D (κλειδιά)	Όχι	Ζεύγη κλειδιού-τιμής

Πίνακας 3.4.1: Σύγκριση των Πλαισίων Δεδομένων (DataFrames) με τις άλλες Δομές

Το DataFrame είναι **η πιο χρήσιμη δομή για data analysis** στην Python λόγω της ευελιξίας και των πλούσιων λειτουργιών της!

Επίσημη σελίδα τεκμηρίωσης: https://pandas.pydata.org/docs/reference/frame.html

Τον κώδικας της παραγράφου μπορείτε να τον βρείτε εδώ:
https://colab.research.google.com/drive/1Yu8rEIL0V9R5N2kiIZDDK2B0bAcLp-P5?usp=sharing



Εικόνα 3.4.1 – Αναπαράσταση ενός Πλαισίου Δεδομένων

3.4.1. Δημιουργία & Βασικές Πληροφορίες

Μέθοδος	Περιγραφή	Παράδειγμα
Δημιουργία DataFrame		
pd.DataFrame()	Κατασκευάζει ένα DataFrame από διάφορες δομές δεδομένων, όπως λεξικά, λίστες ή NumPy arrays.	df = pd.DataFrame({'Ονομα': ['Αννα', 'Γιώργος'], 'Ηλικία': [25, 30]})
pd.read_csv()	Φορτώνει δεδομένα από ένα αρχείο CSV και τα μετατρέπει σε DataFrame.	df = pd.read_csv('το_αρχείο_μου.csv')
pd.read_excel()	Φορτώνει δεδομένα από ένα αρχείο Excel.	df = pd.read_excel('το_αρχείο_μου.xlsx')
pd.json_normalize()	Μετατρέπει ένθετα δεδομένα JSON σε ένα επίπεδο (flat) DataFrame.	pd.json_normalize(api_response['key'])
Βασικές Ιδιότητες και Πληροφορίες		
df.info()	Εμφανίζει μια συνοπτική περιγραφή του DataFrame, συμπεριλαμβανομένων των ονομάτων στηλών, του αριθμού μη-κενών τιμών και του τύπου δεδομένων (dtype).	df.info() Output: RangeIndex: 4 entries, 0 to 3, Data columns (total 5 columns)...
df.head(n)	Επιστρέφει τις πρώτες n γραμμές του DataFrame (προεπιλογή: 5).	df.head(10) Output: (οι πρώτες 10 γραμμές)

Μέθοδος	Περιγραφή	Παράδειγμα
df.tail(n)	Επιστρέφει τις τελευταίες n γραμμές του DataFrame (προεπιλογή: 5).	df.tail() Output: (οι τελευταίες 5 γραμμές)
df.shape	Επιστρέφει ένα tuple με τον αριθμό των γραμμών και στηλών.	df.shape Output: (100, 5)
df.columns	Επιστρέφει το σύνολο των ετικετών (ονομάτων) στηλών.	df.columns Output: Index(['Ονομα', 'Ηλικία'], dtype='object')
df.index	Επιστρέφει τις ετικέτες των γραμμών (το ευρετήριο).	df.index *Output: RangeIndex(start=0, stop=2, step=1)*
df.dtypes	Επιστρέφει τους τύπους δεδομένων για κάθε στήλη.	df.dtypes Output: Ονομα object, Ηλικία int64
df.shape	Επιστρέφει τις διαστάσεις του DataFrame (αριθμός γραμμών, αριθμός στηλών).	df.shape Output: (100, 5)
df.size	Επιστρέφει τον συνολικό αριθμό στοιχείων στο DataFrame.	df.size Output: 500
df.memory_usage()	Επιστρέφει τη χρήση μνήμης για κάθε στήλη (σε bytes).	df.memory_usage()
df.empty	Ελέγχει αν το DataFrame είναι κενό (επιστρέφει True ή False).	df.empty Output: False
df.sample(n)	Επιστρέφει ένα τυχαίο δείγμα n γραμμών από το DataFrame.	df.sample(3) Output: 3 τυχαίες γραμμές.

Πίνακας 3.4.2: Μέθοδοι για Δημιουργία & Βασικές Πληροφορίες στα DataFrames

Μέθοδοι που αφορούν τη **δημιουργία** και την **επισκόπηση** (inspection) ενός DataFrame. Αυτά είναι τα πρώτα βήματα σε κάθε project: πώς φέρνουμε τα δεδομένα μέσα και πώς καταλαβαίνουμε τι περιέχουν.

1. Εισαγωγή Δεδομένων (Data Ingestion)

Η Pandas είναι εξαιρετικά ευέλικτη στο να "διαβάζει" διαφορετικές πηγές.

- **pd.DataFrame()**: Χρησιμοποιείται κυρίως για χειροκίνητη δημιουργία (π.χ. από ένα dictionary). Είναι χρήσιμο για testing ή όταν φτιάχνεις μικρούς πίνακες παραμέτρων.
- **pd.read_csv() & pd.read_excel()**: Οι "βασιλιάδες" της εισαγωγής. Σημείωσε ότι το read_csv είναι συνήθως πολύ πιο γρήγορο από το read_excel.
- **pd.json_normalize()**: Ένα "κρυφό διαμάντι". Αν δουλεύεις με API που επιστρέφουν περίπλοκα JSON (nested), αυτή η μέθοδος τα "ισιώνει" (flattening) αυτόματα σε στήλες, γλιτώνοντάς σε από πολύπλοκα loops.

2. Γρήγορη Ματιά (Initial Preview)

Μόλις φορτώσεις τα δεδομένα, πρέπει να δεις αν "κάθισαν" σωστά.

- **df.head(n) / df.tail(n)**: Τα χρησιμοποιούμε πάντα για να βεβαιωθούμε ότι οι επικεφαλίδες διαβάστηκαν σωστά και ότι το τέλος του αρχείου δεν έχει περίεργα κενά ή labels.

- **df.sample(n)**: Πολύ σημαντικό! Πολλές φορές τα δεδομένα είναι ταξινομημένα (π.χ. όλοι οι άνδρες στην αρχή, όλες οι γυναίκες στο τέλος). Το head() μπορεί να σε παραπλανήσει. Το sample() σου δίνει μια πιο αντιπροσωπευτική εικόνα του συνόλου.

3. Μεταδεδομένα & Δομή (Metadata)

Εδώ ελέγχουμε την "υγεία" και το μέγεθος του DataFrame.

- **df.info()**: Ίσως η πιο χρήσιμη μέθοδος. Σου λέει αμέσως:
    1. Πόση μνήμη καταναλώνει το DataFrame.
    2. Αν υπάρχουν Missing Values (Nulls).
    3. Τους τύπους δεδομένων (π.χ. αν μια ημερομηνία διαβάστηκε ως "object" αντί για "datetime").
- **df.shape & df.size**: Προσοχή στη διαφορά! Το shape επιστρέφει διαστάσεις (10 γραμμές, 5 στήλες), ενώ το size τον συνολικό αριθμό κελιών (10 * 5 = 50).
- **df.dtypes**: Χρήσιμο όταν θέλεις να απομονώσεις μόνο τις αριθμητικές στήλες για υπολογισμούς.

4. Διαχείριση Μνήμης & Ελέγχου

- **df.memory_usage()**: Αν δουλεύεις με μεγάλα δεδομένα (Big Data), αυτή η μέθοδος σου δείχνει ποιες στήλες "τρώνε" τη μνήμη σου. Συχνά, μετατρέποντας μια στήλη από object σε category, η χρήση μνήμης πέφτει κατακόρυφα.
- **df.empty**: Χρησιμοποιείται κυρίως σε αυτοματισμούς και scripts. Πριν ξεκινήσεις μια βαριά επεξεργασία, ελέγχεις αν το DataFrame έχει όντως δεδομένα για να αποφύγεις errors.

```
import pandas as pd
import numpy as np
import json
import tempfile
import os

# ---------------------------------------------------------------------
# 1. ΔΗΜΙΟΥΡΓΙΑ DataFrame
# ---------------------------------------------------------------------

# 1.1 Δημιουργία DataFrame από λεξικό (dict) με λίστες
print("1.1 Δημιουργία DataFrame από λεξικό")
data_dict = {
    'Name': ['Anna', 'George', 'Maria', 'Nikos'],
    'Age': [25, 30, 28, 35],
    'City': ['Athens', 'Thessaloniki', 'Patras', 'Heraklion']
}
df_from_dict = pd.DataFrame(data_dict)
print(df_from_dict)
print("\n" + "-"*50)

# 1.2 Δημιουργία DataFrame από αρχείο CSV
# Δημιουργούμε ένα προσωρινό αρχείο CSV για επίδειξη
print("1.2 Δημιουργία DataFrame από αρχείο CSV")
```

```
with tempfile.NamedTemporaryFile(mode='w', suffix='.csv', delete=False,
encoding='utf-8') as tmp_csv:
    tmp_csv.write("Name,Age,City\n")
    tmp_csv.write("Eleni,22,Volos\n")
    tmp_csv.write("Dimitris,27,Chania\n")
    tmp_csv.write("Sofia,24,Larisa\n")
    tmp_csv_path = tmp_csv.name

# Φόρτωση του CSV σε DataFrame
df_from_csv = pd.read_csv(tmp_csv_path)
print(df_from_csv)

# Διαγραφή προσωρινού αρχείου
os.unlink(tmp_csv_path)
print("\n" + "-"*50)

# 1.3 Δημιουργία DataFrame από αρχείο Excel
# Απαιτείται η εγκατάσταση της βιβλιοθήκης openpyxl ή xlrd.
# Δημιουργούμε ένα προσωρινό αρχείο Excel για επίδειξη.
print("1.3 Δημιουργία DataFrame από αρχείο Excel")
# Για να αποφύγουμε την πραγματική εγγραφή σε Excel (που απαιτεί επιπλέον
βιβλιοθήκες),
# απλά δείχνουμε την εντολή και σχολιάζουμε. Αν θέλετε να τρέξετε,
εγκαταστήστε openpyxl.
# Προσομοίωση: δημιουργούμε ένα μικρό DataFrame και το αποθηκεύουμε
προσωρινά σε Excel.
# Αν δεν έχετε εγκατεστημένο το openpyxl, η αποθήκευση θα αποτύχει. Σε αυτή
την περίπτωση,
# απλώς εκτυπώνουμε την εντολή χωρίς να την εκτελέσουμε.

try:
    with tempfile.NamedTemporaryFile(suffix='.xlsx', delete=False) as
tmp_xlsx:
        tmp_xlsx_path = tmp_xlsx.name
    # Αποθήκευση του προηγούμενου df_from_dict σε Excel
    df_from_dict.to_excel(tmp_xlsx_path, index=False, engine='openpyxl')
    # Ανάγνωση του Excel
    df_from_excel = pd.read_excel(tmp_xlsx_path, engine='openpyxl')
    print(df_from_excel)
    os.unlink(tmp_xlsx_path)
except ImportError:
    print("Η βιβλιοθήκη openpyxl δεν είναι εγκατεστημένη. Παράδειγμα με
read_excel παραλείπεται.")
    print("Για να το εκτελέσετε, εγκαταστήστε: pip install openpyxl")
print("\n" + "-"*50)

# 1.4 Δημιουργία DataFrame από ένθετο JSON (normalize)
print("1.4 Δημιουργία DataFrame από ένθετο JSON με pd.json_normalize")
# Παράδειγμα ένθετου JSON (σαν απάντηση από API)
nested_json = {
    'status': 'success',
    'data': [
        {'id': 1, 'name': 'Product A', 'details': {'price': 100, 'stock':
50}},
        {'id': 2, 'name': 'Product B', 'details': {'price': 200, 'stock':
30}},
        {'id': 3, 'name': 'Product C', 'details': {'price': 150, 'stock':
0}}
    ]
}
# Εξαγωγή του πεδίου 'data' και μετατροπή σε επίπεδο DataFrame
```

```
df_normalized = pd.json_normalize(nested_json['data'])
print(df_normalized)
print("\n" + "-"*50)

# ----------------------------------------------------------------------
# 2. ΒΑΣΙΚΕΣ ΙΔΙΟΤΗΤΕΣ ΚΑΙ ΠΛΗΡΟΦΟΡΙΕΣ DataFrame
# ----------------------------------------------------------------------
# Δημιουργούμε ένα πιο αντιπροσωπευτικό DataFrame για τα επόμενα
παραδείγματα
np.random.seed(42)  # για αναπαραγωγιμότητα
df_demo = pd.DataFrame({
    'A': np.random.randn(100),
    'B': np.random.randint(0, 100, 100),
    'C': np.random.choice(['X', 'Y', 'Z'], 100),
    'D': pd.date_range('2023-01-01', periods=100, freq='D')
})
# Εισαγωγή μερικών κενών τιμών για πληρότητα
df_demo.loc[5, 'A'] = np.nan
df_demo.loc[10, 'B'] = np.nan
df_demo.loc[15, 'C'] = np.nan

print("2. ΒΑΣΙΚΕΣ ΙΔΙΟΤΗΤΕΣ ΚΑΙ ΠΛΗΡΟΦΟΡΙΕΣ")
print("DataFrame df_demo (πρώτες 5 γραμμές):")
print(df_demo.head())
print("\n" + "-"*50)

# 2.1 df.info() - Συνοπτική περιγραφή
print("2.1 df.info() - Συνοπτική περιγραφή")
df_demo.info()
print("\n" + "-"*50)

# 2.2 df.head(n) - Πρώτες n γραμμές (προεπιλογή 5)
print("2.2 df.head(3) - Πρώτες 3 γραμμές")
print(df_demo.head(3))
print("\n" + "-"*50)

# 2.3 df.tail(n) - Τελευταίες n γραμμές (προεπιλογή 5)
print("2.3 df.tail(3) - Τελευταίες 3 γραμμές")
print(df_demo.tail(3))
print("\n" + "-"*50)

# 2.4 df.shape - Διαστάσεις DataFrame (γραμμές, στήλες)
print("2.4 df.shape - Διαστάσεις")
print("Αριθμός γραμμών και στηλών:", df_demo.shape)
print("\n" + "-"*50)

# 2.5 df.columns - Ονόματα στηλών
print("2.5 df.columns - Ονόματα στηλών")
print(df_demo.columns)
print("\n" + "-"*50)

# 2.6 df.index - Ετικέτες γραμμών (ευρετήριο)
print("2.6 df.index - Ετικέτες γραμμών")
print(df_demo.index)
print("\n" + "-"*50)

# 2.7 df.dtypes - Τύποι δεδομένων ανά στήλη
print("2.7 df.dtypes - Τύποι δεδομένων")
print(df_demo.dtypes)
print("\n" + "-"*50)
```

```
# 2.8 df.size - Συνολικός αριθμός στοιχείων
print("2.8 df.size - Συνολικός αριθμός στοιχείων")
print("Μέγεθος (γραμμές * στήλες):", df_demo.size)
print("\n" + "-"*50)

# 2.9 df.memory_usage() - Χρήση μνήμης ανά στήλη (σε bytes)
print("2.9 df.memory_usage() - Χρήση μνήμης ανά στήλη (bytes)")
print(df_demo.memory_usage())
print("\n" + "-"*50)

# 2.10 df.empty - Έλεγχος αν το DataFrame είναι κενό
print("2.10 df.empty - Έλεγχος αν είναι κενό")
print("Το df_demo είναι κενό;", df_demo.empty)
# Δημιουργία κενού DataFrame για επίδειξη
empty_df = pd.DataFrame()
print("Το empty_df είναι κενό;", empty_df.empty)
print("\n" + "-"*50)

# 2.11 df.sample(n) - Τυχαίο δείγμα n γραμμών
print("2.11 df.sample(5) - Τυχαίο δείγμα 5 γραμμών")
print(df_demo.sample(5))
print("\n" + "-"*50)

print("Τέλος προγράμματος.")
```

Κ. 3.4.1: Παράδειγμα των Μεθόδων για Δημιουργία & Βασικές Πληροφορίες στα DataFrames

3.4.2. Δημιουργία DataFrames από πίνακες NumPy

Η δημιουργία **DataFrame** από πίνακες **NumPy** (NumPy arrays) αποτελεί έναν από τους πιο συνηθισμένους και αποδοτικούς τρόπους εισαγωγής αριθμητικών δεδομένων στη βιβλιοθήκη pandas, εκμεταλλευόμενη τη στενή ενσωμάτωση μεταξύ των δύο βιβλιοθηκών. Ένας πίνακας NumPy (ndarray) είναι μια δισδιάστατη, ομογενής δομή δεδομένων που μπορεί να μετατραπεί άμεσα σε DataFrame, με την pandas να προσθέτει αυτόματα ετικέτες ευρετηρίου (index) και ονομάτων στηλών (column names) αν δεν οριστούν ρητά. Αυτή η διαδικασία είναι ιδιαίτερα χρήσιμη σε υπολογιστικά σενάρια όπου τα δεδομένα προέρχονται από μαθηματικές πράξεις NumPy, όπως υπολογισμούς πινάκων ή γεννήτριες τυχαίων αριθμών.[1][2][3][4]

Η βασική σύνταξη για τη μετατροπή ενός πίνακα NumPy σε DataFrame είναι η χρήση του κατασκευαστή `pd.DataFrame(array)`, όπου `array` είναι ένας δισδιάστατος πίνακας NumPy. Σε αυτή την περίπτωση, οι γραμμές του πίνακα γίνονται γραμμές του DataFrame, οι στήλες γίνονται στήλες, ενώ το προεπιλεγμένο ευρετήριο γραμμών είναι ακέραιο (0, 1, 2, ...), και οι στήλες ονομάζονται αυτόματα '0', '1', κ.λπ. Η pandas διατηρεί τον τύπο δεδομένων (dtype) του NumPy array, εξασφαλίζοντας υψηλή απόδοση σε μεγάλους πίνακες.

```
import pandas as pd
import numpy as np

# Δημιουργία δισδιάστατου πίνακα NumPy με τυχαίους αριθμούς
array = np.random.randn(4, 3)  # 4 γραμμές, 3 στήλες, κανονική κατανομή
print("Πίνακας NumPy:")
```

```
print(array)

# Μετατροπή σε DataFrame (προεπιλεγμένα index και columns)
df = pd.DataFrame(array)
print("\nDataFrame από NumPy array:")
print(df)
print("Τύπος δεδομένων:", df.dtypes)
```

Κ. 3.4.2: Παράδειγμα για τη δημιουργία DataFrames από πίνακες numpy

Παρατήρηση: Το DataFrame κληρονόμησε το dtype `float64` από το NumPy array. Οι στήλες ονομάστηκαν αυτόματα '0', '1', '2'.

Για μεγαλύτερη σαφήνεια, μπορούν να οριστούν ρητά τα ονόματα στηλών (column names) και το ευρετήριο γραμμών (index) κατά τη δημιουργία:

```
# DataFrame με ορισμένα columns και index
columns = ['A', 'B', 'C']
index = ['γραμμή1', 'γραμμή2', 'γραμμή3', 'γραμμή4']
df_named = pd.DataFrame(array, columns=columns, index=index)
print("DataFrame με ορισμένα columns και index:")
print(df_named)
```

Παρατήρηση: Οι ετικέτες index επιτρέπουν label-based πρόσβαση, π.χ. `df_named.loc['γραμμή1']` επιστρέφει τη σειρά ως Series.

Η pandas υποστηρίζει επίσης τη μετατροπή μονοδιάστατων πινάκων NumPy σε Series ή DataFrame με μία στήλη:

```
# Μονοδιάστατος πίνακας σε Series
vec = np.array([10, 20, 30, 40])
series = pd.Series(vec, name='Θερμοκρασίες')
print("Series από 1D NumPy array:")
print(series)

# Σε DataFrame με μία στήλη
df_single = pd.DataFrame(vec.reshape(-1, 1), columns=['Θερμοκρασίες'])
print("\nDataFrame από 1D array (reshape σε 2D):")
print(df_single)
```

Κ. 3.4.3: Παράδειγμα για τη δημιουργία Σειρών από πίνακες numpy

Προσοχή: Για μονοδιάστατα arrays, χρησιμοποιήστε `reshape(-1, 1)` για να γίνει στήλη, αλλιώς θα γίνει γραμμή.

3.4.3. Πρόσβαση και Επιλογή

Μέθοδος	Περιγραφή	Εφαρμογή
Βασική Πρόσβαση σε Στήλες		
df['όνομα_στήλης']	Επιλέγει μια στήλη και την επιστρέφει ως **Series** .	ηλικίες = df['Age']

Μέθοδος	Περιγραφή	Εφαρμογή
df[['στηλη1', 'στηλη2']]	Επιλέγει πολλαπλές στήλες και τις επιστρέφει ως νέο **DataFrame** .	υποσύνολο = df[['Name', 'City']]
Πρόσβαση με Ετικέτες (.loc**)**		
df.loc['ετικέτα_γραμμής']	Επιλέγει μια γραμμή με βάση την ετικέτα (label) του δείκτη (index) .	df.loc['b']
df.loc[['γραμ1', 'γραμ2']]	Επιλέγει πολλαπλές γραμμές με βάση τις ετικέτες τους .	df.loc[['a', 'c', 'e']]
df.loc['ετικέτα_αρχής':'ετικέτα_τέλους']	Κάνει slice σε γραμμές. **Σημείωση:** Με το .loc το εύρος συμπεριλαμβάνει και την ετικέτα τέλους .	df.loc['b':'d']
df.loc[:, 'όνομα_στήλης']	Επιλέγει όλες τις γραμμές (:) για μια συγκεκριμένη στήλη .	df.loc[:, 'Age']
df.loc['γραμμή', 'στήλη']	Επιλέγει μια συγκεκριμένη τιμή (κελί) με βάση την ετικέτα γραμμής και το όνομα στήλης .	df.loc['c', 'Name']
df.loc[df['στηλη'] > 5]	Επιλέγει γραμμές που ικανοποιούν μια συνθήκη (Boolean indexing) .	df.loc[df['Age'] > 30]
Πρόσβαση με Θέση (.iloc**)**		
df.iloc[θέση_γραμμής]	Επιλέγει μια γραμμή με βάση την αριθμητική της θέση (ξεκινά από το 0) .	df.iloc[1] # Επιλέγει τη 2η γραμμή
df.iloc[θέση_αρχής:θέση_τέλους]	Κάνει slice σε γραμμές. **Σημείωση:** Με το .iloc το εύρος είναι ημι-ανοιχτό (δεν συμπεριλαμβάνει τη θέση τέλους), όπως στα standard Python lists .	df.iloc[0:3] # Επιλέγει τις 3 πρώτες γραμμές
df.iloc[:, θέση_στήλης]	Επιλέγει όλες τις γραμμές για μια στήλη με βάση την αριθμητική της θέση .	df.iloc[:, 1] # Επιλέγει τη 2η στήλη
df.iloc[θέση_γραμμής, θέση_στήλης]	Επιλέγει μια συγκεκριμένη τιμή (κελί) με βάση τις αριθμητικές θέσεις γραμμής και στήλης .	df.iloc[2, 1] # Τιμή στην 3η γραμμή και 2η στήλη
df.iloc[θέση_αρχής:θέση_τέλους, θέση_στήλης]	Επιλέγει ένα υποσύνολο γραμμών για μια συγκεκριμένη στήλη .	df.iloc[1:4, 2]
Γρήγορη Πρόσβαση σε Στοιχείο		
df.at['ετικέτα_γραμμής', 'όνομα_στήλης']	Παρέχει γρήγορη πρόσβαση σε μια **μεμονωμένη τιμή** χρησιμοποιώντας ετικέτες. Είναι ταχύτερο από το .loc για这一个 σκοπό .	df.at['b', 'City']
df.iat[θέση_γραμμής, θέση_στήλης]	Παρέχει γρήγορη πρόσβαση σε μια **μεμονωμένη τιμή** χρησιμοποιώντας ακέραιες θέσεις. Είναι ταχύτερο από το .iloc για这一个 σκοπό .	df.iat[2, 1]
Επιλογή με Συνθήκες (Boolean Indexing)		
f[df['στηλη'] > 10]	Επιλέγει γραμμές όπου η τιμή σε μια στήλη ικανοποιεί μια συνθήκη .	df[df['Age'] > 30]
df[(df['στηλη1'] == 'τιμή') & (df['στηλη2'] > 5)]	Επιλέγει γραμμές που ικανοποιούν πολλαπλές συνθήκες (λογικό ΚΑΙ, &). Κάθε συνθήκη μπαίνει σε παρένθεση .	df[(df['City'] == 'London') & (df['Age'] > 25)]

Μέθοδος	Περιγραφή	Εφαρμογή
df[(df['στηλη1'] == 'τιμή') | (df['στηλη2'] < 3)]	Επιλέγει γραμμές που ικανοποιούν τουλάχιστον μία από πολλές συνθήκες (λογικό Ή, |) .	df[(df['Name'] == 'Bob') | (df['Age'] < 20)]

Πίνακας 3.4.3: Μέθοδοι για Πρόσβαση και Επιλογή στα Πλαίσια Δεδομένων

1. Βασική Πρόσβαση (Square Brackets [])

Αυτός είναι ο πιο γρήγορος τρόπος για να απομονώσεις στήλες.

- **df['column']**: Επιστρέφει μια Series. Σκεψου το σαν να παίρνεις μια μόνο κάθετη φέτα του πίνακα.
- **df[['col1', 'col2']]**: Επιστρέφει ένα DataFrame. Ακόμα και αν επιλέξεις μία στήλη αλλά χρησιμοποιήσεις διπλές αγκύλες df[['Age']], το αποτέλεσμα θα παραμείνει DataFrame (πίνακας) και όχι Series.

2. Πρόσβαση με Ετικέτες: .loc

Η μέθοδος .loc βασίζεται στα ονόματα (labels) των γραμμών και των στηλών.

- **Slicing προσοχή!** Στο .loc, το slicing συμπεριλαμβάνει το τελευταίο στοιχείο. Αν γράψεις 'b':'d', θα πάρεις και τη γραμμή 'd'.
- **Boolean Indexing**: Το .loc είναι εξαιρετικό για φιλτράρισμα. Για παράδειγμα, το df.loc[df['Age'] > 30] ψάχνει ποιες γραμμές έχουν True στη συνθήκη και τις επιστρέφει όλες.
- **Σύνταξη**: Πάντα ακολουθεί το πρότυπο df.loc[γραμμές, στήλες]. Αν θέλεις όλες τις γραμμές για μια στήλη, χρησιμοποιείς την άνω-κάτω τελεία: df.loc[:, 'Name'].

3. Πρόσβαση με Θέση: .iloc

Η μέθοδος .iloc (integer-location) αγνοεί τα ονόματα και κοιτάζει μόνο τους **ακέραιους δείκτες** (0, 1, 2...).

- **Pythonic Slicing**: Εδώ ισχύει ό,τι και στις λίστες της Python. Το 0:3 θα σου δώσει τις θέσεις 0, 1, 2 (το 3 εξαιρείται).
- **Χρήση**: Είναι ιδανικό όταν θέλεις να πάρεις π.χ. τις πρώτες 10 γραμμές ή τις τελευταίες 2 στήλες, χωρίς να σε νοιάζει πώς ονομάζονται.

4. Γρήγορη Πρόσβαση: .at και .iat

Πολλοί χρήστες τις αγνοούν, αλλά είναι πολύτιμες για βελτιστοποίηση ταχύτητας.

Μέθοδος	Βασική Διαφορά	Πότε τη χρησιμοποιούμε
.at	Πρόσβαση σε **ένα** κελί με όνομα.	Όταν θέλουμε να διαβάσουμε ή να αλλάξουμε μία συγκεκριμένη τιμή γρήγορα.
.iat	Πρόσβαση σε **ένα** κελί με θέση.	Παρόμοιο με το .at, αλλά χρησιμοποιώντας αριθμητικούς δείκτες.

Πίνακας 3.4.4: Διαφορές των μεθόδων .at και .iat

Αν η εφαρμογή σου κάνει χιλιάδες επαναλήψεις (loops) πάνω σε κελιά, η .at είναι σημαντικά ταχύτερη από την .loc.

5. Φιλτράρισμα με Συνθήκες (Boolean Indexing)

Εδώ συμβαίνει η "μαγεία" της ανάλυσης δεδομένων. Μπορείς να συνδυάσεις συνθήκες χρησιμοποιώντας τελεστές bitwise:

- **& (AND):** Πρέπει να ισχύουν και οι δύο συνθήκες.
- **| (OR):** Πρέπει να ισχύει τουλάχιστον μία.
- **~ (NOT):** Αντιστρέφει τη συνθήκη.

Σημαντικό: Πάντα να βάζεις τις συνθήκες σε **παρενθέσεις**, αλλιώς η Python θα μπερδευτεί με την προτεραιότητα των τελεστών (π.χ. df[(df['A'] > 1) & (df['B'] < 10)]).

```
# Πρόγραμμα επίδειξης μεθόδων πρόσβασης και επιλογής σε DataFrame της
# Pandas
import pandas as pd

# 1. Δημιουργία ενός παραδείγματος DataFrame
# ------------------------------------------------
# Το DataFrame περιέχει πληροφορίες για άτομα: Όνομα, Ηλικία, Πόλη, Βαθμός.
# Το index ορίζεται ρητά με γράμματα (a, b, c, d, e) για να δούμε τη χρήση
ετικετών.
data = {
    'Όνομα': ['Άννα', 'Βασίλης', 'Γιώργος', 'Δήμητρα', 'Ελένη'],
    'Ηλικία': [25, 30, 22, 35, 28],
    'Πόλη': ['Αθήνα', 'Θεσσαλονίκη', 'Πάτρα', 'Αθήνα', 'Ηράκλειο'],
    'Βαθμός': [8.5, 7.2, 9.0, 6.8, 8.9]
}
df = pd.DataFrame(data, index=['a', 'b', 'c', 'd', 'e'])

print("=" * 70)
print("ΑΡΧΙΚΟ DATAFRAME")
print("=" * 70)
print(df)
print("\n")

# 2. Βασική πρόσβαση σε στήλες
# --------------------------------
print("=" * 70)
print("2. ΒΑΣΙΚΗ ΠΡΟΣΒΑΣΗ ΣΕ ΣΤΗΛΕΣ")
print("=" * 70)

# Επιλογή μίας στήλης (επιστρέφεται Series)
print("--- Επιλογή μίας στήλης: df['Όνομα'] ---")
ser = df['Όνομα']
print(ser)
print("Τύπος:", type(ser))
print()

# Επιλογή πολλών στηλών (επιστρέφεται DataFrame)
print("--- Επιλογή πολλών στηλών: df[['Όνομα', 'Βαθμός']] ---")
sub_df = df[['Όνομα', 'Βαθμός']]
print(sub_df)
print("Τύπος:", type(sub_df))
print("\n")
```

```python
# 3. Πρόσβαση με βάση ετικέτες (label-based) → .loc[]
# ----------------------------------------------------
print("=" * 70)
print("3. ΠΡΟΣΒΑΣΗ ΜΕ .loc[] (με ετικέτες)")
print("=" * 70)

# Επιλογή μίας γραμμής με την ετικέτα του index
print("--- Επιλογή μίας γραμμής: df.loc['b'] ---")
row_b = df.loc['b']
print(row_b)
print()

# Επιλογή πολλών γραμμών (λίστα ετικετών)
print("--- Επιλογή πολλών γραμμών: df.loc[['a', 'c', 'e']] ---")
rows_ac = df.loc[['a', 'c', 'e']]
print(rows_ac)
print()

# Slice γραμμών (το εύρος είναι κλειστό: συμπεριλαμβάνει και το 'd')
print("--- Slice γραμμών: df.loc['b':'d'] ---")
slice_rows = df.loc['b':'d']
print(slice_rows)
print()

# Επιλογή όλων των γραμμών για μία συγκεκριμένη στήλη
print("--- Όλες οι γραμμές για τη στήλη 'Ηλικία': df.loc[:, 'Ηλικία'] ---")
all_age = df.loc[:, 'Ηλικία']
print(all_age)
print()

# Επιλογή συγκεκριμένου κελιού (γραμμή 'c', στήλη 'Πόλη')
print("--- Συγκεκριμένο κελί: df.loc['c', 'Πόλη'] ---")
cell_value = df.loc['c', 'Πόλη']
print("Τιμή στο df.loc['c', 'Πόλη']:", cell_value)
print()

# Boolean indexing με .loc (επιλογή γραμμών βάσει συνθήκης)
print("--- Boolean indexing: df.loc[df['Ηλικία'] > 27] ---")
filtered_loc = df.loc[df['Ηλικία'] > 27]
print(filtered_loc)
print()

# Συνδυασμός: γραμμές που ικανοποιούν συνθήκη, και μόνο συγκεκριμένες
στήλες
print("--- Συνδυασμός: df.loc[df['Ηλικία'] > 27, ['Όνομα', 'Βαθμός']] ---")
filtered_cols = df.loc[df['Ηλικία'] > 27, ['Όνομα', 'Βαθμός']]
print(filtered_cols)
print("\n")

# 4. Πρόσβαση με βάση ακέραια θέση (position-based) → .iloc[]
# ------------------------------------------------------------
print("=" * 70)
print("4. ΠΡΟΣΒΑΣΗ ΜΕ .iloc[] (με ακέραιες θέσεις)")
print("=" * 70)

# Επιλογή μίας γραμμής με την αριθμητική της θέση (2η γραμμή, δηλαδή index
1)
print("--- Επιλογή 2ης γραμμής: df.iloc[1] ---")
row_1 = df.iloc[1]
print(row_1)
```

```
print()

# Επιλογή πολλών γραμμών (θέσεις 0, 2, 4 → 1η, 3η, 5η)
print("--- Επιλογή γραμμών με λίστα θέσεων: df.iloc[[0, 2, 4]] ---")
rows_0_2_4 = df.iloc[[0, 2, 4]]
print(rows_0_2_4)
print()

# Slice γραμμών (ημι-ανοιχτό εύρος: θέσεις 1 έως 3, δηλ. γραμμές 2,3,4)
print("--- Slice γραμμών: df.iloc[1:4] ---")
slice_iloc = df.iloc[1:4]   # θέσεις 1,2,3 → γραμμές b, c, d
print(slice_iloc)
print()

# Επιλογή όλων των γραμμών για μία στήλη με βάση τη θέση της (στήλη 2 →
'Πόλη')
print("--- Όλες οι γραμμές για τη στήλη θέσης 2 (Πόλη): df.iloc[:, 2] ---")
col_2 = df.iloc[:, 2]
print(col_2)
print()

# Επιλογή συγκεκριμένου κελιού (γραμμή 3, στήλη 1) → γραμμή d, στήλη Ηλικία
print("--- Συγκεκριμένο κελί: df.iloc[3, 1] ---")
cell_iloc = df.iloc[3, 1]   # 4η γραμμή (θέση 3), 2η στήλη (θέση 1) →
Ηλικία της Δήμητρας = 35
print("Τιμή στο df.iloc[3, 1]:", cell_iloc)
print()

# Υποσύνολο γραμμών για μια στήλη (γραμμές 1-3, στήλη 0 → Όνομα)
print("--- Γραμμές 2-4 για τη στήλη 0: df.iloc[1:4, 0] ---")
sub_iloc = df.iloc[1:4, 0]   # θέσεις 1,2,3 στη γραμμή, στήλη 0
print(sub_iloc)
print("\n")

# 5. Γρήγορη πρόσβαση σε μεμονωμένα στοιχεία → .at[] και .iat[]
# ----------------------------------------------------------------
print("=" * 70)
print("5. ΓΡΗΓΟΡΗ ΠΡΟΣΒΑΣΗ ΣΕ ΜΕΜΟΝΩΜΕΝΑ ΣΤΟΙΧΕΙΑ (.at και .iat)")
print("=" * 70)

# .at[]: με ετικέτες (label-based)
print("--- df.at['c', 'Βαθμός'] ---")
val_at = df.at['c', 'Βαθμός']
print("Βαθμός του Γιώργου (γραμμή c):", val_at)
print()

# .iat[]: με ακέραιες θέσεις
print("--- df.iat[4, 3] ---")
val_iat = df.iat[4, 3]   # 5η γραμμή (Ελένη), 4η στήλη (Βαθμός)
print("Βαθμός της Ελένης (θέση [4,3]):", val_iat)
print("\n")

# 6. Επιλογή με συνθήκες (Boolean indexing) απευθείας με df[]
# ----------------------------------------------------------------
print("=" * 70)
print("6. BOOLEAN INDEXING (επιλογή γραμμών με συνθήκες)")
print("=" * 70)

# Απλή συνθήκη: όσοι έχουν ηλικία > 27
print("--- df[df['Ηλικία'] > 27] ---")
bool1 = df[df['Ηλικία'] > 27]
```

```
print(bool1)
print()

# Συνθήκη με ισότητα: όσοι μένουν στην Αθήνα
print("--- df[df['Πόλη'] == 'Αθήνα'] ---")
bool2 = df[df['Πόλη'] == 'Αθήνα']
print(bool2)
print()

# Πολλαπλές συνθήκες με AND (&): Ηλικία > 25 ΚΑΙ Βαθμός >= 8.5
# (κάθε συνθήκη σε παρένθεση)
print("--- df[(df['Ηλικία'] > 25) & (df['Βαθμός'] >= 8.5)] ---")
bool_and = df[(df['Ηλικία'] > 25) & (df['Βαθμός'] >= 8.5)]
print(bool_and)
print()

# Πολλαπλές συνθήκες με OR (|): Ηλικία < 23 Ή Πόλη == 'Ηράκλειο'
print("--- df[(df['Ηλικία'] < 23) | (df['Πόλη'] == 'Ηράκλειο')] ---")
bool_or = df[(df['Ηλικία'] < 23) | (df['Πόλη'] == 'Ηράκλειο')]
print(bool_or)
print()

# Συνδυασμός boolean indexing με επιλογή στηλών (μέσω .loc ή απλά με δύο
[]?)
# Μπορούμε να χρησιμοποιήσουμε .loc ή να βάλουμε την επιλογή στηλών μετά.
print("--- Επιλογή μόνο Ονόματος και Βαθμού για όσους έχουν Ηλικία > 27 ---
")
bool_cols = df.loc[df['Ηλικία'] > 27, ['Όνομα', 'Βαθμός']]
print(bool_cols)
print("\n")

# 7. Πρόσθετες χρήσιμες εντολές γρήγορης επισκόπησης
# ------------------------------------------------------
print("=" * 70)
print("7. ΕΠΙΠΛΕΟΝ ΕΝΤΟΛΕΣ ΕΠΙΣΚΟΠΗΣΗΣ")
print("=" * 70)

# .head() : οι πρώτες γραμμές (προεπιλογή 5)
print("--- df.head(3) (πρώτες 3 γραμμές) ---")
print(df.head(3))
print()

# .tail() : οι τελευταίες γραμμές
print("--- df.tail(2) (τελευταίες 2 γραμμές) ---")
print(df.tail(2))
print()

# .sample() : τυχαίες γραμμές
print("--- df.sample(2) (2 τυχαίες γραμμές) ---")
print(df.sample(2))
print()

# .info() : πληροφορίες DataFrame
print("--- df.info() ---")
df.info()
print()

# .describe() : στατιστική περίληψη για αριθμητικές στήλες
print("--- df.describe() ---")
print(df.describe())
print()
```

```
print("=" * 70)
print("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
print("=" * 70)
```

Πίνακας 3.4.5:Παράδειγμα των Μεθόδων για Πρόσβαση και Επιλογή στα Πλαίσια Δεδομένων

3.4.4. Boolean Indexing & Φιλτράρισμα

Μέθοδος	Περιγραφή	Επιστροφή
Βασικό Boolean Indexing		
df[df['column'] == value]	Φιλτράρει τις γραμμές όπου η τιμή μιας στήλης είναι **ίση** με μια συγκεκριμένη τιμή.	df[df['Age'] == 27]
df[df['column'] != value]	Φιλτράρει τις γραμμές όπου η τιμή μιας στήλης **δεν είναι ίση** με μια τιμή.	df[df['Gender'] != 'Male']
df[df['column'] > value]	Φιλτράρει γραμμές με βάση συγκριτικούς τελεστές (>, <, >=, <=).	df[df['Age'] >= 25]
df[~df['condition']]	Αντιστρέφει μια συνθήκη (τελεστής ~), επιστρέφοντας όσες γραμμές **δεν** την ικανοποιούν.	df[~(df['Age'] < 25)]
Φιλτράρισμα με Πολλαπλές Τιμές		
df[df['column'].isin(list)]	Φιλτράρει γραμμές όπου η τιμή της στήλης περιλαμβάνεται σε μια **λίστα επιθυμητών τιμών**.	df[df['City'].isin(['Paris', 'Tokyo'])]
df[~df['column'].isin(list)]	Φιλτράρει γραμμές όπου η τιμή της στήλης **δεν** περιλαμβάνεται σε μια λίστα.	df[~df['City'].isin(['Paris', 'Tokyo'])]
Πολλαπλές Συνθήκες		
df[(df['col1'] > x) & (df['col2'] == y)]	Συνδυασμός συνθηκών με λογικό **ΚΑΙ (&)**. Κάθε συνθήκη πρέπει να είναι σε παρένθεση.	df[(df['Gender'] == 'Female') & (df['Age'] > 25)]
df[(df['col1'] > x) | (df['col2'] == y)]	Συνδυασμός συνθηκών με λογικό **Ή (|)**.	df[(df['Age'] > 30) | (df['City'] == 'London')]
df[df.eq(value).any(axis=1)]	Βρίσκει γραμμές όπου **οποιαδήποτε** στήλη περιέχει μια συγκεκριμένη τιμή.	df[df.eq('Male').any(axis=1)]

Μέθοδος	Περιγραφή	Επιστροφή
Φιλτράρισμα με Strings		
df[df['column'].str.contains('text')]	Φιλτράρει γραμμές όπου το κείμενο σε μια στήλη **περιέχει** μια υπο-συμβολοσειρά (υποστηρίζει κανονικές εκφράσεις).	df[df['City'].str.contains('on', case=False, na=False)]
df[df['column'].str.startswith('text')]	Φιλτράρει γραμμές όπου το κείμενο **ξεκινάει** με μια συγκεκριμένη συμβολοσειρά.	df[df['Name'].str.startswith('J')]
df[df['column'].str.endswith('text')]	Φιλτράρει γραμμές όπου το κείμενο **τελειώνει** με μια συγκεκριμένη συμβολοσειρά.	df[df['City'].str.endswith('o')]
Φιλτράρισμα με Ημερομηνίες		
df[df['date_column'].dt.year == year]	Φιλτράρει με βάση το **έτος** από μια στήλη ημερομηνίας.	df[df['c'].dt.year % 2 == 1]
df[df['date_column'].dt.month == month]	Φιλτράρει με βάση τον **μήνα**.	df[df['c'].dt.month == 9]
df[df['date_column'].dt.day_name().isin(list)]	Φιλτράρει με βάση το **όνομα της ημέρας**.	df[df['c'].dt.day_name().isin(['Monday', 'Friday'])]
Φιλτράρισμα για Κενές/Μη Κενές Τιμές		
df[df['column'].isna()]	Φιλτράρει γραμμές όπου η τιμή είναι **άγνωστη (NaN)**.	df[df['City'].isna()]
df[df['column'].notna()]	Φιλτράρει γραμμές όπου η τιμή **δεν είναι άγνωστη**.	df[df['City'].notna()]
Προχωρημένες & Εναλλακτικές Μέθοδοι		
df.query("condition")	Επιτρέπει φιλτράρισμα με χρήση **συμβολοσειράς (string)**, παρόμοια με SQL. Είναι πιο ευανάγνωστο για πολύπλοκες συνθήκες.	df.query("Age > 25 and Gender == 'Female'")
df.loc[boolean_condition, columns]	Φιλτράρει με boolean συνθήκη και παράλληλα μπορεί να **επιλέξει συγκεκριμένες στήλες** για επιστροφή.	df.loc[df['Age'] > 25, ['Name', 'City']]

Μέθοδος	Περιγραφή	Επιστροφή
df.mask(cond, other)	Αντικαθιστά τις τιμές όπου η συνθήκη είναι True με άλλες τιμές (χρήσιμο για αντικατάσταση, όχι απλό φιλτράρισμα).	df.mask(df % 3 == 0, -df)

Πίνακας 3.4.6: Μέθοδοι για Boolean Indexing & Φιλτράρισμα για Πλαίσια Δεδομένων

Το φιλτράρισμα (ή αλλιώς "querying") είναι η διαδικασία όπου απομονώνεις την πληροφορία που πραγματικά χρειάζεσαι μέσα από έναν ωκεανό δεδομένων.

1. Boolean Indexing (Η "Κλασική" Μέθοδος)

Αυτή η μέθοδος βασίζεται στη δημιουργία μιας "μάσκας" από True και False.

- **Συγκρίσεις (==, !=, >, <):** Είναι η βάση των πάντων. Όταν γράφεις df['Age'] > 25, η Pandas δημιουργεί μια σειρά από True/False για κάθε γραμμή.
- **Ο τελεστής ~ (NOT):** Είναι εξαιρετικά χρήσιμος όταν ξέρεις τι **δεν** θέλεις. Αντί να γράφεις περίπλοκες συνθήκες για να συμπεριλάβεις τα πάντα εκτός από κάτι συγκεκριμένο, απλώς φτιάξε τη συνθήκη για αυτό που θέλεις να πετάξεις και βάλε το ~ μπροστά.

2. Φιλτράρισμα με Λίστες και Strings

Εδώ η Pandas δείχνει τη δύναμή της στον χειρισμό κειμένου.

- **.isin([])**: Σκέψου το σαν ένα πολλαπλό "Η". Αντί για (df.City == 'A') | (df.City == 'B'), γράφεις df.City.isin(['A', 'B']). Είναι πολύ πιο καθαρό και γρήγορο.
- **.str.contains()**: Το απόλυτο εργαλείο για αναζήτηση λέξεων.

Tip: Χρησιμοποίησε πάντα το na=False μέσα στο contains, γιατί αν η στήλη έχει κενά (NaN), η μέθοδος θα πετάξει σφάλμα. Το case=False σε σώζει από το να ψάχνεις "Athens" και "athens" ξεχωριστά.

3. Πολλαπλές Συνθήκες & any(axis=1)

- **Παρενθέσεις:** Το νούμερο ένα λάθος των αρχαρίων είναι να ξεχνούν τις παρενθέσεις στις συνθήκες. Η Python έχει διαφορετική προτεραιότητα για το & και το |, οπότε οι παρενθέσεις είναι υποχρεωτικές.
- **any(axis=1)**: Αυτό είναι "πυρηνικό όπλο". Σου επιτρέπει να ψάξεις μια τιμή σε **όλες τις στήλες ταυτόχρονα**. Αν για παράδειγμα ψάχνεις τη λέξη "Error" οπουδήποτε μέσα στο DataFrame, το any(axis=1) θα σου βρει τη γραμμή όποια στήλη κι αν την περιέχει.

4. Time-Series Filtering (Ημερομηνίες)

Για να δουλέψουν τα .dt.year, .dt.month κ.λπ., η στήλη σου **πρέπει** να είναι τύπου datetime64. Αν είναι απλό κείμενο (object), χρησιμοποίησε πρώτα την pd.to_datetime(df['column']). Είναι ο πιο εύκολος τρόπος να κάνεις εποχιακή ανάλυση (π.χ. "δώσε μου όλες τις πωλήσεις που έγιναν Παρασκευή").

5. Προχωρημένες Μέθοδοι (query & mask)

- **df.query()**: Αν προέρχεσαι από τον κόσμο της SQL, θα τη λατρέψεις. Σου επιτρέπει να γράφεις τη συνθήκη ως κείμενο, κάτι που κάνει τον κώδικα πολύ πιο αναγνώσιμο.

- *Παράδειγμα:* df.query("Age > 30 and City == 'Paris'").

- **df.mask()**: Η διαφορά εδώ είναι ότι η mask δεν "πετάει" τις γραμμές, αλλά τις "κρύβει" ή τις αντικαθιστά. Είναι ιδανική για data cleaning (π.χ. "όπου η ηλικία είναι αρνητική, κάνε την NaN").

Πρακτικές Συμβουλές (Best Practices)

1. **Για Επιλογή + Φιλτράρισμα:** Χρησιμοποίησε το .loc. Είναι πιο αποδοτικό να γράφεις: df.loc[df['Age'] > 25, 'Name'] παρά το: df[df['Age'] > 25]['Name'] (αυτό λέγεται *chained indexing* και μπορεί να προκαλέσει προειδοποιήσεις στην Pandas).

2. **Missing Values:** Πριν φιλτράρεις, έλεγχε πάντα για isna(). Πολλές φορές οι υπολογισμοί αποτυγχάνουν επειδή υπάρχουν κρυμμένα κενά κελιά.

```
import pandas as pd
import numpy as np

# ----------------------------------------------------------------------
# Δημιουργία ενός αντιπροσωπευτικού DataFrame για τα παραδείγματα
# ----------------------------------------------------------------------
print("ΔΗΜΙΟΥΡΓΙΑ ΔΕΙΓΜΑΤΟΣ DataFrame")
data = {
    'Name': ['Anna', 'George', 'Maria', 'Nikos', 'Eleni', 'Dimitris',
'Sofia', 'John', 'Alice', 'Bob'],
    'Age': [25, 30, 28, 35, 22, 27, 24, 40, 31, 29],
    'Gender': ['Female', 'Male', 'Female', 'Male', 'Female', 'Male',
'Female', 'Male', 'Female', 'Male'],
    'City': ['Athens', 'Thessaloniki', 'Patras', 'Heraklion', 'Volos',
'Chania', 'Larisa', 'London', 'Paris', 'Tokyo'],
    'Salary': [50000, 60000, 55000, 65000, 48000, 62000, 51000, 80000,
75000, 70000],
    'Join_Date': pd.date_range('2020-01-01', periods=10, freq='M')
}
# Εισαγωγή μερικών κενών τιμών για επίδειξη
data['City'][5] = np.nan  # Η Chania γίνεται NaN
data['Salary'][8] = np.nan  # Η Alice έχει NaN στο Salary

df = pd.DataFrame(data)
print("Αρχικό DataFrame:")
print(df)
print("\n" + "="*80 + "\n")

# ----------------------------------------------------------------------
# 1. ΒΑΣΙΚΟ BOOLEAN INDEXING
# ----------------------------------------------------------------------
print("1. ΒΑΣΙΚΟ BOOLEAN INDEXING")
print("-" * 50)

# 1.1 df[df['column'] == value] - Φιλτράρισμα με ισότητα
print("1.1 df[df['column'] == value] - Ίσο με 27 (Age)")
filtered = df[df['Age'] == 27]
print(filtered)
print()

# 1.2 df[df['column'] != value] - Φιλτράρισμα με ανισότητα
print("1.2 df[df['column'] != value] - Διάφορο του 'Male' (Gender)")
filtered = df[df['Gender'] != 'Male']
print(filtered)
print()
```

```python
# 1.3 df[df['column'] > value] - Συγκριτικοί τελεστές
print("1.3 df[df['column'] > value] - Ηλικία >= 30")
filtered = df[df['Age'] >= 30]
print(filtered)
print()

# 1.4 df[~df['condition']] - Αντιστροφή συνθήκης
print("1.4 df[~df['condition']] - Όχι (Ηλικία < 25) (δηλ. >=25)")
filtered = df[~(df['Age'] < 25)]
print(filtered)
print("\n" + "="*80 + "\n")

# ----------------------------------------------------------------------
# 2. ΦΙΛΤΡΑΡΙΣΜΑ ΜΕ ΠΟΛΛΑΠΛΕΣ ΤΙΜΕΣ
# ----------------------------------------------------------------------
print("2. ΦΙΛΤΡΑΡΙΣΜΑ ΜΕ ΠΟΛΛΑΠΛΕΣ ΤΙΜΕΣ")
print("-" * 50)

# 2.1 df[df['column'].isin(list)] - Περιλαμβάνεται σε λίστα
print("2.1 df[df['column'].isin(list)] - Πόλεις Athens, London, Paris")
cities_list = ['Athens', 'London', 'Paris']
filtered = df[df['City'].isin(cities_list)]
print(filtered)
print()

# 2.2 df[~df['column'].isin(list)] - Δεν περιλαμβάνεται σε λίστα
print("2.2 df[~df['column'].isin(list)] - Όχι στις Athens, London, Paris")
filtered = df[~df['City'].isin(cities_list)]
print(filtered)
print("\n" + "="*80 + "\n")

# ----------------------------------------------------------------------
# 3. ΠΟΛΛΑΠΛΕΣ ΣΥΝΘΗΚΕΣ
# ----------------------------------------------------------------------
print("3. ΠΟΛΛΑΠΛΕΣ ΣΥΝΘΗΚΕΣ")
print("-" * 50)

# 3.1 Λογικό ΚΑΙ (&)
print("3.1 (df['Gender'] == 'Female') & (df['Age'] > 25)")
filtered = df[(df['Gender'] == 'Female') & (df['Age'] > 25)]
print(filtered)
print()

# 3.2 Λογικό Ή (|)
print("3.2 (df['Age'] > 35) | (df['City'] == 'London')")
filtered = df[(df['Age'] > 35) | (df['City'] == 'London')]
print(filtered)
print()

# 3.3 df.eq(value).any(axis=1) - Οποιαδήποτε στήλη περιέχει την τιμή 'Male'
print("3.3 df.eq('Male').any(axis=1) - Γραμμές με 'Male' σε οποιαδήποτε
στήλη")
filtered = df[df.eq('Male').any(axis=1)]
print(filtered)
print("\n" + "="*80 + "\n")

# ----------------------------------------------------------------------
# 4. ΦΙΛΤΡΑΡΙΣΜΑ ΜΕ STRINGS
# ----------------------------------------------------------------------
print("4. ΦΙΛΤΡΑΡΙΣΜΑ ΜΕ STRINGS")
```

```
print("-" * 50)

# 4.1 str.contains() - Περιέχει υποσυμβολοσειρά (case insensitive,
χειρισμός NaN)
print("4.1 df['City'].str.contains('on', case=False, na=False) - Πόλεις με
'on'")
filtered = df[df['City'].str.contains('on', case=False, na=False)]
print(filtered)
print()

# 4.2 str.startswith() - Ξεκινά με συγκεκριμένο πρόθεμα
print("4.2 df['Name'].str.startswith('J') - Ονόματα που αρχίζουν με 'J'")
filtered = df[df['Name'].str.startswith('J')]
print(filtered)
print()

# 4.3 str.endswith() - Τελειώνει με συγκεκριμένο επίθημα
print("4.3 df['City'].str.endswith('o') - Πόλεις που τελειώνουν σε 'o'")
filtered = df[df['City'].str.endswith('o', na=False)]  # na=False αποφεύγει
σφάλμα με NaN
print(filtered)
print("\n" + "="*80 + "\n")

# ----------------------------------------------------------------------
# 5. ΦΙΛΤΡΑΡΙΣΜΑ ΜΕ ΗΜΕΡΟΜΗΝΙΕΣ
# ----------------------------------------------------------------------
print("5. ΦΙΛΤΡΑΡΙΣΜΑ ΜΕ ΗΜΕΡΟΜΗΝΙΕΣ")
print("-" * 50)

# Βεβαιωνόμαστε ότι η στήλη Join_Date είναι datetime
df['Join_Date'] = pd.to_datetime(df['Join_Date'])

# 5.1 Φιλτράρισμα με βάση το έτος (π.χ. έτος 2020)
print("5.1 df[df['Join_Date'].dt.year == 2020] - Εγγραφές το 2020")
filtered = df[df['Join_Date'].dt.year == 2020]
print(filtered)
print()

# 5.2 Φιλτράρισμα με βάση τον μήνα (π.χ. μήνας Σεπτέμβριος = 9)
print("5.2 df[df['Join_Date'].dt.month == 9] - Εγγραφές τον Σεπτέμβριο")
# Στα δεδομένα μας οι ημερομηνίες είναι μηνιαίες από Ιανουάριο, οπότε ίσως
δεν υπάρχει Σεπτέμβριος.
# Ας ελέγξουμε για μήνα 1 (Ιανουάριος) για να έχουμε αποτέλεσμα.
filtered = df[df['Join_Date'].dt.month == 1]
print(filtered)
print()

# 5.3 Φιλτράρισμα με βάση ημέρα της εβδομάδας
print("5.3 df[df['Join_Date'].dt.day_name().isin(['Monday', 'Friday'])] -
Δευτέρα ή Παρασκευή")
filtered = df[df['Join_Date'].dt.day_name().isin(['Monday', 'Friday'])]
print(filtered)
print("\n" + "="*80 + "\n")

# ----------------------------------------------------------------------
# 6. ΦΙΛΤΡΑΡΙΣΜΑ ΓΙΑ ΚΕΝΕΣ / ΜΗ ΚΕΝΕΣ ΤΙΜΕΣ
# ----------------------------------------------------------------------
print("6. ΦΙΛΤΡΑΡΙΣΜΑ ΓΙΑ ΚΕΝΕΣ / ΜΗ ΚΕΝΕΣ ΤΙΜΕΣ")
print("-" * 50)

# 6.1 isna() - Κενές τιμές
```

```
print("6.1 df[df['City'].isna()] - Γραμμές με άγνωστη πόλη (NaN)")
filtered = df[df['City'].isna()]
print(filtered)
print()

# 6.2 notna() - Μη κενές τιμές
print("6.2 df[df['Salary'].notna()] - Γραμμές με γνωστό μισθό")
filtered = df[df['Salary'].notna()]
print(filtered)
print("\n" + "="*80 + "\n")

# ----------------------------------------------------------------------
# 7. ΠΡΟΧΩΡΗΜΕΝΕΣ & ΕΝΑΛΛΑΚΤΙΚΕΣ ΜΕΘΟΔΟΙ
# ----------------------------------------------------------------------
print("7. ΠΡΟΧΩΡΗΜΕΝΕΣ & ΕΝΑΛΛΑΚΤΙΚΕΣ ΜΕΘΟΔΟΙ")
print("-" * 50)

# 7.1 df.query() - Φιλτράρισμα με συμβολοσειρά
print("7.1 df.query(\"Age > 25 and Gender == 'Female'\") - Με query")
filtered = df.query("Age > 25 and Gender == 'Female'")
print(filtered)
print()

# 7.2 df.loc[boolean_condition, columns] - Επιλογή συγκεκριμένων στηλών
print("7.2 df.loc[df['Age'] > 25, ['Name', 'City']] - Μόνο Name και City
για ηλικίες >25")
filtered = df.loc[df['Age'] > 25, ['Name', 'City']]
print(filtered)
print()

# 7.3 df.mask(cond, other) - Αντικατάσταση τιμών όπου ισχύει συνθήκη
print("7.3 df.mask(df['Age'] % 2 == 0, -df['Age']) - Αρνητικοποίηση ηλικιών
που είναι άρτιες (μόνο για την Age)")
# Δημιουργούμε αντίγραφο για να μην τροποποιήσουμε το αρχικό DataFrame
df_copy = df.copy()
df_copy['Age'] = df_copy['Age'].mask(df_copy['Age'] % 2 == 0, -
df_copy['Age'])
print(df_copy[['Name', 'Age']])  # Εμφανίζουμε μόνο Name και Age για
σύγκριση
print()

print("\n" + "="*80 + "\n")
print("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
```

Κ. 3.4.4: Παράδειγμα των Μεθόδων για Boolean Indexing & Φιλτράρισμα για Πλαίσια Δεδομένων

3.4.5. Ταξινόμηση & Αναδιάταξη

Μέθοδος	Περιγραφή	Επιστροφή
Ταξινόμηση		
sort_values()	Ταξινομεί το DataFrame βάσει των τιμών μιας ή περισσότερων στηλών. Παρέχει ελέγχο για τη σειρά (αύξουσα/φθίνουσα) και τη διαχείριση κενών τιμών .	df.sort_values(by='Ηλικία', ascending=False) df.sort_values(by=['Τμήμα', 'Μισθός'], ascending=[True, False])
sort_index()	Ταξινομεί το DataFrame βάσει των ετικετών του δείκτη (index) (γραμμές ή στήλες) .	df.sort_index() df.sort_index(axis=1, ascending=False)

Μέθοδος	Περιγραφή	Επιστροφή
nlargest()	Επιστρέφει τις πρώτες n γραμμές με τις μεγαλύτερες τιμές σε μια στήλη. Είναι πιο αποδοτικό από το sort_values().head(n) για μεγάλα σύνολα δεδομένων .	df.nlargest(3, 'Πόντοι')
nsmallest()	Επιστρέφει τις πρώτες n γραμμές με τις μικρότερες τιμές σε μια στήλη .	df.nsmallest(5, 'Ηλικία')
Αναδιάταξη & Αλλαγή Σχήματος		
reset_index()	Επαναφέρει τον δείκτη (index) στις προεπιλεγμένες ακέραιες τιμές (0, 1, 2,...). Ο παλιός δείκτης προστίθεται ως στήλη, εκτός αν οριστεί drop=True .	df.sort_values('Όνομα').reset_index(drop=True)
set_index()	Ορίζει μια υπάρχουσα στήλη ως τον δείκτη (index) του DataFrame .	df.set_index('ID_Πελάτη', inplace=True)
reindex()	Προσαρμόζει το DataFrame σε ένα νέο σύνολο ετικετών (για γραμμές ή στήλες). Μπορεί να κάνει fill ή drop τιμές για να ταιριάξει στο νέο index .	df.reindex(['γ', 'β', 'α'])
melt()	Μετατρέπει ένα DataFrame από "wide" (πολλές στήλες) σε "long" format (λίγες στήλες), μαζεύοντας πολλές στήλες σε ζεύγη μεταβλητής-τιμής .	pd.melt(df, id_vars=['id'], var_name='έτος', value_name='πωλήσεις')
pivot()	Δημιουργεί έναν νέο πίνακα από τα δεδομένα, ορίζοντας ένα νέο index και νέες στήλες (η αντίθετη λειτουργία του melt) .	df.pivot(index='ημερομηνία', columns='μεταβλητή', values='τιμή')
pivot_table()	Λειτουργεί όπως το pivot, αλλά μπορεί να συνοψίσει δεδομένα με συναρτήσεις συνάθροισης (π.χ. mean, sum) όταν υπάρχουν διπλότυπες εγγραφές .	pd.pivot_table(df, values='πώληση', index='περιοχή', columns='προϊόν', aggfunc='sum')
stack()	Συμπτύσσει (συμπιέζει) το επίπεδο των στηλών σε γραμμές, μεταφέροντας το DataFrame σε μια "μακρύτερη" μορφή .	df.stack()
unstack()	Απλώνει (επεκτείνει) ένα επίπεδο του δείκτη (index) σε στήλες, μεταφέροντας το DataFrame σε μια "πλατύτερη" μορφή (αντίστροφο του stack) .	df.unstack()

Μέθοδος	Περιγραφή	Επιστροφή
rename()	Μετονομάζει ετικέτες στηλών ή γραμμών .	df.rename(columns={'old': 'new'})
set_axis()	Αντικαθιστά ολόκληρο τον άξονα (index ή στήλες) με μια νέα λίστα από ετικέτες .	df.set_axis(['A', 'B', 'C'], axis=1, inplace=True)

Πίνακας 3.4.7: Μέθοδοι για ταξινόμηση & αναδιάταξη για Πλαίσια Δεδομένων

Οι μέθοδοι αυτές δεν φιλτράρουν απλώς τι βλέπουμε, αλλά αλλάζουν τη δομή, τη σειρά και το σχήμα του DataFrame για να γίνει κατάλληλο για ανάλυση ή οπτικοποίηση.

1. Ταξινόμηση (Sorting)

Η ταξινόμηση είναι το πρώτο βήμα για να εντοπίσεις outliers ή τάσεις.

- **sort_values()**: Η πιο συνηθισμένη μέθοδος.
    - **Tip:** Όταν ταξινομείς πολλές στήλες, το ascending=[True, False] σου επιτρέπει να δεις π.χ. τα τμήματα αλφαβητικά, αλλά τους μισθούς εντός αυτών από τον μεγαλύτερο στον μικρότερο.
- **nlargest() / nsmallest()**: **Pro tip!** Αν έχεις ένα DataFrame με 1.000.000 γραμμές και θες μόνο τους 5 κορυφαίους, αυτές οι μέθοδοι είναι πολύ πιο γρήγορες από το να ταξινομήσεις ολόκληρο τον πίνακα με sort_values().

2. Διαχείριση Index (Ευρετήριο)

Ο δείκτης (index) είναι η "διεύθυνση" κάθε γραμμής.

- **set_index()**: Χρήσιμο όταν μια στήλη (π.χ. ID_Πελάτη ή Ημερομηνία) είναι μοναδική και θες να τη χρησιμοποιήσεις για γρήγορη αναζήτηση με .loc.
- **reset_index()**: Η "σωτήρια" μέθοδος. Μετά από φιλτράρισμα ή ταξινόμηση, οι δείκτες συχνά ανακατεύονται (π.χ. 5, 2, 100). Με το reset_index(drop=True) επαναφέρεις τη σειρά σε 0, 1, 2... και καθαρίζεις το DataFrame.

3. Αλλαγή Σχήματος (Reshaping)

Εδώ τα πράγματα γίνονται ενδιαφέροντα. Συχνά τα δεδομένα έρχονται σε μορφή που βολεύει την καταχώριση αλλά όχι την ανάλυση.

- **melt() (Wide to Long):** Μετατρέπει πολλές στήλες σε γραμμές. Φαντάσου έναν πίνακα όπου κάθε στήλη είναι ένας μήνας (Ιαν, Φεβ, Μαρ). Με το melt, όλες αυτές γίνονται μια στήλη "Μήνας" και μια στήλη "Τιμή".
- **pivot() & pivot_table() (Long to Wide):** Το αντίστροφο του melt.
    - **Η διαφορά:** Η pivot_table είναι πιο ισχυρή γιατί μπορεί να κάνει **μαθηματικές πράξεις**. Αν έχεις πολλές εγγραφές για την ίδια ημερομηνία, η pivot_table μπορεί να σου βγάλει τον μέσο όρο (mean) ή το άθροισμα (sum).
- **stack() & unstack()**: Δουλεύουν κυρίως με **MultiIndex** (πολλαπλά επίπεδα επικεφαλίδων).
    - Το stack() "σπρώχνει" τις στήλες να γίνουν εσωτερικό επίπεδο των γραμμών.
    - Το unstack() "τραβάει" ένα επίπεδο των γραμμών και το κάνει στήλες.

4. Μετονομασία & Άξονες

- **rename()**: Πολύ ευέλικτο. Μπορείς να αλλάξεις μόνο μία στήλη: df.rename(columns={'παλιό_όνομα': 'νέο_όνομα'}).
- **set_axis()**: Πιο δραστικό. Αντικαθιστά όλα τα ονόματα των στηλών με μια νέα λίστα που του δίνεις. Χρήσιμο όταν ξέρεις ότι η σειρά των στηλών είναι σωστή αλλά τα ονόματα είναι ακαταλαβίστικα.

```
import pandas as pd
import numpy as np

# ---------------------------------------------------------------------
# Δημιουργία ενός αντιπροσωπευτικού DataFrame για τα παραδείγματα
# ---------------------------------------------------------------------
print("ΔΗΜΙΟΥΡΓΙΑ ΔΕΙΓΜΑΤΟΣ DataFrame")
print("-" * 80)

# Αρχικό DataFrame με ποικιλία δεδομένων
data = {
    'ID': [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],
    'Name': ['Anna', 'George', 'Maria', 'Nikos', 'Eleni', 'Dimitris',
'Sofia', 'John', 'Alice', 'Bob'],
    'Department': ['Sales', 'IT', 'Sales', 'HR', 'IT', 'Sales', 'HR', 'IT',
'Sales', 'HR'],
    'Salary': [50000, 60000, 55000, 65000, 48000, 62000, 51000, 80000,
75000, 70000],
    'Years': [2, 5, 3, 8, 1, 6, 4, 10, 7, 9],
    'Score': [85, 92, 78, 88, 95, 67, 72, 90, 84, 79]
}
df_original = pd.DataFrame(data)
df_original['Join_Date'] = pd.date_range('2018-01-01', periods=10,
freq='Y')
print("Αρχικό DataFrame:")
print(df_original)
print("\n" + "="*100 + "\n")

# Για ορισμένες μεθόδους θα χρειαστούμε διαφορετικές δομές, οπότε
δημιουργούμε αντίγραφα ή νέα DataFrames.

# ---------------------------------------------------------------------
# 1. ΤΑΞΙΝΟΜΗΣΗ (SORTING)
# ---------------------------------------------------------------------
print("1. ΤΑΞΙΝΟΜΗΣΗ (SORTING)")
print("=" * 60)

# 1.1 sort_values() - Ταξινόμηση βάσει τιμών στηλών
print("\n1.1 sort_values() - Ταξινόμηση κατά Μισθό φθίνουσα")
df_sorted_salary = df_original.sort_values(by='Salary', ascending=False)
print(df_sorted_salary[['Name', 'Salary']])
print()

print("1.1 sort_values() - Ταξινόμηση κατά Τμήμα (αύξουσα) και Μισθό
(φθίνουσα)")
df_sorted_multi = df_original.sort_values(by=['Department', 'Salary'],
ascending=[True, False])
print(df_sorted_multi[['Name', 'Department', 'Salary']])
print()

# 1.2 sort_index() - Ταξινόμηση βάσει δείκτη
```

```
print("\n1.2 sort_index() - Ταξινόμηση βάσει δείκτη (προεπιλογή αύξουσα)")
# Ανακατεύουμε το DataFrame για να έχουμε μη ταξινομημένο index
df_shuffled = df_original.sample(frac=1,
random_state=42).reset_index(drop=True)
print("Ανακατεμένο DataFrame (τυχαία σειρά):")
print(df_shuffled.head())
df_sorted_index = df_shuffled.sort_index()
print("Μετά από sort_index():")
print(df_sorted_index.head())
print()

print("sort_index() - Φθίνουσα ταξινόμηση και κατά μήκος στηλών (axis=1)")
# Δημιουργούμε ένα μικρό DataFrame με στήλες εκτός αλφαβητικής σειράς
df_cols = pd.DataFrame({'B': [1, 2], 'A': [3, 4], 'C': [5, 6]})
print("Πριν:")
print(df_cols)
df_cols_sorted = df_cols.sort_index(axis=1, ascending=False)
print("Μετά από sort_index(axis=1, ascending=False):")
print(df_cols_sorted)
print()

# 1.3 nlargest() - Οι μεγαλύτερες τιμές
print("\n1.3 nlargest(3, 'Score') - 3 υψηλότερες βαθμολογίες")
df_largest = df_original.nlargest(3, 'Score')
print(df_largest[['Name', 'Score']])
print()

# 1.4 nsmallest() - Οι μικρότερες τιμές
print("\n1.4 nsmallest(2, 'Years') - 2 χαμηλότερα χρόνια προϋπηρεσίας")
df_smallest = df_original.nsmallest(2, 'Years')
print(df_smallest[['Name', 'Years']])
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 2. ΑΝΑΔΙΑΤΑΞΗ & ΑΛΛΑΓΗ ΣΧΗΜΑΤΟΣ (RESHAPING & REARRANGING)
# ----------------------------------------------------------------------
print("2. ΑΝΑΔΙΑΤΑΞΗ & ΑΛΛΑΓΗ ΣΧΗΜΑΤΟΣ")
print("=" * 60)

# 2.1 reset_index() - Επαναφορά δείκτη
print("\n2.1 reset_index() - Επαναφορά δείκτη")
# Δημιουργούμε DataFrame με ορισμένο index
df_with_index = df_original.set_index('ID')
print("DataFrame με index = ID:")
print(df_with_index.head())
df_reset = df_with_index.reset_index()
print("Μετά από reset_index():")
print(df_reset.head())
print("reset_index(drop=True) - Χωρίς διατήρηση παλιού index")
df_reset_drop = df_with_index.reset_index(drop=True)
print(df_reset_drop.head())
print()

# 2.2 set_index() - Ορισμός στήλης ως index
print("\n2.2 set_index() - Ορισμός στήλης 'Name' ως index")
df_set = df_original.set_index('Name')
print(df_set.head())
print()

# 2.3 reindex() - Προσαρμογή σε νέες ετικέτες
print("\n2.3 reindex() - Νέο ευρετήριο γραμμών")
```

```
df_reindex = df_original.set_index('ID').reindex([101, 103, 105, 107, 109,
999])
print(df_reindex[['Name', 'Salary']])  # 999 δεν υπάρχει, θα έχει NaN
print()

print("reindex() με fill_value")
df_reindex_filled = df_original.set_index('ID').reindex([101, 103, 105,
107, 109, 999], fill_value=0)
print(df_reindex_filled[['Name', 'Salary']])
print()

# 2.4 melt() - Μετατροπή από wide σε long
print("\n2.4 melt() - Wide σε long format")
# Δημιουργούμε ένα wide DataFrame με πωλήσεις ανά έτος
wide_df = pd.DataFrame({
    'id': [1, 2, 3],
    'name': ['A', 'B', 'C'],
    '2020_sales': [100, 200, 300],
    '2021_sales': [150, 250, 350],
    '2022_sales': [200, 300, 400]
})
print("Wide DataFrame:")
print(wide_df)
melted = pd.melt(wide_df, id_vars=['id', 'name'], var_name='year',
value_name='sales')
print("Μετά από melt() - Long format:")
print(melted)
print()

# 2.5 pivot() - Αντίστροφο του melt (long to wide)
print("\n2.5 pivot() - Long σε wide format (αντίστροφο melt)")
pivoted = melted.pivot(index=['id', 'name'], columns='year',
values='sales').reset_index()
print("Μετά από pivot() - Επιστροφή σε wide (με multiindex στηλών):")
print(pivoted)
# Για να γίνει όπως το αρχικό, ίσως χρειαστεί rename στηλών
pivoted.columns.name = None  # Αφαίρεση ονόματος από columns
print(pivoted)
print()

# 2.6 pivot_table() - Συνοπτικός πίνακας με συνάθροιση
print("\n2.6 pivot_table() - Συνοπτικός πίνακας με μέσο μισθό ανά Τμήμα")
# Δημιουργούμε DataFrame με διπλότυπα για να φανεί η συνάθροιση
dup_data = {
    'Dept': ['Sales', 'Sales', 'IT', 'IT', 'HR', 'HR', 'Sales'],
    'Emp': ['A', 'B', 'C', 'D', 'E', 'F', 'G'],
    'Salary': [50000, 55000, 60000, 62000, 65000, 63000, 58000]
}
df_dup = pd.DataFrame(dup_data)
print("Δεδομένα με διπλότυπα Dept:")
print(df_dup)
pt = pd.pivot_table(df_dup, values='Salary', index='Dept', aggfunc='mean')
print("pivot_table - Μέσος μισθός ανά Dept:")
print(pt)
print()

# 2.7 stack() - Συμπίεση στηλών σε γραμμές
print("\n2.7 stack() - Συμπίεση επιπέδου στηλών")
# Δημιουργούμε DataFrame με MultiIndex στις στήλες
cols = pd.MultiIndex.from_tuples([('Math', 'Score'), ('Math', 'Grade'),
('Science', 'Score'), ('Science', 'Grade')])
```

```
df_stack = pd.DataFrame(np.random.randint(50, 100, size=(3, 4)),
columns=cols, index=['Student1', 'Student2', 'Student3'])
print("DataFrame με MultiIndex στις στήλες:")
print(df_stack)
stacked = df_stack.stack()
print("Μετά από stack() (οι στήλες γίνονται index):")
print(stacked)
print()

# 2.8 unstack() - Αντίστροφο του stack
print("\n2.8 unstack() - Επέκταση index σε στήλες")
unstacked = stacked.unstack()
print("Μετά από unstack() (επιστροφή στο αρχικό σχήμα):")
print(unstacked)
print()

# 2.9 rename() - Μετονομασία στηλών ή index
print("\n2.9 rename() - Μετονομασία στηλών")
df_renamed = df_original.rename(columns={'Name': 'Employee', 'Salary':
'AnnualSalary'})
print(df_renamed.head())
print()

# 2.10 set_axis() - Αντικατάσταση ολόκληρου άξονα
print("\n2.10 set_axis() - Αντικατάσταση ετικετών στηλών")
df_axis = df_original.copy()
new_columns = ['ID', 'EMPLOYEE', 'DEPT', 'SALARY', 'YEARS', 'SCORE',
'JOIN_DATE']
# Διόρθωση: είτε με set_axis και ανάθεση, είτε απευθείας με columns
df_axis.columns = new_columns
print("DataFrame με νέες ετικέτες στηλών:")
print(df_axis.head())
print()

print("\n" + "="*100 + "\n")
print("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
```

Κ. 3.4.5: Παράδειγμα για τις μεθόδους ταξινόμηση & αναδιάταξη των Πλαισίων Δεδομένων

3.4.6. Επεξεργασία NaN Τιμών

Μέθοδος	Περιγραφή	Παράδειγμα
Ανίχνευση & Πληροφόρηση		
isna() (ή isnull())	Επιστρέφει ένα νέο DataFrame με boolean τιμές (True/False), όπου True υποδηλώνει την ύπαρξη ελλείπουσας τιμής (NaN ή None) στο αντίστοιχο κελί. Είναι το πρώτο βήμα για τον εντοπισμό των κενών δεδομένων .	import pandas as pd import numpy as np df = pd.DataFrame({'A': [1, np.nan, 3], 'B': [4, 5, np.nan]}) print(df.isna()) # A B # 0 False False # 1 True False # 2 False True
Διαγραφή		

Μέθοδος	Περιγραφή	Παράδειγμα

dropna()	Διαγράφει γραμμές (axis=0) ή στήλες (axis=1) που περιέχουν τιμές NaN. Η παράμετρος how καθορίζει αν θα διαγραφούν σειρές με *οποιοδήποτε* (any) NaN ή μόνο αν *όλα* (all) τα στοιχεία είναι NaN. Με την παράμετρο subset μπορείτε να ελέγξετε συγκεκριμένες στήλες .	# Διαγραφή γραμμών με τουλάχιστον ένα NaN
		df_clean = df.dropna()
		print(df_clean)
		# A B
		# 0 1.0 4.0
		
		# Διαγραφή γραμμών μόνο αν η στήλη 'A' έχει NaN
		df_clean_sub = df.dropna(subset=['A'])
		print(df_clean_sub)
		# A B
		# 0 1.0 4.0
		# 2 3.0 NaN

Συμπλήρωση		

fillna()	Αντικαθιστά τις τιμές NaN με μια συγκεκριμένη τιμή ή με μια τιμή που προκύπτει από μια μέθοδο. Μπορεί να συμπληρώσει με σταθερή τιμή (π.χ. 0, "Άγνωστο"), με στατιστικό μέτρο (μέσος όρος, διάμεσος) ή με χρήση μεθόδου γειτνίασης .	# Συμπλήρωση όλων των NaN με το 0
		df_filled = df.fillna(0)
		print(df_filled)
		# A B
		# 0 1.0 4.0
		# 1 0.0 5.0
		# 2 3.0 0.0
		
		# Συμπλήρωση με το μέσο όρο της στήλης
		df['A'].fillna(df['A'].mean(), inplace=True)

ffill() (forward fill)	Συμπληρώνει τα NaN με την προηγούμενη έγκυρη τιμή κατά μήκος των γραμμών (axis=0) ή στηλών (axis=1). Είναι εξειδικευμένη μέθοδος του fillna(method='ffill') .	# Προωθητική συμπλήρωση (προς τα κάτω)
		df_ffill = df.ffill()
		print(df_ffill)
		# A B
		# 0 1.0 4.0
		# 1 1.0 5.0 (το NaN στη A έγινε 1.0)
		# 2 3.0 5.0 (το NaN στη B έγινε 5.0)

Μέθοδος	Περιγραφή	Παράδειγμα

bfill() (backward fill)	Συμπληρώνει τα NaN με την *επόμενη* έγκυρη τιμή. Είναι εξειδικευμένη μέθοδος του fillna(method='bfill') .	# Οπισθοχωρητική συμπλήρωση (προς τα πάνω)
		df_bfill = df.bfill()
		print(df_bfill)
		# A B
		# 0 1.0 4.0
		# 1 3.0 5.0 (το NaN στη A έγινε 3.0 από την επόμενη γραμμή)
		# 2 3.0 NaN (το NaN στη B παραμένει αν δεν υπάρχει επόμενη τιμή)

interpolate()	Πραγματοποιεί παρεμβολή (interpolation) για την εκτίμηση και συμπλήρωση των NaN τιμών. Χρησιμοποιεί διάφορες μεθόδους όπως γραμμική (linear), πολυωνυμική, κ.λπ., λαμβάνοντας υπόψη την τάση των δεδομένων .	# Γραμμική παρεμβολή
		df_interp = df.interpolate()
		print(df_interp)
		# A B
		# 0 1.0 4.0
		# 1 2.0 5.0 (το NaN στη A έγινε 2.0, ως ο μέσος του 1.0 και 3.0)
		# 2 3.0 NaN (η παρεμβολή στη B δεν μπορεί να γίνει για το τελευταίο στοιχείο)

Πίνακας 3.4.8: Μέθοδοι για επεξεργασία NaN τιμών για Πλαίσια Δεδομένων

1. Ανίχνευση: Πού είναι τα κενά;

Πριν διορθώσεις, πρέπει να μετρήσεις.

- **isna()**: Παρόλο που επιστρέφει έναν πίνακα από True/False, σπάνια τον διαβάζουμε έτσι.

Pro Tip: Συνήθως χρησιμοποιούμε το df.isna().sum(). Αυτό σου δίνει αμέσως τον αριθμό των κενών τιμών ανά στήλη, βοηθώντας σε να αποφασίσεις αν μια στήλη είναι "άχρηστη" (π.χ. αν λείπει το 90% των δεδομένων).

2. Διαγραφή: Η λύση της "σκούπας"

Η dropna() είναι η πιο γρήγορη λύση, αλλά θέλει προσοχή για να μη χάσεις πολύτιμη πληροφορία.

- **subset**: Είναι η πιο ασφαλής παράμετρος. Αντί να διαγράψεις μια ολόκληρη γραμμή επειδή λείπει μια δευτερεύουσα πληροφορία, τη διαγράφεις μόνο αν λείπει ένα "ζωτικό" στοιχείο (π.χ. subset=['Email']).
- **how='all'**: Πολύ χρήσιμο όταν έχεις εντελώς κενές γραμμές στο τέλος ενός αρχείου Excel.

3. Συμπλήρωση: Η τέχνη του Imputation

Αν δεν θέλεις να διαγράψεις δεδομένα, πρέπει να τα "μαντέψεις" ή να τα αντικαταστήσεις.

- **fillna() με στατιστικά:** Η χρήση του μέσου όρου (mean()) ή της διαμέσου (median()) είναι η κλασική στατιστική προσέγγιση.

- **ffill() (Forward Fill):** Σκέψου ένα αρχείο Excel όπου το όνομα μιας εταιρείας γράφεται μόνο στην πρώτη γραμμή και οι επόμενες είναι κενές μέχρι να αλλάξει η εταιρεία. Το ffill "σπρώχνει" την τιμή προς τα κάτω για να γεμίσει τα κενά.
- **bfill() (Backward Fill):** Το αντίστροφο. Χρήσιμο όταν η επόμενη τιμή είναι αυτή που ορίζει το πλαίσιο.

4. Interpolate: Η "έξυπνη" συμπλήρωση

Η interpolate() δεν βάζει απλώς μια σταθερή τιμή, αλλά προσπαθεί να καταλάβει τη **διαδρομή** των δεδομένων.

- **Γραμμική Παρεμβολή:** Αν έχεις τις τιμές 10 (στη γραμμή 1) και 20 (στη γραμμή 3), το interpolate() θα βάλει 15 στη γραμμή 2. Είναι ιδανικό για χρονοσειρές (time-series) ή δεδομένα αισθητήρων όπου οι τιμές αλλάζουν σταδιακά.

```
import pandas as pd
import numpy as np

# ---------------------------------------------------------------------
# Δημιουργία DataFrame με ελλιπείς τιμές (NaN)
# ---------------------------------------------------------------------
print("ΔΗΜΙΟΥΡΓΙΑ ΔΕΙΓΜΑΤΟΣ DataFrame ΜΕ NaN")
print("-" * 60)

# Δημιουργούμε ένα μικρό DataFrame με σκόπιμα τοποθετημένα NaN
df = pd.DataFrame({
    'A': [1, np.nan, 3, 4, np.nan],
    'B': [4, 5, np.nan, 7, 8],
    'C': [np.nan, 2, 3, np.nan, 5],
    'D': [10, 11, 12, 13, 14]
}, index=['row0', 'row1', 'row2', 'row3', 'row4'])

print("Αρχικό DataFrame (με NaN):")
print(df)
print("\n" + "="*80 + "\n")

# ---------------------------------------------------------------------
# 1. ΑΝΙΧΝΕΥΣΗ & ΠΛΗΡΟΦΟΡΗΣΗ - isna() / isnull()
# ---------------------------------------------------------------------
print("1. ΑΝΙΧΝΕΥΣΗ ΕΛΛΕΙΠΟΥΣΩΝ ΤΙΜΩΝ")
print("-" * 60)

# 1.1 isna() - Επιστρέφει True για κενές τιμές
print("1.1 isna() - Boolean πίνακας με True όπου υπάρχει NaN:")
print(df.isna())
print()

# 1.2 isnull() - Ακριβώς το ίδιο με isna() (συνώνυμο)
print("1.2 isnull() - Ίδιο αποτέλεσμα:")
print(df.isnull())
print()

# 1.3 Συνοπτική εικόνα - sum() για αριθμό NaN ανά στήλη
print("1.3 Πλήθος NaN ανά στήλη:")
print(df.isna().sum())
print()
```

```
# 1.4 Σύνολο NaN στο DataFrame
print("1.4 Συνολικό πλήθος NaN στο DataFrame:")
print(df.isna().sum().sum())
print("\n" + "="*80 + "\n")

# ------------------------------------------------------------------
# 2. ΔΙΑΓΡΑΦΗ - dropna()
# ------------------------------------------------------------------
print("2. ΔΙΑΓΡΑΦΗ ΓΡΑΜΜΩΝ/ΣΤΗΛΩΝ ΜΕ NaN")
print("-" * 60)

# 2.1 Διαγραφή γραμμών που περιέχουν οποιοδήποτε NaN (προεπιλογή axis=0,
how='any')
print("2.1 dropna() - Διαγραφή γραμμών με τουλάχιστον ένα NaN:")
df_drop_any = df.dropna()
print(df_drop_any)
print()

# 2.2 Διαγραφή γραμμών μόνο αν ΟΛΑ τα στοιχεία είναι NaN (how='all')
print("2.2 dropna(how='all') - Διαγραφή μόνο γραμμών που είναι εξ ολοκλήρου
NaN:")
# Δημιουργούμε μια γραμμή με όλα NaN
df_with_all_nan = pd.DataFrame({
    'A': [1, np.nan, np.nan],
    'B': [2, np.nan, np.nan],
    'C': [3, np.nan, np.nan]
}, index=['r1', 'r2', 'r3'])
print("Πριν:")
print(df_with_all_nan)
df_drop_all = df_with_all_nan.dropna(how='all')
print("Μετά:")
print(df_drop_all)
print()

# 2.3 Διαγραφή στηλών που περιέχουν NaN (axis=1)
print("2.3 dropna(axis=1) - Διαγραφή στηλών με τουλάχιστον ένα NaN:")
df_drop_cols = df.dropna(axis=1)
print(df_drop_cols)
print()

# 2.4 Διαγραφή γραμμών με NaN μόνο σε συγκεκριμένες στήλες (subset)
print("2.4 dropna(subset=['A', 'B']) - Διαγραφή γραμμών μόνο αν υπάρχει NaN
στις στήλες Α ή Β:")
df_drop_subset = df.dropna(subset=['A', 'B'])
print(df_drop_subset)
print("\n" + "="*80 + "\n")

# ------------------------------------------------------------------
# 3. ΣΥΜΠΛΗΡΩΣΗ - fillna()
# ------------------------------------------------------------------
print("3. ΣΥΜΠΛΗΡΩΣΗ ΜΕ ΣΤΑΘΕΡΗ ΤΙΜΗ")
print("-" * 60)

# 3.1 Συμπλήρωση όλων των NaN με σταθερή τιμή (π.χ. 0)
print("3.1 fillna(0) - Συμπλήρωση όλων των NaN με 0:")
df_fill_zero = df.fillna(0)
print(df_fill_zero)
print()

# 3.2 Συμπλήρωση με σταθερή τιμή ανά στήλη (π.χ. dictionary)
```

```
print("3.2 fillna({'A': -1, 'B': -2, 'C': -3}) - Διαφορετικές τιμές ανά
στήλη:")
df_fill_dict = df.fillna({'A': -1, 'B': -2, 'C': -3})
print(df_fill_dict)
print()

# 3.3 Συμπλήρωση με στατιστικό μέτρο (μέσος όρος στήλης)
print("3.3 Συμπλήρωση της στήλης 'A' με τον μέσο όρο της (προσοχή: δεν
γίνεται inplace εδώ):")
df_fill_mean = df.copy()
df_fill_mean['A'] = df_fill_mean['A'].fillna(df_fill_mean['A'].mean())
print(df_fill_mean)
print()

# 3.4 inplace=True (προσοχή: τροποποιεί το αρχικό DataFrame)
print("3.4 fillna(inplace=True) - Παράδειγμα με inplace (τροποποιεί το
αρχικό):")
df_inplace = df.copy()
df_inplace.fillna(99, inplace=True)
print(df_inplace)
print("\n" + "="*80 + "\n")

# ------------------------------------------------------------------
# 4. ΠΡΟΩΘΗΤΙΚΗ ΣΥΜΠΛΗΡΩΣΗ - ffill()
# ------------------------------------------------------------------
print("4. ΠΡΟΩΘΗΤΙΚΗ ΣΥΜΠΛΗΡΩΣΗ (FORWARD FILL)")
print("-" * 60)

# 4.1 ffill() - Συμπλήρωση με την προηγούμενη έγκυρη τιμή (προς τα κάτω)
print("4.1 ffill() - Προωθητική συμπλήρωση κατά μήκος γραμμών:")
df_ffill = df.ffill()
print(df_ffill)
print()

# 4.2 ffill(axis=1) - Προωθητική συμπλήρωση κατά μήκος στηλών (οριζόντια)
print("4.2 ffill(axis=1) - Οριζόντια προωθητική συμπλήρωση:")
df_ffill_horiz = df.ffill(axis=1)
print(df_ffill_horiz)
print("\n" + "="*80 + "\n")

# ------------------------------------------------------------------
# 5. ΟΠΙΣΘΟΧΩΡΗΤΙΚΗ ΣΥΜΠΛΗΡΩΣΗ - bfill()
# ------------------------------------------------------------------
print("5. ΟΠΙΣΘΟΧΩΡΗΤΙΚΗ ΣΥΜΠΛΗΡΩΣΗ (BACKWARD FILL)")
print("-" * 60)

# 5.1 bfill() - Συμπλήρωση με την επόμενη έγκυρη τιμή (προς τα πάνω)
print("5.1 bfill() - Οπισθοχωρητική συμπλήρωση:")
df_bfill = df.bfill()
print(df_bfill)
print()

# 5.2 bfill(axis=1) - Οριζόντια οπισθοχωρητική συμπλήρωση
print("5.2 bfill(axis=1) - Οριζόντια οπισθοχωρητική συμπλήρωση:")
df_bfill_horiz = df.bfill(axis=1)
print(df_bfill_horiz)
print("\n" + "="*80 + "\n")

# ------------------------------------------------------------------
# 6. ΠΑΡΕΜΒΟΛΗ - interpolate()
# ------------------------------------------------------------------
```

```
print("6. ΠΑΡΕΜΒΟΛΗ (INTERPOLATION)")
print("-" * 60)

# 6.1 Γραμμική παρεμβολή (linear) - προεπιλογή
print("6.1 interpolate() - Γραμμική παρεμβολή (προεπιλογή):")
df_interp_linear = df.interpolate()
print(df_interp_linear)
print()

# 6.2 Παρεμβολή με περιορισμό στην κατεύθυνση (π.χ. μόνο προς τα εμπρός)
print("6.2 interpolate(limit_direction='forward') - Μόνο προς τα εμπρός:")
df_interp_forward = df.interpolate(limit_direction='forward')
print(df_interp_forward)
print()

# 6.3 Παρεμβολή με άλλη μέθοδο (π.χ. πολυωνυμική, απαιτεί εγκατάσταση
scipy)
# Προαιρετικά: αν θέλετε να δοκιμάσετε polynomial ή spline, χρειάζεται
scipy.
# Εδώ το αφήνουμε σχολιασμένο για αποφυγή εξαρτήσεων.
# print("6.3 interpolate(method='polynomial', order=2) - Πολυωνυμική
παρεμβολή 2ου βαθμού:")
# df_interp_poly = df.interpolate(method='polynomial', order=2)
# print(df_interp_poly)
print("\n" + "="*80 + "\n")

# ---------------------------------------------------------------------
# 7. ΣΥΓΚΕΝΤΡΩΤΙΚΗ ΕΠΙΣΚΟΠΗΣΗ ΜΕΘΟΔΩΝ
# ---------------------------------------------------------------------
print("7. ΣΥΓΚΕΝΤΡΩΤΙΚΗ ΕΠΙΣΚΟΠΗΣΗ")
print("-" * 60)
print("Αρχικό DataFrame (για υπενθύμιση):")
print(df)

print("\n--- isna() ---")
print(df.isna())

print("\n--- dropna() ---")
print(df.dropna())

print("\n--- fillna(0) ---")
print(df.fillna(0))

print("\n--- ffill() ---")
print(df.ffill())

print("\n--- bfill() ---")
print(df.bfill())

print("\n--- interpolate() ---")
print(df.interpolate())

print("\n" + "="*80 + "\n")
print("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
```

Κ. 3.4.6: Παράδειγμα για τις μεθόδους επεξεργασίας NaN τιμών των Πλαισίων Δεδομένων

3.4.7. Μετασχηματισμοί Τύπων & Τιμών

Μέθοδος	Περιγραφή	Παράδειγμα
astype()	Μετατρέπει τον τύπο δεδομένων μιας στήλης ή ολόκληρου του DataFrame. Μπορείς να ορίσεις διαφορετικό τύπο για κάθε στήλη χρησιμοποιώντας ένα λεξικό .	df['ηλικία'] = df['ηλικία'].astype(int) df = df.astype({'ύψος': float, 'όνομα': 'category'})
pd.to_numeric()	Μετατρέπει ένα όρισμα (π.χ., μια στήλη) σε αριθμητικό τύπο. Είναι πιο ευέλικτο από το astype() γιατί μπορεί να χειριστεί σφάλματα, μετατρέποντας μη έγκυρες τιμές σε NaN (με errors='coerce') και να μειώσει την κατανάλωση μνήμης .	df['τιμή'] = pd.to_numeric(df['τιμή'], errors='coerce') df['εισόδημα'] = pd.to_numeric(df['εισόδημα'], downcast='float')
convert_dtypes()	Μετατρέπει αυτόματα τις στήλες στον βέλτιστο τύπο δεδομένων που υποστηρίζει η Pandas για missing values, χρησιμοποιώντας την εσωτερική υποδομή Nullable (π.χ., Int64, string) .	df_converted = df.convert_dtypes()
infer_objects()	Προσπαθεί να βελτιστοποιήσει τους τύπους δεδομένων σε στήλες τύπου object (που συχνά περιέχουν μικτές ή μη-βέλτιστες μορφές), μετατρέποντάς τες σε πιο συγκεκριμένους τύπους (π.χ., int, float, datetime) όπου είναι δυνατόν .	df_inferred = df.infer_objects()
map()	Χρησιμοποιείται για να αντικαταστήσει κάθε τιμή σε μια **Series** (στήλη) με βάση μια αντιστοίχιση (λεξικό) ή μια συνάρτηση. Δεν λειτουργεί απευθείας σε DataFrame .	df['φύλο'] = df['φύλο'].map({'female': 1, 'male': 0}) df['τετράγωνο'] = df['αριθμός'].map(lambda x: x**2)
replace()	Αντικαθιστά συγκεκριμένες τιμές σε ένα DataFrame ή Series με άλλες τιμές. Μπορεί να δουλέψει με μεμονωμένες τιμές, λίστες ή λεξικά .	df.replace(0, 100) df.replace({'NA': pd.NA, 'unknown': None})
where()	Διατηρεί τις αρχικές τιμές όπου μια συνθήκη είναι **True**, αλλιώς τις αντικαθιστά με μια άλλη τιμή (προεπιλογή NaN) .	df.where(df >= 0, 0) *Αντικαθιστά όλες τις αρνητικές τιμές με 0.*
mask()	Το αντίθετο του where(). Αντικαθιστά τις τιμές όπου μια συνθήκη είναι **True** .	df.mask(df < 0, 0) *Κάνει ακριβώς το ίδιο με το παραπάνω παράδειγμα where() - αντικαθιστά τις αρνητικές τιμές με 0.*
transform()	Εφαρμόζει μια συνάρτηση σε κάθε τιμή του DataFrame, επιστρέφοντας ένα νέο DataFrame με τον ίδιο αριθμό γραμμών. Μπορεί να εφαρμόσει πολλές συναρτήσεις ταυτόχρονα .	df.transform(lambda x: x * 100) df.transform([np.sqrt, np.exp])

Μέθοδος	Περιγραφή	Παράδειγμα
apply()	Εφαρμόζει μια συνάρτηση κατά μήκος ενός άξονα (στήλες ή γραμμές) του DataFrame. Είναι πιο γενικό από το transform() και μπορεί να επιστρέψει μια διαφορετική μορφή .	df.apply(sum) *Άθροισμα κάθε στήλης.*
applymap()	Εφαρμόζει μια συνάρτηση σε κάθε μεμονωμένο στοιχείο του DataFrame (element-wise) .	df.applymap(lambda x: f"{x:.2f}") *Μορφοποιεί όλες τις αριθμητικές τιμές με 2 δεκαδικά.*
fillna()	Αντικαθιστά τις τιμές που λείπουν (NaN / NA) με μια συγκεκριμένη τιμή ή με μια μέθοδο γεμίσματος (π.χ., προώθηση της τελευταίας γνωστής τιμής) .	df.fillna(0) df.fillna(method='ffill') *Γεμίζει τα κενά προωθώντας την προηγούμενη τιμή.*
loc[] **με Συνθήκη**	Η χρήση του loc με μια boolean συνθήκη είναι ένας πολύ συνηθισμένος και ισχυρός τρόπος για αντικατάσταση τιμών υπό συνθήκη .	df.loc[df['ηλικία'] > 60, 'κατηγορία_ηλικίας'] = 'Senior' *Σε όσους έχουν ηλικία >60, ορίζει την τιμή της στήλης 'κατηγορία_ηλικίας' σε 'Senior'.*

Κ. 3.4.7: Παράδειγμα για τις μεθόδους NaN τιμών για Πλαίσια Δεδομένων

Σε αυτή την ενότητα εξετάζουμε τη **Μεταμόρφωση Δεδομένων (Data Transformation)**. Εδώ είναι που μετατρέπουμε τις ακατέργαστες πληροφορίες σε μια μορφή έτοιμη για μαθηματικά μοντέλα ή στατιστική ανάλυση.

1. Διαχείριση Τύπων Δεδομένων (Dtypes)

Η σωστή επιλογή τύπου δεδομένων είναι κρίσιμη για τη μνήμη και την ορθή εκτέλεση των πράξεων.

- **astype() vs pd.to_numeric()**:
    - Η astype() είναι αυστηρή. Αν προσπαθήσεις να μετατρέψεις τη στήλη ['1', '2', 'λάθος'] σε integer, θα κρασάρει.
    - Η **pd.to_numeric()** είναι η "έξυπνη" εναλλακτική. Με το errors='coerce', μετατρέπει το 'λάθος' σε NaN, επιτρέποντάς σου να συνεχίσεις τη δουλειά σου χωρίς διακοπές.
- **convert_dtypes()**: Εισήχθη πρόσφατα στην Pandas για να λύσει το πρόβλημα των "Nullable Integers". Παραδοσιακά, αν μια στήλη ακεραίων είχε έστω και ένα κενό (NaN), η Pandas τη μετέτρεπε σε float. Η convert_dtypes() επιτρέπει τη διατήρηση των ακεραίων ακόμα και με κενά.

2. Αντικατάσταση & Χαρτογράφηση

- **map()**: Ιδανικό για κωδικοποίηση κατηγορικών δεδομένων (π.χ. μετατροπή "Ναι"/"Όχι" σε 1/0). Λειτουργεί μόνο σε Series.
- **replace()**: Πιο ευέλικτο από το map, καθώς μπορείς να αντικαταστήσεις συγκεκριμένες τιμές σε ολόκληρο το DataFrame χωρίς να επηρεάσεις τις υπόλοιπες.
- **where() & mask()**: Λειτουργούν σαν το "IF" του Excel.
    - where(συνθήκη, τιμή): Κράτα την τιμή αν ισχύει η συνθήκη, αλλιώς άλλαξέ την.

- mask(συνθήκη, τιμή): Άλλαξε την τιμή αν ισχύει η συνθήκη.

3. Η "Τριάδα" των Functions: apply, transform, applymap

Εδώ συχνά μπερδεύονται οι χρήστες. Ας δούμε τη διαφορά τους:

1. **apply()**: Λειτουργεί ανά άξονα. Μπορεί να πάρει μια στήλη και να την επιστρέψει ως μία μόνο τιμή (π.χ. sum).
2. **transform()**: Πρέπει πάντα να επιστρέφει ένα αποτέλεσμα με το **ίδιο μήκος** όπως το αρχικό. Είναι εξαιρετικό για normalization (π.χ. να διαιρέσεις κάθε τιμή με το σύνολο της στήλης της).
3. **applymap()** (στις νεότερες εκδόσεις μετονομάστηκε σε map για DataFrames): Εφαρμόζει τη συνάρτηση σε **κάθε κελί ξεχωριστά**. Χρήσιμο για μορφοποίηση (π.χ. προσθήκη του συμβόλου "€" σε κάθε αριθμό).

4. Δημιουργία Στηλών υπό Συνθήκη

- **df.loc[συνθήκη, 'νέα_στήλη'] = 'τιμή'**: Αυτός είναι ο πιο ασφαλής και "Pandas-friendly" τρόπος να δημιουργήσεις μια νέα στήλη βασισμένη σε κριτήρια. Είναι ταχύτερος και πιο ευανάγνωστος από το να χρησιμοποιείς apply με if-else συναρτήσεις.

```
import pandas as pd
import numpy as np

# ---------------------------------------------------------------
# Δημιουργία δείγματος DataFrame με ποικιλία τύπων και τιμών
# ---------------------------------------------------------------
print("ΔΗΜΙΟΥΡΓΙΑ ΔΕΙΓΜΑΤΟΣ DataFrame")
print("-" * 60)

data = {
    'Name': ['Anna', 'George', 'Maria', 'Nikos', 'Eleni'],
    'Age': ['25', '30', '28', '35', '22'],        # αρχικά ως string
    'Height': [1.65, 1.80, 1.70, 1.75, 1.68],
    'Salary': [50000, 60000, None, 65000, 48000],  # None (NaN)
    'Department': ['Sales', 'IT', 'Sales', 'HR', 'IT'],
    'Join_Date': ['2020-01-15', '2019-03-20', '2021-07-01', '2018-11-05',
'2022-02-10'],
    'Score': [85, 92, 78, 88, 95],
    'Is_FullTime': ['yes', 'no', 'yes', 'yes', 'no']
}
df = pd.DataFrame(data)
print("Αρχικό DataFrame:")
print(df)
print("\nΠληροφορίες τύπων:")
print(df.dtypes)
print("\n" + "="*100 + "\n")

# ---------------------------------------------------------------
# 1. astype() - Μετατροπή τύπου δεδομένων
# ---------------------------------------------------------------
print("1. astype() - Μετατροπή τύπου δεδομένων")
print("-" * 60)

# Μετατροπή μιας στήλης από string σε int
df['Age'] = df['Age'].astype(int)
print("Μετά από astype(int) στη στήλη 'Age':")
print(df[['Name', 'Age']].head())
print()
```

```python
# Μετατροπή πολλών στηλών με λεξικό
df = df.astype({'Height': 'float32', 'Is_FullTime': 'category'})
print("Μετά από astype({'Height': 'float32', 'Is_FullTime': 'category'}):")
print(df.dtypes)
print("\n" + "="*100 + "\n")

# ------------------------------------------------------------------
# 2. pd.to_numeric() - Μετατροπή σε αριθμητικό τύπο με χειρισμό σφαλμάτων
# ------------------------------------------------------------------
print("2. pd.to_numeric() - Ευέλικτη μετατροπή σε αριθμό")
print("-" * 60)

# Δημιουργούμε μια στήλη με ανάμεικτες τιμές
df['Temp'] = ['35.5', '40.2', 'abc', '38.0', '37.5']
print("Στήλη 'Temp' πριν:")
print(df['Temp'])

# Μετατροπή με errors='coerce' (οι μη έγκυρες γίνονται NaN)
df['Temp'] = pd.to_numeric(df['Temp'], errors='coerce')
print("Μετά από pd.to_numeric(errors='coerce'):")
print(df['Temp'])
print()

# Χρήση downcast για εξοικονόμηση μνήμης
df['Salary'] = pd.to_numeric(df['Salary'], downcast='float')
print("Μετά από downcast='float' στη στήλη 'Salary':")
print(df['Salary'])
print("\n" + "="*100 + "\n")

# ------------------------------------------------------------------
# 3. convert_dtypes() - Αυτόματη μετατροπή σε βέλτιστους nullable τύπους
# ------------------------------------------------------------------
print("3. convert_dtypes() - Αυτόματη βελτιστοποίηση τύπων")
print("-" * 60)

df_converted = df.convert_dtypes()
print("Τύποι μετά από convert_dtypes():")
print(df_converted.dtypes)
print("\n" + "="*100 + "\n")

# ------------------------------------------------------------------
# 4. infer_objects() - Βελτιστοποίηση στηλών τύπου object
# ------------------------------------------------------------------
print("4. infer_objects() - Συμπερασμός καλύτερων τύπων για object")
print("-" * 60)

# Δημιουργούμε ένα DataFrame με στήλες object
df_obj = pd.DataFrame({'A': ['1', '2', '3'], 'B': [1.5, 2.5, 3.5]},
dtype='object')
print("Πριν infer_objects():")
print(df_obj.dtypes)

df_inferred = df_obj.infer_objects()
print("Μετά infer_objects():")
print(df_inferred.dtypes)
print("\n" + "="*100 + "\n")

# ------------------------------------------------------------------
# 5. map() - Αντικατάσταση τιμών σε Series με λεξικό ή συνάρτηση
# ------------------------------------------------------------------
```

```
print("5. map() - Αντιστοίχιση τιμών σε στήλη")
print("-" * 60)

# Με λεξικό
df['Is_FullTime_Code'] = df['Is_FullTime'].map({'yes': 1, 'no': 0})
print("map() με λεξικό (yes->1, no->0):")
print(df[['Name', 'Is_FullTime', 'Is_FullTime_Code']].head())
print()

# Με συνάρτηση lambda
df['Score_Squared'] = df['Score'].map(lambda x: x**2)
print("map() με lambda (τετράγωνο Score):")
print(df[['Name', 'Score', 'Score_Squared']].head())
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 6. replace() - Αντικατάσταση συγκεκριμένων τιμών
# ----------------------------------------------------------------------
print("6. replace() - Αντικατάσταση τιμών")
print("-" * 60)

# Αντικατάσταση μεμονωμένης τιμής
df_replace1 = df.replace('Sales', 'Πωλήσεις')
print("replace('Sales', 'Πωλήσεις') - στην πρώτη γραμμή:")
print(df_replace1[['Name', 'Department']].head())
print()

# Αντικατάσταση με λεξικό (πολλές αντικαταστάσεις)
df_replace2 = df.replace({'HR': 'Ανθρώπινο Δυναμικό', 'IT': 'Πληροφορική'})
print("replace με λεξικό για τμήματα:")
print(df_replace2[['Name', 'Department']].head())
print()

# Αντικατάσταση με λίστες
df_replace3 = df.replace([np.nan, 'yes'], [0, 1])
print("replace λίστας τιμών με λίστα αντικαταστάσεων (NaN->0, 'yes'->1):")
print(df_replace3[['Salary', 'Is_FullTime']].head())
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 7. where() - Διατήρηση τιμών όπου ισχύει συνθήκη, αντικατάσταση αλλού
# ----------------------------------------------------------------------
print("7. where() - Διατήρηση όπου True, αντικατάσταση όπου False")
print("-" * 60)

# Αντικατάσταση αρνητικών ή μη επιθυμητών τιμών (εδώ, Score < 90
αντικαθίσταται με NaN)
df_where = df.copy()
df_where['Score'] = df_where['Score'].where(df_where['Score'] >= 90,
other=np.nan)
print("where(Score >= 90) - κρατά μόνο βαθμολογίες >=90:")
print(df_where[['Name', 'Score']])
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 8. mask() - Αντικατάσταση τιμών όπου ισχύει συνθήκη (αντίθετο του where)
# ----------------------------------------------------------------------
print("8. mask() - Αντικατάσταση όπου True, διατήρηση όπου False")
print("-" * 60)

df_mask = df.copy()
```

```
df_mask['Score'] = df_mask['Score'].mask(df_mask['Score'] < 90, other=0)
print("mask(Score < 90, 0) - αντικαθιστά με 0 όσες βαθμολογίες είναι <90:")
print(df_mask[['Name', 'Score']])
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 9. transform() - Εφαρμογή συνάρτησης σε κάθε τιμή (επιστρέφει ίδιο σχήμα)
# ----------------------------------------------------------------------
print("9. transform() - Εφαρμογή συνάρτησης σε κάθε στοιχείο")
print("-" * 60)

# Εφαρμογή μιας συνάρτησης (π.χ. πολλαπλασιασμός επί 100)
df_transform = df[['Height', 'Score']].transform(lambda x: x * 100)
print("transform(lambda x: x*100) - στις στήλες Height και Score:")
print(df_transform.head())
print()

# Εφαρμογή πολλών συναρτήσεων (επιστρέφει DataFrame με MultiIndex)
df_multi = df[['Height', 'Score']].transform([np.sqrt, np.exp])
print("transform([np.sqrt, np.exp]) - επιστρέφει πολλές στήλες:")
print(df_multi.head())
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 10. apply() - Εφαρμογή συνάρτησης κατά μήκος άξονα (στήλες ή γραμμές)
# ----------------------------------------------------------------------
print("10. apply() - Εφαρμογή συνάρτησης σε άξονα")
print("-" * 60)

# Άθροισμα κάθε στήλης (μόνο για αριθμητικές)
print("apply(sum) - άθροισμα στηλών (αριθμητικές):")
print(df[['Age', 'Height', 'Salary', 'Score']].apply(sum))
print()

# Εφαρμογή custom συνάρτησης σε κάθε γραμμή (axis=1)
df['Name_Length'] = df.apply(lambda row: len(row['Name']), axis=1)
print("apply(lambda row: len(row['Name']), axis=1) - μήκος ονόματος:")
print(df[['Name', 'Name_Length']].head())
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 11. applymap() - Εφαρμογή συνάρτησης σε κάθε στοιχείο DataFrame (element-
wise)
# ----------------------------------------------------------------------
print("11. applymap() - Συνάρτηση σε κάθε κελί")
print("-" * 60)

# Μορφοποίηση αριθμητικών στηλών ως string με 2 δεκαδικά
df_formatted = df[['Height', 'Score']].applymap(lambda x: f"{x:.2f}")
print("applymap(lambda x: f'{x:.2f}') - μορφοποίηση:")
print(df_formatted.head())
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 12. fillna() - Συμπλήρωση ελλειπουσών τιμών
# ----------------------------------------------------------------------
print("12. fillna() - Γέμισμα NaN")
print("-" * 60)

# Συμπλήρωση με σταθερή τιμή
df_filled = df.fillna({'Salary': df['Salary'].mean()})
```

```
print("fillna με μέσο όρο στη Salary:")
print(df_filled[['Name', 'Salary']])
print()

# Προωθητική συμπλήρωση (ffill) - σε νέα στήλη για παράδειγμα
df['Salary_ffill'] = df['Salary'].fillna(method='ffill')
print("ffill στη Salary:")
print(df[['Name', 'Salary', 'Salary_ffill']])
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 13. loc[] με συνθήκη - Υπό συνθήκη αντικατάσταση σε συγκεκριμένες στήλες
# ----------------------------------------------------------------------
print("13. loc με συνθήκη - Αντικατάσταση υπό συνθήκη")
print("-" * 60)

# Ορισμός νέας στήλης κατηγορίας βάσει ηλικίας
df.loc[df['Age'] > 28, 'Age_Group'] = 'Senior'
df.loc[df['Age'] <= 28, 'Age_Group'] = 'Junior'
print("loc με συνθήκη για Age_Group (Senior αν Age>28):")
print(df[['Name', 'Age', 'Age_Group']])
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
print("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
```

Κ. 3.4.8: Παράδειγμα για τις μεθόδους μετασχηματισμούς τύπων και τιμών των Πλαισίων Δεδομένων

3.4.8. Στατιστικές Μέθοδοι

Μέθοδος	Περιγραφή	Επιστροφή
Περιγραφική Στατιστική		
describe()	Δημιουργεί μια συνοπτική στατιστική περιγραφή για τις αριθμητικές στήλες, που περιλαμβάνει count, mean, std, min, quartiles, max .	df.describe() df.describe(percentiles=[.1, .5, .9]) # με προσαρμοσμένα ποσοστιαία σημεία
info()	Παρέχει μια σύνοψη του DataFrame, όπως τον αριθμό των γραμμών, τα ονόματα των στηλών, τους τύπους δεδομένων και τη χρήση μνήμης .	df.info()
Μέτρα Κεντρικής Τάσης		
mean()	Υπολογίζει τον μέσο όρο των τιμών .	df.mean() # Μέσος όρος κάθε στήλης df["Revenue"].mean() # Μέσος όρος συγκεκριμένης στήλης
median()	Υπολογίζει τη διάμεσο (το μεσαίο) των τιμών .	df.median()
mode()	Επιστρέφει την επικρατούσα τιμή (τιμή ή τιμές που εμφανίζονται πιο συχνά) .	df.mode()
Μέτρα Διασποράς		
std()	Υπολογίζει την τυπική απόκλιση, που δείχνει τη διασπορά των τιμών γύρω από τον μέσο όρο .	df.std()
var()	Υπολογίζει τη διακύμανση .	df.var()

min()	Επιστρέφει την ελάχιστη τιμή .	df.min()
max()	Επιστρέφει τη μέγιστη τιμή .	df.max()
quantile()	Επιστρέφει την τιμή στο συγκεκριμένο ποσοστημόριο (π.χ., 0.25 για το 25%) .	df.quantile(0.25) # 1ο τεταρτημόριο
Συσσωρευτικά Μέτρα		
cumsum()	Υπολογίζει το αθροιστικό άθροισμα των τιμών .	df.cumsum()
cumprod()	Υπολογίζει το αθροιστικό γινόμενο των τιμών .	df.cumprod()
cummax()	Επιστρέφει το αθροιστικό μέγιστο των τιμών .	df.cummax()
cummin()	Επιστρέφει το αθροιστικό ελάχιστο των τιμών .	df.cummin()
Συσχέτιση & Συνδιακύμανση		
corr()	Υπολογίζει τη συσχέτιση (π.χ., Pearson) μεταξύ των στηλών .	df.corr()
cov()	Υπολογίζει τη συνδιακύμανση μεταξύ των στηλών .	df.cov()
Μοναδικές & Απαρίθμηση Τιμών		
count()	Επιστρέφει τον αριθμό των μη-κενών (non-NA) τιμών σε κάθε στήλη/σειρά .	df.count()
nunique()	Επιστρέφει τον αριθμό των μοναδικών τιμών .	df.nunique()
value_counts()	Επιστρέφει μια σειρά (Series) που περιέχει την απαρίθμηση των μοναδικών τιμών . (Συνήθως χρησιμοποιείται σε στήλες)	df["Product"].value_counts()
Ειδικές Στατιστικές		
abs()	Επιστρέφει ένα DataFrame με την απόλυτη τιμή κάθε στοιχείου .	df.abs()
skew()	Υπολογίζει τη λοξότητα (ασυμμετρία) της κατανομής των τιμών .	df.skew()
kurtosis()	Υπολογίζει την κύρτωση (μέτρο του "αιχμηρού" μιας κατανομής) των τιμών .	df.kurtosis()
diff()	Υπολογίζει τη διαφορά μεταξύ ενός στοιχείου και ενός άλλου (συνήθως του προηγούμενου) .	df.diff()
pct_change()	Υπολογίζει την ποσοστιαία μεταβολή μεταξύ του τρέχοντος και ενός προηγούμενου στοιχείου .	df.pct_change()

Πίνακας 3.4.9: Μέθοδοι για Στατιστική επεξεργασία των Πλαισίων Δεδομένων

Σε αυτήν την ενότητα περνάμε στην **Περιγραφική Στατιστική και Ανάλυση**. Αυτές οι μέθοδοι είναι που μετατρέπουν ένα DataFrame από έναν απλό πίνακα δεδομένων σε ένα εργαλείο λήψης αποφάσεων, αποκαλύπτοντας τάσεις, αποκλίσεις και σχέσεις μεταξύ των μεταβλητών.

1. Η Μεγάλη Εικόνα (Exploratory Data Analysis)

Πριν εμβαθύνεις, χρειάζεσαι μια γρήγορη ακτινογραφία των δεδομένων σου.

- **describe()**: Είναι ο "ελβετικός σουγιάς" της στατιστικής.
    - **Tip:** Από προεπιλογή δείχνει μόνο αριθμητικές στήλες. Αν θες να δεις στατιστικά για κείμενο (όπως η πιο συχνή τιμή), χρησιμοποίησε το df.describe(include='all').
- **value_counts()**: Ίσως η πιο συχνά χρησιμοποιούμενη μέθοδος για κατηγορικές στήλες. Σου λέει αμέσως "πόσοι πελάτες είναι από την Αθήνα" ή "πόσα προϊόντα είναι out of stock".

2. Κεντρική Τάση & Διασπορά

Εδώ καταλαβαίνουμε πού "βρίσκονται" τα δεδομένα μας και πόσο "σκορπισμένα" είναι.

- **mean() vs median()**:
    - Αν τα δεδομένα σου έχουν ακραίες τιμές (outliers), ο μέσος όρος (mean) μπορεί να σε παραπλανήσει.
    - Η διάμεσος (median) είναι πιο ανθεκτική. Για παράδειγμα, αν 10 άτομα βγάζουν 1.000€ και ένας βγάζει 1.000.000€, ο μέσος όρος θα είναι πολύ υψηλός, αλλά η διάμεσος θα παραμείνει στα 1.000€.
- **std() & var()**: Η τυπική απόκλιση σου λέει αν οι τιμές σου είναι όλες κοντά στον μέσο όρο ή αν υπάρχει τεράστια αστάθεια.

3. Συσχέτιση: Πώς επηρεάζει το ένα το άλλο;

- **corr()**: Παράγει έναν πίνακα (correlation matrix) που δείχνει πώς συνδέονται οι στήλες.
    - Τιμή **1.0**: Τέλεια θετική συσχέτιση.
    - Τιμή **-1.0**: Τέλεια αρνητική συσχέτιση.
    - Τιμή **0**: Καμία γραμμική σχέση.

4. Στατιστικά Χρονοσειρών (Time-Series Insights)

Αν τα δεδομένα σου είναι ταξινομημένα χρονικά (π.χ. τιμές μετοχών), αυτές οι μέθοδοι είναι απαραίτητες:

- **cumsum()**: Ιδανικό για να βλέπεις τις συνολικές πωλήσεις έτους μέρα με τη μέρα.
- **pct_change()**: Η "βασίλισσα" των χρηματοοικονομικών. Σου δείχνει την απόδοση (π.χ. "σήμερα η μετοχή ανέβηκε 2%").
- **diff()**: Χρήσιμο για να δεις την απόλυτη διαφορά (π.χ. "πόσες παραπάνω πωλήσεις κάναμε σήμερα σε σχέση με χθες;").

5. Προχωρημένες Κατανομές

- **skew() (Λοξότητα)**: Σου λέει αν η κατανομή σου γέρνει προς τα αριστερά ή δεξιά.

- **kurtosis() (Κύρτωση)**: Σου δείχνει αν οι ακραίες τιμές (outliers) είναι πολύ συχνές ("βαριές ουρές").

```
import pandas as pd
import numpy as np

# ----------------------------------------------------------------------
# Δημιουργία δείγματος DataFrame με ποικιλία δεδομένων
# ----------------------------------------------------------------------
print("ΔΗΜΙΟΥΡΓΙΑ ΔΕΙΓΜΑΤΟΣ DataFrame")
print("-" * 70)

# Για αναπαραγωγιμότητα
np.random.seed(42)

# Δημιουργία ημερομηνιών
date_rng = pd.date_range(start='2023-01-01', end='2023-12-31', freq='M')

# Δεδομένα
df = pd.DataFrame({
    'A': np.random.normal(50, 10, 12),         # κανονική κατανομή
    'B': np.random.uniform(20, 80, 12),        # ομοιόμορφη κατανομή
    'C': np.random.randint(1, 100, 12),         # ακέραιοι
    'D': np.random.choice(['X', 'Y', 'Z'], 12), # κατηγορική
    'E': np.random.permutation([np.nan, 10, 20, 30, 40, 50, 60, 70, 80, 90,
100, np.nan]), # με NaN
    'Date': date_rng
})

# Προσθήκη μερικών ακόμα στηλών για πλουσιότερα παραδείγματα
df['F'] = df['A'] * 2 + np.random.normal(0, 5, 12)  # συσχετισμένη με A
df['G'] = np.random.choice(['Cat', 'Dog', 'Bird'], 12)

print("DataFrame (πρώτες 5 γραμμές):")
print(df.head())
print("\nΠληροφορίες τύπων:")
print(df.dtypes)
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 1. ΠΕΡΙΓΡΑΦΙΚΗ ΣΤΑΤΙΣΤΙΚΗ
# ----------------------------------------------------------------------
print("1. ΠΕΡΙΓΡΑΦΙΚΗ ΣΤΑΤΙΣΤΙΚΗ")
print("-" * 50)

# 1.1 describe() - Συνοπτική στατιστική για αριθμητικές στήλες
print("1.1 describe() - προεπιλογή (μόνο αριθμητικές):")
print(df.describe())
print()

# describe με προσαρμοσμένα ποσοστιαία σημεία
print("describe(percentiles=[.1, .5, .9]) - με επιπλέον ποσοστιαία:")
print(df[['A', 'B', 'C']].describe(percentiles=[.1, .5, .9]))
print()

# include='all' για να συμπεριλάβει και κατηγορικές (όχι όλα τα στατιστικά)
print("describe(include='all') - περιλαμβάνει κατηγορικές:")
print(df.describe(include='all'))
print()

# 1.2 info() - Συνοπτική πληροφόρηση DataFrame
```

```
print("1.2 info():")
df.info()
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 2. ΜΕΤΡΑ ΚΕΝΤΡΙΚΗΣ ΤΑΣΗΣ
# ----------------------------------------------------------------------
print("2. ΜΕΤΡΑ ΚΕΝΤΡΙΚΗΣ ΤΑΣΗΣ")
print("-" * 50)

# 2.1 mean() - Μέσος όρος
print("2.1 mean() - Μέσος όρος κάθε αριθμητικής στήλης:")
print(df.mean(numeric_only=True))
print()

# Μέσος όρος συγκεκριμένης στήλης
print("Μέσος όρος στήλης A:", df['A'].mean())
print()

# 2.2 median() - Διάμεσος
print("2.2 median() - Διάμεσος:")
print(df.median(numeric_only=True))
print()

# 2.3 mode() - Επικρατούσα τιμή (μπορεί να έχει πολλές)
print("2.3 mode() - Επικρατούσες τιμές:")
print(df.mode(numeric_only=True).iloc[0])  # πρώτη επικρατούσα (αν υπάρχουν
πολλές)
# Για κατηγορικές
print("Επικρατούσα στην D:", df['D'].mode().values)
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 3. ΜΕΤΡΑ ΔΙΑΣΠΟΡΑΣ
# ----------------------------------------------------------------------
print("3. ΜΕΤΡΑ ΔΙΑΣΠΟΡΑΣ")
print("-" * 50)

# 3.1 std() - Τυπική απόκλιση
print("3.1 std() - Τυπική απόκλιση:")
print(df.std(numeric_only=True))
print()

# 3.2 var() - Διακύμανση
print("3.2 var() - Διακύμανση:")
print(df.var(numeric_only=True))
print()

# 3.3 min() - Ελάχιστη τιμή
print("3.3 min() - Ελάχιστη:")
print(df.min(numeric_only=True))
print()

# 3.4 max() - Μέγιστη τιμή
print("3.4 max() - Μέγιστη:")
print(df.max(numeric_only=True))
print()

# 3.5 quantile() - Ποσοστημόριο
print("3.5 quantile(0.25) - 1ο τεταρτημόριο:")
print(df.quantile(0.25, numeric_only=True))
```

```python
print("quantile([0.25, 0.5, 0.75]) - πολλά ποσοστιαία:")
print(df.quantile([0.25, 0.5, 0.75], numeric_only=True))
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 4. ΣΥΣΣΩΡΕΥΤΙΚΑ ΜΕΤΡΑ (CUMULATIVE)
# ----------------------------------------------------------------------
print("4. ΣΥΣΣΩΡΕΥΤΙΚΑ ΜΕΤΡΑ")
print("-" * 50)

# Δημιουργούμε ένα μικρότερο DataFrame για πιο ευανάγνωστα αποτελέσματα
df_cum = df[['A', 'B', 'C']].head(6).copy()
print("Υποσύνολο για συσσωρευτικά (πρώτες 6 γραμμές):")
print(df_cum)
print()

# 4.1 cumsum() - Αθροιστικό άθροισμα
print("4.1 cumsum():")
print(df_cum.cumsum())
print()

# 4.2 cumprod() - Αθροιστικό γινόμενο
print("4.2 cumprod():")
print(df_cum.cumprod())
print()

# 4.3 cummax() - Αθροιστικό μέγιστο
print("4.3 cummax():")
print(df_cum.cummax())
print()

# 4.4 cummin() - Αθροιστικό ελάχιστο
print("4.4 cummin():")
print(df_cum.cummin())
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 5. ΣΥΣΧΕΤΙΣΗ & ΣΥΝΔΙΑΚΥΜΑΝΣΗ
# ----------------------------------------------------------------------
print("5. ΣΥΣΧΕΤΙΣΗ & ΣΥΝΔΙΑΚΥΜΑΝΣΗ")
print("-" * 50)

# 5.1 corr() - Συσχέτιση Pearson
print("5.1 corr() - Συσχέτιση:")
print(df[['A', 'B', 'C', 'F']].corr())
print()

# 5.2 cov() - Συνδιακύμανση
print("5.2 cov() - Συνδιακύμανση:")
print(df[['A', 'B', 'C', 'F']].cov())
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 6. ΜΟΝΑΔΙΚΕΣ & ΑΠΑΡΙΘΜΗΣΗ ΤΙΜΩΝ
# ----------------------------------------------------------------------
print("6. ΜΟΝΑΔΙΚΕΣ & ΑΠΑΡΙΘΜΗΣΗ ΤΙΜΩΝ")
print("-" * 50)

# 6.1 count() - Πλήθος μη-κενών τιμών
print("6.1 count() - Μη-κενά ανά στήλη:")
print(df.count())
```

```
print()

# 6.2 nunique() - Πλήθος μοναδικών τιμών
print("6.2 nunique() - Μοναδικές τιμές ανά στήλη:")
print(df.nunique())
print()

# 6.3 value_counts() - Συχνότητες μοναδικών τιμών (σε στήλη)
print("6.3 value_counts() - Συχνότητες για στήλη D:")
print(df['D'].value_counts())
print("\nΓια στήλη G (με κατηγορικές):")
print(df['G'].value_counts())
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 7. ΕΙΔΙΚΕΣ ΣΤΑΤΙΣΤΙΚΕΣ
# ----------------------------------------------------------------------
print("7. ΕΙΔΙΚΕΣ ΣΤΑΤΙΣΤΙΚΕΣ")
print("-" * 50)

# 7.1 abs() - Απόλυτες τιμές
df_abs = df.copy()
df_abs['Neg'] = np.random.normal(0, 5, 12)  # προσθήκη αρνητικών τιμών
print("7.1 abs() - Απόλυτες τιμές (στήλη Neg):")
print(df_abs.assign(Neg_abs=df_abs['Neg'].abs())[['Neg',
'Neg_abs']].head())
print()

# 7.2 skew() - Λοξότητα
print("7.2 skew() - Λοξότητα:")
print(df.skew(numeric_only=True))
print()

# 7.3 kurtosis() - Κύρτωση
print("7.3 kurtosis() - Κύρτωση:")
print(df.kurtosis(numeric_only=True))
print()

# 7.4 diff() - Διαφορές (προεπιλογή: periods=1)
print("7.4 diff() - Διαφορές μεταξύ διαδοχικών γραμμών (στήλη A):")
print(df['A'].diff().head(6))
print()

# 7.5 pct_change() - Ποσοστιαία μεταβολή
print("7.5 pct_change() - Ποσοστιαία μεταβολή (στήλη A):")
print(df['A'].pct_change().head(6))
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 8. ΣΥΓΚΕΝΤΡΩΤΙΚΗ ΕΠΙΣΚΟΠΗΣΗ (προαιρετική)
# ----------------------------------------------------------------------
print("8. ΣΥΓΚΕΝΤΡΩΤΙΚΗ ΕΠΙΣΚΟΠΗΣΗ ΣΤΟΙΧΕΙΩΝ")
print("-" * 50)
print("Ολοκληρώθηκε η επίδειξη όλων των στατιστικών μεθόδων.")
print("Τέλος προγράμματος.")
```

Κ. 3.4.9: Παράδειγμα για τις στατιστικές μεθόδους των Πλαισίων Δεδομένων

3.4.9. Σωρευτικοί Υπολογισμοί και Μεταβολές

Μέθοδος	Περιγραφή	Επιστροφή
Σωρευτικοί Υπολογισμοί (Cumulative Operations)		
cumsum()	Υπολογίζει το αθροιστικό άθροισμα (cumulative sum) των τιμών κατά μήκος ενός άξονα (συνήθως ανά στήλη ή ανά γραμμή). Επιστρέφει ένα νέο DataFrame με το ίδιο μέγεθος.	df.cumsum() df.cumsum(axis=1) (αθροιστικό άθροισμα ανά γραμμή)
cumprod()	Υπολογίζει το αθροιστικό γινόμενο (cumulative product) των τιμών.	df.cumprod()
cummax()	Επιστρέφει το αθροιστικό μέγιστο (cumulative maximum) των τιμών μέχρι το τρέχον στοιχείο.	df.cummax()
cummin()	Επιστρέφει το αθροιστικό ελάχιστο (cumulative minimum) των τιμών.	df.cummin()
expanding().sum()	Παρέχει λειτουργικότητα για "διογκούμενα" παράθυρα (expanding windows). Με το .sum() υπολογίζει το άθροισμα όλων των προηγούμενων τιμών (παρόμοιο με το cumsum αλλά πιο ευέλικτο).	df.expanding().sum()
expanding().mean()	Υπολογίζει τον αθροιστικό μέσο όρο (cumulative mean).	df.expanding().mean()
Υπολογισμοί Μεταβολών (Differences & Shifts)		
diff()	Υπολογίζει την πρώτη διακριτή διαφορά (discrete difference) μεταξύ στοιχείων. Προαιρετικά, μπορεί να οριστεί ο αριθμός των περιόδων (periods) για τη μετατόπιση.	df.diff() (διαφορά από την προηγούμενη γραμμή) df.diff(periods=2) (διαφορά από δύο γραμμές πριν) df.diff(axis=1) (διαφορά από την προηγούμενη στήλη)
pct_change()	Υπολογίζει την ποσοστιαία μεταβολή (percentage change) μεταξύ του τρέχοντος και ενός προηγούμενου στοιχείου.	df.pct_change() df.pct_change(periods=2)
shift()	Μετατοπίζει (shifts) τα δεδομένα κατά έναν επιθυμητό αριθμό περιόδων, δημιουργώντας μια "λίστα" με προηγούμενες ή επόμενες τιμές. Δεν υπολογίζει κάποια πράξη, αλλά είναι χρήσιμη για συγκρίσεις.	df.shift() (μετατοπίζει όλες τις τιμές κατά μία γραμμή προς τα κάτω) df.shift(periods=-1) (μετατοπίζει προς τα πάνω)

Πίνακας 3.4.10: Μέθοδοι Σωρευτικών Υπολογισμών και Μεταβολών για Πλαίσια Δεδομένων

Αυτές οι μέθοδοι αποτελούν τον πυρήνα της **Ανάλυσης Χρονοσειρών (Time-Series Analysis)** και της χρηματοοικονομικής μοντελοποίησης στην Pandas. Ενώ οι απλές στατιστικές (όπως ο μέσος όρος) μας δίνουν μια στατική εικόνα, οι σωρευτικοί υπολογισμοί και οι μετατοπίσεις μας επιτρέπουν να δούμε την **κίνηση** και την **εξέλιξη** των δεδομένων στον χρόνο.

1. Σωρευτικοί Υπολογισμοί (Cumulative Operations)

Οι μέθοδοι αυτές "κουβαλάνε" την πληροφορία από τις προηγούμενες γραμμές στις επόμενες.

- **cumsum()**: Το πιο κλασικό παράδειγμα είναι η παρακολούθηση των εσόδων (Revenue) ενός καταστήματος. Κάθε μέρα προστίθενται οι πωλήσεις στις ήδη υπάρχουσες για να δούμε το σύνολο "μέχρι σήμερα".
- **cummax() / cummin()**: Χρησιμοποιούνται συχνά για να εντοπίσουμε ιστορικά υψηλά ή χαμηλά. Για παράδειγμα, σε μια μετοχή, το cummax() μας δείχνει ποια ήταν η υψηλότερη τιμή που έχει καταγραφεί από την αρχή του dataset μέχρι εκείνη τη στιγμή.
- **expanding()**: Αυτή είναι μια πιο εξελιγμένη μορφή των cumulative operations. Ενώ η cumsum() κάνει μόνο άθροισμα, η expanding() σου επιτρέπει να εφαρμόσεις **οποιαδήποτε** πράξη (π.χ. mean(), std()) σε ένα παράθυρο που μεγαλώνει συνεχώς.

Tip: Η expanding().mean() είναι εξαιρετική για να βλέπεις πώς σταθεροποιείται ο μέσος όρος ενός πειράματος καθώς προστίθενται νέα δεδομένα.

2. Υπολογισμοί Μεταβολών (Differences & Shifts)

Αυτές οι μέθοδοι μας βοηθούν να συγκρίνουμε το "τώρα" με το "χθες".

- **diff()**: Υπολογίζει την απόλυτη μεταβολή.
    - *Παράδειγμα:* Αν χθες είχαμε 100 πωλήσεις και σήμερα 120, το diff() θα επιστρέψει 20.
- **pct_change()**: Είναι η πιο σημαντική μέθοδος για οικονομική ανάλυση. Μετατρέπει τις απόλυτες τιμές σε ποσοστά, κάνοντας τα δεδομένα συγκρίσιμα. Μια άνοδος 10€ έχει άλλη βαρύτητα αν το προϊόν κάνει 100€ και άλλη αν κάνει 1.000€.
- **shift()**: Ίσως η πιο "παρεξηγημένη" αλλά χρήσιμη μέθοδος. Δεν αλλάζει τις τιμές, αλλά τις **μετακινεί**.
    - **Γιατί είναι χρήσιμη;** Αν θέλεις να υπολογίσεις κάτι χειροκίνητα συγκρίνοντας τη σημερινή τιμή με την προηγούμενη στην ίδια γραμμή, χρησιμοποιείς το shift() για να φέρεις την τιμή του "χθες" δίπλα στην τιμή του "σήμερα".

```
import pandas as pd
import numpy as np

# ---------------------------------------------------------------------
# Δημιουργία δείγματος DataFrame με αριθμητικά δεδομένα
# ---------------------------------------------------------------------
print("ΔΗΜΙΟΥΡΓΙΑ ΔΕΙΓΜΑΤΟΣ DataFrame")
print("-" * 60)

# Δημιουργούμε ένα μικρό DataFrame με τρεις στήλες και 5 γραμμές
data = {
    'A': [10, 20, 15, 30, 25],
    'B': [5, 8, 12, 10, 15],
```

```
    'C': [100, 150, 130, 200, 180]
}
df = pd.DataFrame(data, index=['Row1', 'Row2', 'Row3', 'Row4', 'Row5'])
print("Αρχικό DataFrame:")
print(df)
print("\n" + "="*90 + "\n")

# ----------------------------------------------------------------------
# 1. ΣΩΡΕΥΤΙΚΟΙ ΥΠΟΛΟΓΙΣΜΟΙ (CUMULATIVE OPERATIONS)
# ----------------------------------------------------------------------
print("1. ΣΩΡΕΥΤΙΚΟΙ ΥΠΟΛΟΓΙΣΜΟΙ")
print("-" * 40)

# 1.1 cumsum() - Αθροιστικό άθροισμα ανά στήλη (axis=0, προεπιλογή)
print("1.1 cumsum() - Αθροιστικό άθροισμα (ανά στήλη):")
print(df.cumsum())
print()

# cumsum() με axis=1 (αθροιστικό άθροισμα ανά γραμμή)
print("cumsum(axis=1) - Αθροιστικό άθροισμα ανά γραμμή:")
print(df.cumsum(axis=1))
print()

# 1.2 cumprod() - Αθροιστικό γινόμενο
print("1.2 cumprod() - Αθροιστικό γινόμενο (ανά στήλη):")
print(df.cumprod())
print()

# 1.3 cummax() - Αθροιστικό μέγιστο
print("1.3 cummax() - Αθροιστικό μέγιστο (ανά στήλη):")
print(df.cummax())
print()

# 1.4 cummin() - Αθροιστικό ελάχιστο
print("1.4 cummin() - Αθροιστικό ελάχιστο (ανά στήλη):")
print(df.cummin())
print()

# 1.5 expanding().sum() - Αθροιστικό σύνολο με expanding window (ίδιο με
cumsum)
print("1.5 expanding().sum() - Αθροιστικό σύνολο (ίδιο με cumsum):")
print(df.expanding().sum())
print()

# 1.6 expanding().mean() - Αθροιστικός μέσος όρος
print("1.6 expanding().mean() - Αθροιστικός μέσος όρος:")
print(df.expanding().mean())
print()

print("\n" + "="*90 + "\n")

# ----------------------------------------------------------------------
# 2. ΥΠΟΛΟΓΙΣΜΟΙ ΜΕΤΑΒΟΛΩΝ & ΜΕΤΑΤΟΠΙΣΕΩΝ
# ----------------------------------------------------------------------
print("2. ΥΠΟΛΟΓΙΣΜΟΙ ΜΕΤΑΒΟΛΩΝ & ΜΕΤΑΤΟΠΙΣΕΩΝ")
print("-" * 50)

# 2.1 diff() - Διαφορές μεταξύ διαδοχικών στοιχείων
print("2.1 diff() - Διαφορά από προηγούμενη γραμμή (periods=1):")
print(df.diff())
print()
```

```python
print("diff(periods=2) - Διαφορά από δύο γραμμές πριν:")
print(df.diff(periods=2))
print()

print("diff(axis=1) - Διαφορά από προηγούμενη στήλη:")
print(df.diff(axis=1))
print()

# 2.2 pct_change() - Ποσοστιαία μεταβολή
print("2.2 pct_change() - Ποσοστιαία μεταβολή από προηγούμενη γραμμή:")
print(df.pct_change())
print()

print("pct_change(periods=2) - Ποσοστιαία μεταβολή από δύο γραμμές πριν:")
print(df.pct_change(periods=2))
print()

# 2.3 shift() - Μετατόπιση δεδομένων
print("2.3 shift() - Μετατόπιση προς τα κάτω (periods=1, προεπιλογή):")
print(df.shift())
print()

print("shift(periods=-1) - Μετατόπιση προς τα πάνω:")
print(df.shift(periods=-1))
print()

print("shift(periods=2, axis=1) - Μετατόπιση στηλών προς τα δεξιά:")
print(df.shift(periods=2, axis=1))
print()

print("\n" + "="*90 + "\n")

# ----------------------------------------------------------------------
# 3. ΠΑΡΑΔΕΙΓΜΑ ΣΥΝΔΥΑΣΜΟΥ ΜΕΘΟΔΩΝ
# ----------------------------------------------------------------------
print("3. ΣΥΝΔΥΑΣΜΟΣ ΜΕΘΟΔΩΝ (π.χ. diff και pct_change)")
print("-" * 50)

# Δημιουργούμε μια χρονοσειρά για πιο ρεαλιστικό παράδειγμα
dates = pd.date_range('2024-01-01', periods=5, freq='D')
ts_df = pd.DataFrame({
    'Sales': [200, 220, 210, 240, 250]
}, index=dates)
print("Χρονοσειρά Πωλήσεων:")
print(ts_df)
print()

# Ημερήσια μεταβολή σε απόλυτους αριθμούς
print("Ημερήσια διαφορά (diff()):")
print(ts_df.diff())
print()

# Ημερήσια ποσοστιαία μεταβολή
print("Ημερήσια % μεταβολή (pct_change()):")
print(ts_df.pct_change())
print()

# Δημιουργία μετατοπισμένης στήλης για σύγκριση
ts_df['Sales_prev'] = ts_df['Sales'].shift(1)
print("Προσθήκη στήλης με προηγούμενες πωλήσεις (shift):")
```

```
print(ts_df)
print()

# Αθροιστικό άθροισμα πωλήσεων
print("Αθροιστικές πωλήσεις (cumsum()):")
print(ts_df['Sales'].cumsum())
print()

print("\n" + "="*90 + "\n")
print("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
```

Κ. 3.4.10: Παράδειγμα Μεθόδων Σωρευτικών Υπολογισμών και Μεταβολών των Πλαισίων Δεδομένων

3.4.10. Μαθηματικές Πράξεις – Αριθμητικές Μέθοδοι

Μέθοδος	Περιγραφή	Παράδειγμα
add()	Πρόσθεση στοιχείο-προς-στοιχείο. Ισοδύναμο με τον τελεστή **+** .	df.add(1) ή df + 1 (προσθέτει 1 σε κάθε στοιχείο)
sub()	Αφαίρεση στοιχείο-προς-στοιχείο. Ισοδύναμο με τον τελεστή **-** .	df.sub([1, 2], axis='columns') (αφαιρεί 1 από την 1η στήλη και 2 από τη 2η)
mul()	Πολλαπλασιασμός στοιχείο-προς-στοιχείο. Ισοδύναμο με τον τελεστή ***** .	df.mul(10) (πολλαπλασιάζει κάθε στοιχείο με το 10)
div()	Διαίρεση στοιχείο-προς-στοιχείο (κινητής υποδιαστολής). Ισοδύναμο με τον τελεστή **/** .	df.div(10) (διαιρεί κάθε στοιχείο με το 10)
truediv()	Ακριβώς ίδια με την div() .	df.truediv(10)
floordiv()	Ακέραια διαίρεση στοιχείο-προς-στοιχείο. Ισοδύναμο με τον τελεστή **//** .	df.floordiv(3) (διαιρεί και κρατά το ακέραιο μέρος)
pow()	Ύψωση σε δύναμη στοιχείο-προς-στοιχείο. Ισοδύναμο με τον τελεστή ****** .	df.pow(2) (υψώνει κάθε στοιχείο στο τετράγωνο)
mod()	Υπόλοιπο διαίρεσης (modulo) στοιχείο-προς-στοιχείο. Ισοδύναμο με τον τελεστή **%** .	df.mod(2) (βρίσκει αν κάθε στοιχείο είναι άρτιο ή περιττό)
abs()	Υπολογίζει την απόλυτη τιμή κάθε στοιχείου .	df.abs() (μετατρέπει όλες τις αρνητικές τιμές σε θετικές)
round()	Στρογγυλοποιεί κάθε στοιχείο σε καθορισμένο αριθμό δεκαδικών ψηφίων .	df.round(2) (στρογγυλοποιεί όλες τις τιμές σε 2 δεκαδικά)

Πίνακας 3.4.11: Μέθοδοι Αριθμητικών Πράξεων των Πλαισίων Δεδομένων

Παρόλο που η Python επιτρέπει τη χρήση συμβόλων (όπως +, -, *), οι ενσωματωμένες μέθοδοι της Pandas (add, sub, κ.λπ.) προσφέρουν ένα κρίσιμο πλεονέκτημα: την παράμετρο **fill_value** και τον έλεγχο του άξονα (**axis**).

1. Γιατί να χρησιμοποιείς τις μεθόδους αντί για τα σύμβολα;

Αν προσπαθήσεις να προσθέσεις δύο DataFrames που έχουν κενές τιμές (NaN) χρησιμοποιώντας το σύμβολο +, το αποτέλεσμα θα είναι NaN. Αν όμως χρησιμοποιήσεις την df1.add(df2, fill_value=0), η Pandas θα αντικαταστήσει τα κενά με 0 **πριν** την πράξη, σώζοντας τα δεδομένα σου.

2. Ανάλυση των Μεθόδων

- **Βασικές Πράξεις**

- **add() & sub()**: Εκτός από απλούς αριθμούς, μπορείς να αφαιρέσεις μια ολόκληρη Series από ένα DataFrame. Για παράδειγμα, αν θες να αφαιρέσεις τον μέσο όρο κάθε στήλης από τις τιμές της, χρησιμοποιείς το df.sub(df.mean(), axis='columns').
- **mul() & div()**: Πολύ χρήσιμες για μετατροπές μονάδων (π.χ. μετατροπή τιμών από δολάρια σε ευρώ).
- **Εξειδικευμένες Διαιρέσεις**
- **floordiv() (//)**: Χρήσιμο όταν θες να ομαδοποιήσεις δεδομένα σε "κουτιά" (bins). Π.χ. df['Age'].floordiv(10) θα σου δώσει τη δεκαετία ηλικίας (2 για τα 20-29, 3 για τα 30-39).
- **mod() (%)**: Ιδανικό για να βρεις μοτίβα. Για παράδειγμα, σε χρονοσειρές, το mod(7) μπορεί να βοηθήσει στον εντοπισμό εβδομαδιαίων κύκλων.
- **Μορφοποίηση και Διόρθωση**
- **abs()**: Απαραίτητο στη στατιστική, ειδικά όταν υπολογίζεις σφάλματα (residuals), όπου μας ενδιαφέρει η απόσταση από την πραγματική τιμή και όχι το πρόσημο.
- **round()**: Η τελευταία πινελιά πριν την παρουσίαση.

Tip: Μπορείς να δώσεις λεξικό στην round() για διαφορετικά δεκαδικά ανά στήλη: df.round({'Price': 2, 'Quantity': 0}).

3. Broadcasting: Πώς ευθυγραμμίζονται τα δεδομένα

Όταν κάνεις πράξεις στην Pandas, συμβαίνει το λεγόμενο **Broadcasting**. Η βιβλιοθήκη προσπαθεί να ταιριάξει τα labels των γραμμών και των στηλών. Αν ένα label υπάρχει στο ένα DataFrame αλλά όχι στο άλλο, η Pandas θα δημιουργήσει μια νέα στήλη με NaN (εκτός αν χρησιμοποιήσεις το fill_value που αναφέραμε).

```
import pandas as pd
import numpy as np

# ------------------------------------------------------------------
# Δημιουργία δείγματος DataFrame με αριθμητικά δεδομένα
# ------------------------------------------------------------------
print("ΔΗΜΙΟΥΡΓΙΑ ΔΕΙΓΜΑΤΟΣ DataFrame")
print("-" * 60)

data = {
    'A': [10, -5, 7, -3, 12],
    'B': [3, 8, -2, 5, 0],
    'C': [2.5, 4.1, 6.3, 1.2, 9.8]
}
df = pd.DataFrame(data, index=['R1', 'R2', 'R3', 'R4', 'R5'])
print("Αρχικό DataFrame:")
print(df)
print("\n" + "="*100 + "\n")

# ------------------------------------------------------------------
# 1. ΠΡΟΣΘΕΣΗ - add() / +
# ------------------------------------------------------------------
print("1. ΠΡΟΣΘΕΣΗ - add()")
print("-" * 40)

# Προσθήκη σταθεράς σε όλα τα στοιχεία
```

```
print("add(1) - προσθέτει 1 σε κάθε στοιχείο:")
print(df.add(1))
print("\nΙσοδύναμο με τελεστή + : df + 1")
print(df + 1)
print()

# Προσθήκη λίστας τιμών ανά στήλη (χρειάζεται axis='columns' ή axis=1)
print("add([1, 2, 3], axis='columns') - προσθέτει 1 στη στήλη A, 2 στη B, 3
στη C:")
print(df.add([1, 2, 3], axis='columns'))
print()

# Προσθήκη λίστας ανά γραμμή (axis='index' ή axis=0)
print("add([10, 20, 30, 40, 50], axis='index') - προσθέτει ανά γραμμή:")
print(df.add([10, 20, 30, 40, 50], axis='index'))
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 2. ΑΦΑΙΡΕΣΗ - sub() / -
# ----------------------------------------------------------------------
print("2. ΑΦΑΙΡΕΣΗ - sub()")
print("-" * 40)

# Αφαίρεση σταθεράς
print("sub(5) - αφαιρεί 5 από κάθε στοιχείο:")
print(df.sub(5))
print("\nΙσοδύναμο με τελεστή - : df - 5")
print(df - 5)
print()

# Αφαίρεση λίστας ανά στήλη
print("sub([1, 2, 0.5], axis='columns') - αφαιρεί διαφορετικές τιμές ανά
στήλη:")
print(df.sub([1, 2, 0.5], axis='columns'))
print()

# Αφαίρεση ανά γραμμή
print("sub([100, 200, 300, 400, 500], axis='index') - αφαιρεί ανά γραμμή:")
print(df.sub([100, 200, 300, 400, 500], axis='index'))
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 3. ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΣ - mul() / *
# ----------------------------------------------------------------------
print("3. ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΣ - mul()")
print("-" * 40)

# Πολλαπλασιασμός με σταθερά
print("mul(3) - πολλαπλασιάζει κάθε στοιχείο με 3:")
print(df.mul(3))
print("\nΙσοδύναμο με τελεστή * : df * 3")
print(df * 3)
print()

# Πολλαπλασιασμός με λίστα ανά στήλη
print("mul([2, 5, 10], axis='columns') - διαφορετικοί συντελεστές ανά
στήλη:")
print(df.mul([2, 5, 10], axis='columns'))
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
```

```python
# 4. ΔΙΑΙΡΕΣΗ - div() / truediv() / /
# ------------------------------------------------------------------
print("4. ΔΙΑΙΡΕΣΗ - div() / truediv()")
print("-" * 40)

# Διαίρεση με σταθερά
print("div(2) - διαιρεί κάθε στοιχείο με 2:")
print(df.div(2))
print("\nΙσοδύναμο με τελεστή / : df / 2")
print(df / 2)
print()

# truediv() είναι συνώνυμο της div()
print("truediv(2) - ίδιο αποτέλεσμα:")
print(df.truediv(2))
print()

# Διαίρεση με λίστα ανά γραμμή
print("div([1, 2, 1, 2, 1], axis='index') - διαίρεση ανά γραμμή:")
print(df.div([1, 2, 1, 2, 1], axis='index'))
print("\n" + "="*100 + "\n")

# ------------------------------------------------------------------
# 5. ΑΚΕΡΑΙΑ ΔΙΑΙΡΕΣΗ - floordiv() / //
# ------------------------------------------------------------------
print("5. ΑΚΕΡΑΙΑ ΔΙΑΙΡΕΣΗ - floordiv()")
print("-" * 40)

# Ακέραια διαίρεση με σταθερά
print("floordiv(3) - ακέραιο πηλίκο διαίρεσης με 3:")
print(df.floordiv(3))
print("\nΙσοδύναμο με τελεστή // : df // 3")
print(df // 3)
print()

# Ακέραια διαίρεση με λίστα ανά στήλη
print("floordiv([2, 3, 4], axis='columns') - ανά στήλη:")
print(df.floordiv([2, 3, 4], axis='columns'))
print("\n" + "="*100 + "\n")

# ------------------------------------------------------------------
# 6. ΥΨΩΣΗ ΣΕ ΔΥΝΑΜΗ - pow() / **
# ------------------------------------------------------------------
print("6. ΥΨΩΣΗ ΣΕ ΔΥΝΑΜΗ - pow()")
print("-" * 40)

# Ύψωση σε δύναμη (σταθερά)
print("pow(2) - υψώνει κάθε στοιχείο στο τετράγωνο:")
print(df.pow(2))
print("\nΙσοδύναμο με τελεστή ** : df ** 2")
print(df ** 2)
print()

# Ύψωση σε δύναμη ανά στήλη
print("pow([2, 3, 1], axis='columns') - διαφορετικοί εκθέτες:")
print(df.pow([2, 3, 1], axis='columns'))
print()

# Προσοχή: αρνητικές βάσεις με δεκαδικό εκθέτη δίνουν μιγαδικούς, αλλά
αποφεύγουμε
print("\n" + "="*100 + "\n")
```

```python
# ------------------------------------------------------------------
# 7. ΥΠΟΛΟΙΠΟ ΔΙΑΙΡΕΣΗΣ - mod() / %
# ------------------------------------------------------------------
print("7. ΥΠΟΛΟΙΠΟ ΔΙΑΙΡΕΣΗΣ - mod()")
print("-" * 40)

# Υπόλοιπο διαίρεσης με σταθερά
print("mod(3) - υπόλοιπο διαίρεσης με 3:")
print(df.mod(3))
print("\nΙσοδύναμο με τελεστή % : df % 3")
print(df % 3)
print()

# Υπόλοιπο με λίστα ανά στήλη
print("mod([2, 5, 3], axis='columns') - ανά στήλη:")
print(df.mod([2, 5, 3], axis='columns'))
print("\n" + "="*100 + "\n")

# ------------------------------------------------------------------
# 8. ΑΠΟΛΥΤΗ ΤΙΜΗ - abs()
# ------------------------------------------------------------------
print("8. ΑΠΟΛΥΤΗ ΤΙΜΗ - abs()")
print("-" * 40)

print("abs() - απόλυτη τιμή κάθε στοιχείου:")
print(df.abs())
print()

# Σημείωση: η abs() δεν έχει τελεστή, είναι μόνο μέθοδος.
print("\n" + "="*100 + "\n")

# ------------------------------------------------------------------
# 9. ΣΤΡΟΓΓΥΛΟΠΟΙΗΣΗ - round()
# ------------------------------------------------------------------
print("9. ΣΤΡΟΓΓΥΛΟΠΟΙΗΣΗ - round()")
print("-" * 40)

print("round(1) - στρογγυλοποίηση σε 1 δεκαδικό ψηφίο:")
print(df.round(1))
print()

# Μπορούμε να ορίσουμε διαφορετικό αριθμό δεκαδικών ανά στήλη
print("round({'A':0, 'B':0, 'C':2}) - διαφορετικά δεκαδικά ανά στήλη:")
print(df.round({'A':0, 'B':0, 'C':2}))
print()

# Χωρίς όρισμα, στρογγυλοποιεί σε ακέραιο (0 δεκαδικά)
print("round() (default) - ακέραιες τιμές:")
print(df.round())
print("\n" + "="*100 + "\n")

# ------------------------------------------------------------------
# 10. ΣΥΝΔΥΑΣΜΕΝΑ ΠΑΡΑΔΕΙΓΜΑΤΑ
# ------------------------------------------------------------------
print("10. ΣΥΝΔΥΑΣΜΟΣ ΠΡΑΞΕΩΝ")
print("-" * 40)

# Δημιουργία αντιγράφου για πράξεις
df2 = df.copy()
```

```
# Προσθήκη νέας στήλης με υπολογισμούς
df2['D'] = df2['A'].add(df2['B']).mul(2).mod(5)
print("Δημιουργία στήλης D = (A + B) * 2 % 5:")
print(df2[['A', 'B', 'D']])
print()

# Χρήση round και abs μαζί
df2['E'] = df2['C'].sub(5).abs().round(1)
print("Στήλη E = |C - 5| με στρογγυλοποίηση 1 δεκαδικό:")
print(df2[['C', 'E']])
print()

print("\n" + "="*100 + "\n")
print("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
```

Κ. 3.4.11: Παράδειγμα για τις Μεθόδους Αριθμητικών Πράξεων των Πλαισίων Δεδομένων

3.4.11. Σχέσεις Σύγκρισης (Boolean Output)

Μέθοδος	Περιγραφή	Παράδειγμα
Βασικοί Τελεστές Σύγκρισης (<, >, ==, !=, <=, >=)	Εφαρμόζουν μια στοιχειακή (element-wise) σύγκριση μεταξύ ενός DataFrame και μιας άλλης δομής (π.χ. scalar, Series, άλλο DataFrame). Το αποτέλεσμα είναι ένα boolean DataFrame όπου κάθε κελί δείχνει αν η συνθήκη ισχύει ή όχι .	df[df['Ηλικία'] > 30] Επιστρέφει μόνο τις γραμμές όπου η στήλη 'Ηλικία' έχει τιμή μεγαλύτερη του 30.
eq() (equal)	Στοιχειακός έλεγχος ισότητας. Ισοδυναμεί με τον τελεστή == .	df['Marks'].eq(62) Επιστρέφει μια boolean Series που είναι True για κάθε γραμμή όπου ο βαθμός είναι 62.
ne() (not equal)	Στοιχειακός έλεγχος ανισότητας. Ισοδυναμεί με τον τελεστή != .	df[df['Κατάσταση'].ne('Ενεργό')] Επιλέγει όλες τις γραμμές όπου η κατάσταση δεν είναι 'Ενεργό'.
gt(), ge(), lt(), le()	Στοιχειακοί έλεγχοι διάταξης: μεγαλύτερο από (gt, >), μεγαλύτερο ή ίσο (ge, >=), μικρότερο από (lt, <), μικρότερο ή ίσο (le, <=) .	df[df['Τιμή'].gt(100)] Επιλέγει γραμμές όπου η τιμή είναι μεγαλύτερη από 100.
isin()	Ελέγχει αν τα στοιχεία ενός DataFrame ή μιας Series περιέχονται σε μια συγκεκριμένη λίστα ή σύνολο τιμών .	df[df['Πόλη'].isin(['Αθήνα', 'Θεσσαλονίκη'])] Επιστρέφει όσες γραμμές έχουν πόλη είτε 'Αθήνα' είτε 'Θεσσαλονίκη'.
eq() (σύγκριση στηλών)	Μπορεί να χρησιμοποιηθεί για τη στοιχειακή σύγκριση δύο στηλών του ίδιου DataFrame .	df['Αποτέλεσμα'] = df['Στήλη1'].eq(df['Στήλη2']) Δημιουργεί μια νέα στήλη 'Αποτέλεσμα' που περιέχει True όπου οι τιμές των δύο στηλών είναι ίσες.

Μέθοδος	Περιγραφή	Παράδειγμα
equals()	Συγκρίνει δύο ολόκληρα DataFrames (ή Series) για να δει αν είναι **ακριβώς ίδια**, συμπεριλαμβανομένων των δεδομένων, των δεικτών (index) και των ονομάτων στηλών. Επιστρέφει ένα μόνο boolean (True ή False) .	if df1.equals(df2): print("Τα DataFrames είναι ίδια")
compare()	Συγκρίνει δύο DataFrames και επιστρέφει ένα νέο DataFrame που επισημαίνει τις διαφορές τους .	df1.compare(df2) Επιστρέφει ένα DataFrame όπου εμφανίζονται οι διαφορές μεταξύ των δύο αρχικών DataFrame.
any() **και** all()	Χρησιμοποιούνται σε boolean πίνακες (π.χ. αποτέλεσμα σύγκρισης) για να ελέγξουν αν υπάρχει τουλάχιστον μία any() ή αν όλες all() οι τιμές είναι True .	if (df['Ηλικία'] > 100).any(): print("Υπάρχει άτομο άνω των 100 ετών")
str.contains()	Μέθοδος για στήλες string. Ελέγχει αν η συμβολοσειρά σε κάθε κελί περιέχει ένα συγκεκριμένο υπόστρωμα (pattern) .	df[df['Ονομα'].str.contains('άννα', case=False)] Βρίσκει γραμμές όπου το όνομα περιέχει 'άννα' (π.χ. 'Ιωάννα', 'Άννα').
str.startswith() **/** str.endswith()	Ελέγχει αν οι συμβολοσειρές σε μια στήλη ξεκινούν ή τελειώνουν με ένα συγκεκριμένο υπόστρωμα .	df[df['Ταχυδρομικός_Κώδικας'].str.startswith('1')] Βρίσκει γραμμές με ΤΚ που ξεκινά από 1.
isna() **/** isnull()	Εντοπίζει τις ελλείπουσες τιμές (NaN, None). Επιστρέφει ένα boolean DataFrame/Series όπου True σημαίνει "λείπει" .	df[df['Διεύθυνση'].isna()] Επιστρέφει γραμμές όπου η διεύθυνση λείπει.
notna() **/** notnull()	Εντοπίζει τις μη-ελλείπουσες τιμές. Επιστρέφει ένα boolean DataFrame/Series όπου True σημαίνει "υπάρχει τιμή" .	df[df['email'].notna()] Επιστρέφει γραμμές όπου υπάρχει καταχωρημένο email.
query()	Παρέχει έναν πιο ευανάγνωστο τρόπο φιλτραρίσματος γραμμών με χρήση σύνταξης που μοιάζει με SQL. Η συνθήκη δίνεται ως string .	df.query('Ηλικία >= 18 and Φύλο == "Γυναίκα"') Επιλέγει τις ενήλικες γυναίκες.
isin() (για index)	Μπορεί να ελέγξει αν οι ετικέτες (labels) του ευρετηρίου (index) ανήκουν σε ένα σύνολο τιμών .	df.index.isin(['Γραμμή_1', 'Γραμμή_3']) Επιστρέφει ένα boolean array για το αν κάθε ετικέτα του δείκτη είναι 'Γραμμή_1' ή 'Γραμμή_3'.

Πίνακας 3.4.12: Μέθοδοι για τις Σχέσεις Σύγκρισης (Boolean Output) των Πλαισίων Δεδομένων

Τελεστές και Μέθοδοι Σύγκρισης: Αυτά τα εργαλεία είναι η βάση για τη λήψη αποφάσεων: σου επιτρέπουν να συγκρίνεις δεδομένα, να εντοπίζεις διαφορές μεταξύ πινάκων και να εκτελείς σύνθετους ελέγχους ποιότητας.

1. Στοιχειώδης Σύγκριση (Element-wise)

Όπως και στις αριθμητικές πράξεις, οι μέθοδοι eq(), ne(), gt(), κ.λπ. προσφέρουν μεγαλύτερη ευελιξία από τα σύμβολα (==, !=, >), ειδικά όταν θέλεις να ευθυγραμμίσεις διαφορετικά DataFrames.

- **eq() vs ==**: Η χρήση του df1.eq(df2) σου επιτρέπει να ορίσεις πώς θα χειριστείς τα κενά ή πώς θα γίνει η ευθυγράμμιση των αξόνων.
- **isin()**: Είναι ο πιο αποδοτικός τρόπος να κάνεις "φιλτράρισμα λίστας". Αντί για 10 διαφορετικά OR, χρησιμοποιείς μια λίστα τιμών.

2. Σύγκριση Ολόκληρων Πινάκων

Εδώ η Pandas μας δίνει εργαλεία για να δούμε τη "μεγάλη εικόνα".

- **equals()**: **Προσοχή!** Ο τελεστής == μεταξύ δύο DataFrames θα σου επιστρέψει έναν πίνακα από True/False για κάθε κελί. Αν θέλεις μια μοναδική απάντηση (Ναι ή Όχι) για το αν δύο πίνακες είναι ολόκληροι ίδιοι, η equals() είναι η μόνη λύση.
- **compare()**: Ένα πανίσχυρο εργαλείο για "Debugging" δεδομένων. Αν έχεις δύο εκδόσεις του ίδιου αρχείου και θέλεις να δεις ακριβώς τι άλλαξε, η compare() θα σου δείξει τις παλιές και τις νέες τιμές δίπλα-δίπλα.

3. Λογικοί Έλεγχοι: any() και all()

Αυτές οι μέθοδοι "συμπυκνώνουν" boolean αποτελέσματα.

- **any()**: Επιστρέφει True αν υπάρχει **τουλάχιστον ένα** True στον άξονα που εξετάζεις. Χρήσιμο για να βρεις αν υπάρχουν outliers ή λάθη.
- **all()**: Επιστρέφει True μόνο αν **όλα** τα στοιχεία είναι True. Χρήσιμο για να επιβεβαιώσεις ότι τα δεδομένα σου πληρούν μια προϋπόθεση (π.χ. "είναι όλες οι ημερομηνίες εντός του 2024;").

4. Εξειδικευμένοι Έλεγχοι (Strings & Nulls)

- **str.contains()**: Η δύναμη των Regular Expressions (Regex). Μπορείς να ψάξεις για σύνθετα μοτίβα (π.χ. emails ή τηλέφωνα) μέσα σε στήλες κειμένου.
- **isna() / notna()**: Η βάση του καθαρισμού δεδομένων. Χωρίς αυτά, δεν μπορείς να προχωρήσεις σε καμία στατιστική ανάλυση, καθώς οι κενές τιμές αλλοιώνουν τα αποτελέσματα.

5. Η μέθοδος query()

Η query() είναι η πιο "κομψή" μέθοδος της Pandas. Επιτρέπει τη συγγραφή συνθηκών ως κείμενο, κάτι που κάνει τον κώδικα να διαβάζεται σαν φυσική γλώσσα ή SQL. Είναι ιδιαίτερα αποδοτική σε πολύ μεγάλα DataFrames.

Σημαντικές σημειώσεις

- **Ασάφεια στη Σύγκριση:** Η Python και η Pandas δεν επιτρέπουν τη χρήση ενός ολόκληρου boolean DataFrame ή Series (π.χ. df > 5) απευθείας σε μια συνθήκη if, γιατί το αποτέλεσμα είναι πολλαπλών τιμών και άρα διφορούμενο. Για αυτό το λόγο χρησιμοποιούμε μεθόδους όπως η .any() ή η .all() για να το "συμπυκνώσουμε" σε μία μοναδική boolean τιμή .
- **Τελεστής** in**:** Σε ένα DataFrame, ο τελεστής in ελέγχει αν ένα όνομα υπάρχει στις **στήλες** και όχι στα δεδομένα . Για παράδειγμα, 'Ηλικία' in df ελέγχει αν υπάρχει στήλη με όνομα 'Ηλικία'.

- **Σύγκριση με** NaN: Να θυμάστε ότι η σύγκριση με τιμές NaN έχει ιδιαιτερότητες. Για παράδειγμα, NaN == NaN επιστρέφει False. Για τον εντοπισμό ελλειπουσών τιμών χρησιμοποιούμε πάντα τις isna() / isnull() .

```
import pandas as pd
import numpy as np

# ----------------------------------------------------------------
# Δημιουργία δείγματος DataFrame με ποικιλία δεδομένων
# ----------------------------------------------------------------
print("ΔΗΜΙΟΥΡΓΙΑ ΔΕΙΓΜΑΤΟΣ DataFrame")
print("-" * 70)

data = {
    'Name': ['Anna', 'George', 'Maria', 'Nikos', 'Eleni', 'Dimitris',
'Sofia', 'John', 'Alice', 'Bob'],
    'Age': [25, 30, 28, 35, 22, 27, 24, 40, 31, 29],
    'Gender': ['Female', 'Male', 'Female', 'Male', 'Female', 'Male',
'Female', 'Male', 'Female', 'Male'],
    'City': ['Athens', 'Thessaloniki', 'Patras', 'Heraklion', 'Volos',
'Chania', 'Larisa', 'London', 'Paris', 'Tokyo'],
    'Salary': [50000, 60000, 55000, 65000, 48000, 62000, 51000, 80000,
75000, 70000],
    'Department': ['Sales', 'IT', 'Sales', 'HR', 'IT', 'Sales', 'HR', 'IT',
'Sales', 'HR']
}

# Εισαγωγή μερικών ελλειπουσών τιμών για επίδειξη isna/notna
data['Salary'][5] = np.nan
data['City'][2] = np.nan

df = pd.DataFrame(data, index=[f'Row{i}' for i in range(1, 11)])
print("Αρχικό DataFrame:")
print(df)
print("\nΠληροφορίες τύπων:")
print(df.dtypes)
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------
# 1. ΒΑΣΙΚΟΙ ΤΕΛΕΣΤΕΣ ΣΥΓΚΡΙΣΗΣ (>, <, ==, !=, <=, >=)
# ----------------------------------------------------------------
print("1. ΒΑΣΙΚΟΙ ΤΕΛΕΣΤΕΣ ΣΥΓΚΡΙΣΗΣ")
print("-" * 50)

# Μεγαλύτερο από
print("df['Age'] > 28:")
print(df['Age'] > 28)
print()

# Φιλτράρισμα με βάση τη συνθήκη
print("df[df['Age'] > 28] (γραμμές με Age > 28):")
print(df[df['Age'] > 28][['Name', 'Age']])
print()

# Ισότητα
print("df['Gender'] == 'Female':")
print(df['Gender'] == 'Female')
print()

# Διάφορο
```

```
print("df[df['Department'] != 'Sales']:")
print(df[df['Department'] != 'Sales'][['Name', 'Department']])
print()

# Συνδυασμός συνθηκών με & (ΚΑΙ) και | (Η)
print("df[(df['Age'] >= 30) & (df['Gender'] == 'Male')]:")
print(df[(df['Age'] >= 30) & (df['Gender'] == 'Male')][['Name', 'Age',
'Gender']])
print()
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 2. ΜΕΘΟΔΟΙ ΣΥΓΚΡΙΣΗΣ (eq, ne, gt, ge, lt, le)
# ----------------------------------------------------------------------
print("2. ΜΕΘΟΔΟΙ ΣΥΓΚΡΙΣΗΣ")
print("-" * 50)

# eq() - ίσο με
print("df['Age'].eq(27):")
print(df['Age'].eq(27))
print()

# ne() - όχι ίσο
print("df[df['Department'].ne('IT')][['Name', 'Department']]:")
print(df[df['Department'].ne('IT')][['Name', 'Department']])
print()

# gt() - μεγαλύτερο από
print("df['Salary'].gt(60000):")
print(df['Salary'].gt(60000))
print()

# ge() - μεγαλύτερο ή ίσο
print("df['Age'].ge(30):")
print(df['Age'].ge(30))
print()

# lt() - μικρότερο από
print("df['Salary'].lt(55000):")
print(df['Salary'].lt(55000))
print()

# le() - μικρότερο ή ίσο
print("df['Age'].le(25):")
print(df['Age'].le(25))
print()

# Σύγκριση δύο στηλών με eq (π.χ. Age και μια νέα στήλη)
df['Age_copy'] = df['Age']  # δημιουργούμε αντίγραφο για σύγκριση
print("df['Age'].eq(df['Age_copy']) - σύγκριση στηλών:")
print(df['Age'].eq(df['Age_copy']))
print()
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 3. isin() - ΕΛΕΓΧΟΣ ΣΕ ΛΙΣΤΑ ΤΙΜΩΝ
# ----------------------------------------------------------------------
print("3. isin()")
print("-" * 50)

# Έλεγχος σε στήλη
```

```
cities = ['Athens', 'London', 'Paris']
print(f"df['City'].isin({cities}):")
print(df['City'].isin(cities))
print()

# Φιλτράρισμα με isin
print("df[df['City'].isin(cities)][['Name', 'City']]:")
print(df[df['City'].isin(cities)][['Name', 'City']])
print()

# Αντίστροφο με ~ (όχι)
print("df[~df['City'].isin(cities)][['Name', 'City']]:")
print(df[~df['City'].isin(cities)][['Name', 'City']])
print()

# isin() για έλεγχο index
print("df.index.isin(['Row1', 'Row3', 'Row5']):")
print(df.index.isin(['Row1', 'Row3', 'Row5']))
print()
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 4. equals() και compare() - ΣΥΓΚΡΙΣΗ DATAFRAMES
# ----------------------------------------------------------------------
print("4. equals() και compare()")
print("-" * 50)

# Δημιουργία αντιγράφων
df1 = df.copy()
df2 = df.copy()
df2.loc['Row2', 'Age'] = 99  # αλλαγή σε ένα κελί

# equals() - επιστρέφει ένα boolean
print("df1.equals(df1):", df1.equals(df1))
print("df1.equals(df2):", df1.equals(df2))
print()

# compare() - επιστρέφει DataFrame με διαφορές
print("df1.compare(df2):")
print(df1.compare(df2))
print()
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 5. any() και all() - ΕΛΕΓΧΟΣ Boolean ΣΤΟΙΧΕΙΩΝ
# ----------------------------------------------------------------------
print("5. any() και all()")
print("-" * 50)

# Έλεγχος αν υπάρχει τουλάχιστον ένα άτομο άνω των 60
print("(df['Age'] > 60).any():", (df['Age'] > 60).any())
print("(df['Age'] > 40).any():", (df['Age'] > 40).any())
print()

# Έλεγχος αν όλες οι ηλικίες είναι > 18
print("(df['Age'] > 18).all():", (df['Age'] > 18).all())
print()

# Σε boolean DataFrame (π.χ. αποτέλεσμα isna)
print("df.isna().any(axis=0) - στήλες με τουλάχιστον ένα NaN:")
print(df.isna().any(axis=0))
```

```
print("df.isna().all(axis=1) - γραμμές με όλα τα στοιχεία NaN (δεν υπάρχει καμία):")
print(df.isna().all(axis=1))
print()
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 6. ΜΕΘΟΔΟΙ STRING: contains, startswith, endswith
# ----------------------------------------------------------------------
print("6. ΜΕΘΟΔΟΙ STRING")
print("-" * 50)

# str.contains() - περιέχει υπόστρωμα (case=False για μη ευαισθησία πεζών-κεφαλαίων)
print("df[df['Name'].str.contains('an', case=False)][['Name']]:")
print(df[df['Name'].str.contains('an', case=False)][['Name']])
print()

# str.startswith()
print("df[df['City'].str.startswith('L', na=False)][['City']]:")
print(df[df['City'].str.startswith('L', na=False)][['City']])
print()

# str.endswith()
print("df[df['Name'].str.endswith('a', na=False)][['Name']]:")
print(df[df['Name'].str.endswith('a', na=False)][['Name']])
print()
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 7. isna() / isnull()  και  notna() / notnull()
# ----------------------------------------------------------------------
print("7. Έλεγχος ελλειπουσών τιμών")
print("-" * 50)

# isna()
print("df.isna():")
print(df.isna())
print()

# isnull() (ίδιο με isna)
print("df.isnull() (ίδιο):")
print(df.isnull())
print()

# notna()
print("df.notna():")
print(df.notna())
print()

# Φιλτράρισμα με isna()
print("df[df['City'].isna()][['Name', 'City']]:")
print(df[df['City'].isna()][['Name', 'City']])
print()

# Φιλτράρισμα με notna()
print("df[df['Salary'].notna()][['Name', 'Salary']]:")
print(df[df['Salary'].notna()][['Name', 'Salary']])
print()
print("\n" + "="*100 + "\n")
```

```
# ----------------------------------------------------------------------
# 8. query() - Φιλτράρισμα με σύνταξη σαν SQL
# ----------------------------------------------------------------------
print("8. query()")
print("-" * 50)

# Απλή συνθήκη
print("df.query('Age > 30'):")
print(df.query('Age > 30')[['Name', 'Age']])
print()

# Συνδυασμός συνθηκών
print("df.query('Age >= 25 and Gender == \"Female\"'):")
print(df.query('Age >= 25 and Gender == "Female"')[['Name', 'Age',
'Gender']])
print()

# Χρήση μεταβλητής με @
threshold = 60000
print(f"df.query('Salary > @threshold'):")
print(df.query('Salary > @threshold')[['Name', 'Salary']])
print()
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 9. isin() για index (το είδαμε ήδη στο 3, αλλά το επαναλαμβάνουμε για
πληρότητα)
# ----------------------------------------------------------------------
print("9. isin() για index")
print("-" * 50)
print("df.index.isin(['Row1', 'Row4', 'Row7']):")
print(df.index.isin(['Row1', 'Row4', 'Row7']))
print()

# ----------------------------------------------------------------------
# 10. ΠΑΡΑΔΕΙΓΜΑ ΣΥΝΔΥΑΣΜΟΥ ΜΕΘΟΔΩΝ
# ----------------------------------------------------------------------
print("10. ΣΥΝΔΥΑΣΜΟΣ ΜΕΘΟΔΩΝ")
print("-" * 50)

# Εύρεση ατόμων που είναι άνω των 30 και δεν είναι στο Sales
result = df[(df['Age'].gt(30)) & (df['Department'].ne('Sales'))]
print("Άτομα >30 και όχι Sales:")
print(result[['Name', 'Age', 'Department']])
print()

# Εύρεση πόλεων που περιέχουν 'o' ή 'i' χωρίς διάκριση πεζών-κεφαλαίων
pattern_cities = df[df['City'].str.contains('o|i', case=False, na=False)]
print("Πόλεις που περιέχουν 'o' ή 'i':")
print(pattern_cities[['Name', 'City']])
print()

print("\n" + "="*100 + "\n")
print("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
```

Κ. 3.4.12: Παράδειγμα για τις Μεθόδους για τις Σχέσεις Σύγκρισης των Πλαισίων Δεδομένων

3.4.12. Μοναδικές Τιμές & Συχνότητες

Μέθοδος	Περιγραφή	Παράδειγμα
unique()	Επιστρέφει έναν πίνακα (array) με τις μοναδικές τιμές μιας στήλης (Series). Δεν επιστρέφει συχνότητες, μόνο τις τιμές.	df['Χρώμα'].unique() *Επιστρέφει:* ['Κόκκινο' 'Μπλε' 'Πράσινο']
nunique()	Επιστρέφει τον *αριθμό* (integer) των μοναδικών τιμών σε μια στήλη ή σε ολόκληρο το DataFrame. Η παράμετρος dropna=False συμπεριλαμβάνει και τις τιμές NaN.	df['Χρώμα'].nunique() *Επιστρέφει:* 3 df.nunique() *Επιστρέφει το πλήθος μοναδικών τιμών για κάθε στήλη.*
value_counts()	Υπολογίζει μια συχνότητα (frequency table). Επιστρέφει μια Series με τις μοναδικές τιμές ως δείκτη (index) και το πλήθος εμφάνισής τους ως τιμές. Διαθέτει παραμέτρους όπως normalize=True για σχετικές συχνότητες (ποσοστά), sort=False, dropna=False.	df['Χρώμα'].value_counts() *Επιστρέφει:* Μπλε 12 Κόκκινο 8 Πράσινο 5 df['Χρώμα'].value_counts(normalize=True) *Επιστρέφει ποσοστά.*
groupby().size() **ή** groupby().count()	Χρησιμοποιείται για την εύρεση μοναδικών συνδυασμών τιμών από *δύο ή περισσότερες στήλες* και την καταμέτρηση της συχνότητάς τους. Η size() επιστρέφει το πλήθος κάθε ομάδας, λαμβάνοντας υπόψη και τις τιμές NaN (ως ξεχωριστή ομάδα). Η count() αποκλείει τις NaN.	df.groupby(['Επώνυμο', 'Όνομα']).size().reset_index(name='Συχνότητα') *Δημιουργεί ένα νέο DataFrame με στήλες: Επώνυμο, Όνομα, Συχνότητα.*
pd.crosstab()	Συνάρτηση για τη δημιουργία ενός πίνακα διασταύρωσης (cross-tabulation), που υπολογίζει τη συχνότητα εμφάνισης των συνδυασμών τιμών μεταξύ δύο ή περισσότερων στηλών. Είναι πιο κοντά στην έννοια του "pivot table" για συχνότητες.	pd.crosstab(df['Φύλο'], df['Χρώμα']) *Δημιουργεί έναν πίνακα όπου οι γραμμές είναι τα μοναδικά 'Φύλο', οι στήλες τα μοναδικά 'Χρώμα', και τα κελιά περιέχουν το πλήθος εμφάνισης.*
describe()	Παρέχει μια γρήγορη σύνοψη των δεδομένων. Για στήλες με αντικείμενα (object) ή κατηγορικά δεδομένα, επιστρέφει: count, unique, top (η πιο συχνή τιμή) και freq (η συχνότητα της πιο συχνής τιμής).	df['Χρώμα'].describe() *Επιστρέφει:* count 25 unique 3 top Μπλε freq 12

Πίνακας 3.4.13: Μέθοδοι για την Ανάλυση Μοναδικότητας και Συχνότητας των Πλαισίων Δεδομένων

Σε αυτή την ενότητα εξετάζουμε τις μεθόδους **Μοναδικότητας και Συχνότητας**. Αυτές οι συναρτήσεις είναι τα "μάτια" μας όταν προσπαθούμε να καταλάβουμε την ποιότητα και την ποικιλία των κατηγορικών δεδομένων (π.χ. Πόλεις, Προϊόντα, Κατηγορίες).

1. Μοναδικές Τιμές: unique() και nunique()

Αυτές οι μέθοδοι μας λένε "τι υπάρχει" και "πόσα διαφορετικά πράγματα υπάρχουν" στη στήλη μας.

- **unique()**: Επιστρέφει ένα **NumPy array**. Είναι χρήσιμο όταν θέλεις να δεις γρήγορα αν υπάρχουν ορθογραφικά λάθη (π.χ. να δεις "Αθήνα" και "Αθηνα" ως δύο διαφορετικές τιμές).
- **nunique()**: Επιστρέφει έναν **αριθμό**. Είναι εξαιρετικά χρήσιμο για να ελέγξεις αν μια στήλη μπορεί να χρησιμεύσει ως "Primary Key" (αν ο αριθμός μοναδικών τιμών ισούται με τον αριθμό των γραμμών).

2. Συχνότητες: value_counts()

Αυτή είναι ίσως η πιο αγαπημένη μέθοδος των αναλυτών. Μετατρέπει μια στήλη σε ένα γρήγορο στατιστικό δελτίο.

- **normalize=True**: Αντί για απόλυτα νούμερα (10, 20, 30), σου δίνει ποσοστά (0.1, 0.2, 0.3). Είναι ο πιο γρήγορος τρόπος να πεις "Το 40% των πωλήσεων είναι από το Μπλε χρώμα".
- **sort=True**: Από προεπιλογή, η Pandas βάζει την πιο συχνή τιμή πρώτη. Αυτό σε βοηθά να εντοπίσεις αμέσως τους "πρωταγωνιστές" του dataset σου.

3. Σύνθετες Συχνότητες: groupby() και pd.crosstab()

Όταν η μία στήλη δεν αρκεί και θέλεις να δεις πώς συνδυάζονται δύο μεταβλητές.

- **groupby().size()**: Σου δείχνει πόσες φορές εμφανίζεται κάθε συνδυασμός.
    - *Παράδειγμα:* Πόσοι "Παπαδόπουλοι" ονομάζονται "Γιάννης".
- **pd.crosstab()**: Είναι η "βασίλισσα" της σύγκρισης κατηγοριών. Δημιουργεί έναν πίνακα δύο διαστάσεων που είναι πολύ πιο εύκολο να διαβαστεί από έναν απλό κατάλογο.
    - *Παράδειγμα:* Ένας πίνακας όπου οι γραμμές είναι "Άνδρες/Γυναίκες" και οι στήλες "iPhone/Android". Τα κελιά θα σου πουν ακριβώς πόσοι ανήκουν σε κάθε κατηγορία.

4. describe() για Κατηγορικά Δεδομένα

Πολλοί χρησιμοποιούν την describe() μόνο για αριθμούς (μέσος όρος, κ.λπ.). Όμως, αν την εφαρμόσεις σε στήλη με κείμενο, σου δίνει 4 κρίσιμα στοιχεία:

1. **count**: Πόσες μη-κενές τιμές υπάρχουν.
2. **unique**: Πόσες διαφορετικές κατηγορίες.
3. **top**: Ποια είναι η επικρατούσα τιμή (το "mode").
4. **freq**: Πόσες φορές εμφανίζεται η επικρατούσα τιμή.

```
import pandas as pd
import numpy as np

# ----------------------------------------------------------------
```

```
# Δημιουργία δείγματος DataFrame με κατηγορικές και αριθμητικές στήλες
# ----------------------------------------------------------------------
print("ΔΗΜΙΟΥΡΓΙΑ ΔΕΙΓΜΑΤΟΣ DataFrame")
print("-" * 70)

# Τυχαίες τιμές για αναπαραγωγιμότητα
# Τυχαίες τιμές για αναπαραγωγιμότητα
np.random.seed(42)

data = {
    'Color': np.random.choice(['Red', 'Blue', 'Green', 'Yellow'], size=20,
p=[0.4, 0.3, 0.2, 0.1]),
    'Size': np.random.choice(['S', 'M', 'L', 'XL'], size=20),
    'Price': np.random.randint(10, 100, size=20).astype(float),   # <--
Διόρθωση εδώ
    'Category': np.random.choice(['A', 'B', 'C'], size=20),
    'InStock': np.random.choice([True, False], size=20),
    'Customer': np.random.choice(['John', 'Anna', 'George', 'Eleni',
'Nikos'], size=20)
}
# Εισαγωγή μερικών NaN
data['Color'][5] = np.nan
data['Size'][10] = np.nan
data['Price'][15] = np.nan   # Τώρα δουλεύει χωρίς σφάλμα

df = pd.DataFrame(data)
print("Αρχικό DataFrame (πρώτες 10 γραμμές):")
print(df.head(10))
print("\nΠληροφορίες στηλών:")
print(df.dtypes)
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 1. unique() - Μοναδικές τιμές μιας στήλης (επιστρέφει array)
# ----------------------------------------------------------------------
print("1. unique()")
print("-" * 50)

# unique() σε κατηγορική στήλη
unique_colors = df['Color'].unique()
print("df['Color'].unique():")
print(unique_colors)
print("Τύπος επιστροφής:", type(unique_colors))
print()

# unique() σε αριθμητική στήλη (αν και συνήθως χρησιμοποιείται σε
κατηγορικές)
unique_prices = df['Price'].unique()
print("df['Price'].unique() (μερικές τιμές):")
print(unique_prices[:5], "...")  # δείχνουμε μόνο τις πρώτες 5
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 2. nunique() - Πλήθος μοναδικών τιμών
# ----------------------------------------------------------------------
print("2. nunique()")
print("-" * 50)

# Πλήθος μοναδικών σε μια στήλη (προεπιλογή dropna=True)
print("df['Color'].nunique() (dropna=True):", df['Color'].nunique())
```

```
print("df['Color'].nunique(dropna=False) (συμπεριλαμβάνει NaN):",
df['Color'].nunique(dropna=False))
print()

# nunique() σε όλο το DataFrame
print("df.nunique() (ανά στήλη, dropna=True):")
print(df.nunique())
print()

print("df.nunique(dropna=False):")
print(df.nunique(dropna=False))
print("\n" + "="*100 + "\n")

# ------------------------------------------------------------------
# 3. value_counts() - Πίνακας συχνοτήτων
# ------------------------------------------------------------------
print("3. value_counts()")
print("-" * 50)

# Απλή καταμέτρηση
print("df['Color'].value_counts():")
print(df['Color'].value_counts())
print()

# Συμπερίληψη NaN
print("df['Color'].value_counts(dropna=False):")
print(df['Color'].value_counts(dropna=False))
print()

# Σχετικές συχνότητες (ποσοστά)
print("df['Color'].value_counts(normalize=True):")
print(df['Color'].value_counts(normalize=True))
print()

# Ταξινόμηση κατά αλφαβητική σειρά (όχι κατά συχνότητα)
print("df['Color'].value_counts(sort=False):")
print(df['Color'].value_counts(sort=False))
print()

# value_counts() σε αριθμητική στήλη (γίνονται ομάδες ανά τιμή)
print("df['Price'].value_counts().head():")
print(df['Price'].value_counts().head())
print("\n" + "="*100 + "\n")

# ------------------------------------------------------------------
# 4. groupby().size() και groupby().count() - Συχνότητες συνδυασμών
# ------------------------------------------------------------------
print("4. groupby().size() και groupby().count()")
print("-" * 50)

# groupby().size() - επιστρέφει Series με πλήθος γραμμών ανά ομάδα
(συμπεριλαμβάνει NaN ως ομάδα)
print("df.groupby(['Color', 'Size']).size():")
print(df.groupby(['Color', 'Size']).size())
print()

# Για καλύτερη παρουσίαση, μετατροπή σε DataFrame
size_df = df.groupby(['Color', 'Size']).size().reset_index(name='Count')
print("Ως DataFrame με reset_index():")
print(size_df)
print()
```

```
# groupby().count() - επιστρέφει DataFrame με πλήθος μη-κενών τιμών ανά
στήλη για κάθε ομάδα.
# Για να πάρουμε το πλήθος των γραμμών (όπως size) μπορούμε να
χρησιμοποιήσουμε οποιαδήποτε στήλη.
print("df.groupby(['Color', 'Size']).count() (επιστρέφει πολλές στήλες):")
print(df.groupby(['Color', 'Size']).count())
print()

# Συνήθως χρησιμοποιούμε μία στήλη, π.χ. 'Price', για να πάρουμε το count
(ίδιο με size αλλά αγνοεί NaN στην Price)
print("df.groupby(['Color', 'Size'])['Price'].count():")
print(df.groupby(['Color', 'Size'])['Price'].count())
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 5. pd.crosstab() - Πίνακας διασταύρωσης
# ----------------------------------------------------------------------
print("5. pd.crosstab()")
print("-" * 50)

# Απλός πίνακας διασταύρωσης μεταξύ δύο στηλών
crosstab_result = pd.crosstab(df['Color'], df['Size'])
print("pd.crosstab(df['Color'], df['Size']):")
print(crosstab_result)
print()

# Προσθήκη περιθωρίων (σύνολα)
crosstab_margins = pd.crosstab(df['Color'], df['Size'], margins=True,
margins_name='Σύνολο')
print("Με margins=True:")
print(crosstab_margins)
print()

# Κανονικοποίηση (ποσοστά)
crosstab_norm = pd.crosstab(df['Color'], df['Size'], normalize='index')  #
ανά γραμμή
print("normalize='index' (ποσοστά ανά γραμμή):")
print(crosstab_norm.round(2))
print()

crosstab_norm_col = pd.crosstab(df['Color'], df['Size'],
normalize='columns')
print("normalize='columns' (ποσοστά ανά στήλη):")
print(crosstab_norm_col.round(2))
print()

# Crosstab με τρεις διαστάσεις (επιστρέφει DataFrame με MultiIndex)
crosstab_3d = pd.crosstab([df['Color'], df['Size']], df['Category'])
print("pd.crosstab([df['Color'], df['Size']], df['Category']):")
print(crosstab_3d)
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 6. describe() για κατηγορικές στήλες
# ----------------------------------------------------------------------
print("6. describe() για κατηγορικές στήλες")
print("-" * 50)

# Για κατηγορική στήλη (object ή category), η describe() επιστρέφει:
# count, unique, top, freq
```

```
print("df['Color'].describe():")
print(df['Color'].describe())
print()

# Για όλες τις στήλες, η describe() συμπεριλαμβάνει μόνο αριθμητικές, εκτός αν δηλωθεί include='all'
print("df.describe(include='all') (περιλαμβάνει κατηγορικές):")
print(df.describe(include='all').T)  # .T για καλύτερη εμφάνιση
print()

# Για να δούμε μόνο κατηγορικές: include=['object', 'category']
print("df.describe(include=['object', 'bool']):")
print(df.describe(include=['object', 'bool']))
print("\n" + "="*100 + "\n")

# ------------------------------------------------------------------
# 7. Πρόσθετο: value_counts() για πολλές στήλες (με groupby)
# ------------------------------------------------------------------
print("7. value_counts() για πολλές στήλες (ισοδύναμο groupby().size())")
print("-" * 50)

# Στις σύγχρονες εκδόσεις pandas, η value_counts() μπορεί να εφαρμοστεί σε DataFrame
print("df[['Color', 'Size']].value_counts():")
print(df[['Color', 'Size']].value_counts())
print()

print("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
```

Κ. 3.4.13: Παράδειγμα Μεθόδων για την Ανάλυση Μοναδικότητας και Συχνότητας των Πλαισίων Δεδομένων

3.4.13. Μέθοδοι Κειμένου (str accessor)

Μέθοδος	Περιγραφή	Παράδειγμα
Μετατροπή Κεφαλαίων/Πεζών (Case Conversion)		
lower()	Μετατρέπει όλους τους χαρακτήρες σε πεζούς.	df['Όνομα'].str.lower()
upper()	Μετατρέπει όλους τους χαρακτήρες σε κεφαλαίους.	df['Χώρα'].str.upper()
capitalize()	Κάνει τον πρώτο χαρακτήρα κεφαλαίο και τους υπόλοιπους πεζούς.	df['Περιγραφή'].str.capitalize()
title()	Κάνει το πρώτο γράμμα κάθε λέξης κεφαλαίο.	df['Διεύθυνση'].str.title()
swapcase()	Αλλάζει τους πεζούς σε κεφαλαίους και τους κεφαλαίους σε πεζούς.	df['Κείμενο'].str.swapcase()
Αφαίρεση & Συμπλήρωση Κενών (Whitespace Removal & Padding)		

Μέθοδος	Περιγραφή	Παράδειγμα
strip()	Αφαιρεί τα κενά (whitespace) από την αρχή και το τέλος της συμβολοσειράς.	df['Όνομα'].str.strip()
lstrip()	Αφαιρεί τα κενά μόνο από την αρχή (αριστερά) της συμβολοσειράς.	df['Χρήστης'].str.lstrip()
rstrip()	Αφαιρεί τα κενά μόνο από το τέλος (δεξιά) της συμβολοσειράς.	df['Σχόλιο'].str.rstrip()
pad()	Προσθέτει χαρακτήρες (από προεπιλογή κενά) στην αριστερή, δεξιά ή και στις δύο πλευρές μέχρι να φτάσει ένα συγκεκριμένο μήκος.	df['Κωδικός'].str.pad(width=10, side='left', fillchar='0')
center()	Κεντράρει τη συμβολοσειρά προσθέτοντας χαρακτήρες (π.χ. κενά) εκατέρωθεν.	df['Τίτλος'].str.center(20, '-')
zfill()	Συμπληρώνει την αριστερή πλευρά της συμβολοσειράς με μηδενικά.	df['Ταχυδρομικός_Κώδικας'].str.zfill(5)
Διαχωρισμός & Αντικατάσταση (Splitting & Replacing)		
split(pat, expand)	Διαχωρίζει κάθε συμβολοσειρά με βάση ένα διαχωριστικό (pat). Αν expand=True, επιστρέφει DataFrame.	df[['Όνομα', 'Επίθετο']] = df['Πλήρες_Όνομα'].str.split(' ', expand=True)
rsplit(pat, expand)	Όμοιο με το split, αλλά ξεκινά από τα δεξιά.	df['Διαδρομή'].str.rsplit('/', n=1, expand=True)
replace(pat, repl, regex)	Αντικαθιστά ένα μοτίβο (pat) με μια άλλη συμβολοσειρά (repl). Υποστηρίζει κανονικές εκφράσεις (regex).	df['Ημερομηνία'].str.replace('/', '-', regex=False)
repeat(repeats)	Επαναλαμβάνει κάθε συμβολοσειρά συγκεκριμένο αριθμό φορών.	df['Αστερίσκος'].str.repeat(3) *# π.χ. το 'A' γίνεται 'AAA'*
cat(sep, na_rep)	Συνενώνει τα στοιχεία μιας σειράς (Series) σε μια ενιαία συμβολοσειρά.	df['Λέξη'].str.cat(sep=', ')
get_dummies()	Διαχωρίζει κάθε συμβολοσειρά σε ψευδομεταβλητές (dummy variables).	df['Χρώμα'].str.get_dummies(sep=',')
Εξαγωγή & Αναζήτηση με Κανονικές Εκφράσεις (Regex)		

Μέθοδος	Περιγραφή	Παράδειγμα
extract(pat, expand)	Εξάγει ομάδες από μια κανονική έκφραση (pat) ως νέες στήλες.	df[['Πόλη', 'ΤΚ']] = df['Τοποθεσία'].str.extract(r'(.+), (\d{5})')
extractall(pat)	Εξάγει όλες τις ομάδες από κάθε συμβολοσειρά (όταν ένα μοτίβο εμφανίζεται πολλές φορές).	df['Σημειώσεις'].str.extractall(r'#(\w+)')
contains(pat, case, regex)	Ελέγχει αν κάθε συμβολοσειρά περιέχει ένα συγκεκριμένο μοτίβο (pat). Επιστρέφει boolean Series.	df[df['Σχόλιο'].str.contains('καλός', case=False)]
match(pat)	Ελέγχει αν κάθε συμβολοσειρά **ξεκινά** με ένα συγκεκριμένο μοτίβο (pat).	df['Κωδικός_Προϊόντος'].str.match(r'^[A-Z]{3}')
findall(pat)	Βρίσκει όλες τις εμφανίσεις ενός μοτίβου (pat) σε κάθε συμβολοσειρά.	df['Κείμενο'].str.findall(r'\d+') *# Βρίσκει όλους τους αριθμούς*
count(pat)	Μετράει πόσες φορές εμφανίζεται ένα μοτίβο (pat) σε κάθε συμβολοσειρά.	df['Περίοδος'].str.count(r';')
Έλεγχοι Περιεχομένου & Μήκος		
len()	Υπολογίζει το μήκος (αριθμό χαρακτήρων) κάθε συμβολοσειράς.	df['Σχόλιο'].str.len().max()
startswith(pat)	Ελέγχει αν κάθε συμβολοσειρά ξεκινά με ένα συγκεκριμένο μοτίβο.	df['Χώρα'].str.startswith('Ελ')
endswith(pat)	Ελέγχει αν κάθε συμβολοσειρά τελειώνει με ένα συγκεκριμένο μοτίβο.	df['Email'].str.endswith('.gr')
isalnum()	Ελέγχει αν όλοι οι χαρακτήρες είναι αλφαριθμητικοί (γράμματα ή αριθμοί).	df['Κωδικός'].str.isalnum()
isalpha()	Ελέγχει αν όλοι οι χαρακτήρες είναι αλφαβητικοί.	df['Όνομα'].str.isalpha()
isdigit()	Ελέγχει αν όλοι οι χαρακτήρες είναι αριθμοί.	df['Ηλικία_ως_κείμενο'].str.isdigit()
isnumeric()	Παρόμοιο με το isdigit, αλλά αναγνωρίζει και άλλους αριθμητικούς χαρακτήρες (π.χ. κλάσματα).	df['Ποσότητα'].str.isnumeric()
isspace()	Ελέγχει αν όλοι οι χαρακτήρες είναι κενά (whitespace).	df['Κενή_Στήλη'].str.isspace()
islower()	Ελέγχει αν όλοι οι χαρακτήρες είναι πεζοί.	df['Οδηγία'].str.islower()

Μέθοδος	Περιγραφή	Παράδειγμα
isupper()	Ελέγχει αν όλοι οι χαρακτήρες είναι κεφαλαίοι.	df['Τίτλος'].str.isupper()
istitle()	Ελέγχει αν κάθε συμβολοσειρά έχει μορφή τίτλου (κάθε λέξη ξεκινά με κεφαλαίο).	df['Βιβλίο'].str.istitle()
Αναζήτηση με Ευρετήριο (Indexing)		
get(i)	Παίρνει το στοιχείο στη θέση i από κάθε συμβολοσειρά (αν η συμβολοσειρά είναι λίστα ή διαχωρισμένη).	df['Ημερομηνία'].str.split('-').str.get(1) # *Παίρνει τον μήνα*
find(sub)	Επιστρέφει τη χαμηλότερη θέση (index) όπου βρέθηκε το sub, ή -1 αν δεν βρέθηκε.	df['Κείμενο'].str.find('σπουδαίος')
rfind(sub)	Επιστρέφει την υψηλότερη θέση (index) όπου βρέθηκε το sub, ή -1 αν δεν βρέθηκε.	df['Κείμενο'].str.rfind('τέλος')
index(sub)	Παρόμοιο με το find, αλλά επιστρέφει ValueError αν δεν βρεθεί το μοτίβο.	df['Αναγνωριστικό'].str.index('Χ')
rindex(sub)	Παρόμοιο με το rfind, αλλά επιστρέφει ValueError αν δεν βρεθεί το μοτίβο.	df['Αναγνωριστικό'].str.rindex('Χ')
slice(start, stop)	Κόβει (slice) κάθε συμβολοσειρά.	df['Χρόνος'].str.slice(0, 4) # *Παίρνει τους πρώτους 4 χαρακτήρες (έτος)*
slice_replace(start, stop, repl)	Αντικαθιστά ένα τμήμα της συμβολοσειράς.	df['Τηλέφωνο'].str.slice_replace(0, 3, '***')

Πίνακας 3.4.14: Μέθοδοι επεξεργασίας αλφαριθμητικών (Κειμένου) στα Πλαίσια Δεδομένων

Αυτή είναι η πιο ολοκληρωμένη λίστα μεθόδων για τον χειρισμό κειμένου (**String Methods**) στην Pandas. Όταν εργαζόμαστε με δεδομένα, οι στήλες τύπου object (κείμενο) είναι συχνά οι πιο "ακατάστατες". Η πρόσβαση σε αυτές τις μεθόδους γίνεται πάντα μέσω του **.str** accessor.

1. Καθαρισμός και Μορφοποίηση (Case & Whitespace)

Αυτές οι μέθοδοι είναι το πρώτο βήμα στο **Data Cleaning**.

- **strip()**: Απαραίτητη μετά από εισαγωγή δεδομένων από CSV ή Excel. Συχνά υπάρχουν κρυμμένα κενά (π.χ. "Αθήνα ") που εμποδίζουν τα φίλτρα να λειτουργήσουν.
- **capitalize() vs title()**: Χρησιμοποίησε την title() για ονοματεπώνυμα ή διευθύνσεις και την capitalize() για προτάσεις.
- **zfill()**: Η καλύτερη μέθοδος για τη διόρθωση κωδικών ή ΤΚ. Αν ένας ΤΚ είναι 1534, το zfill(5) θα τον κάνει 01534.

2. Διαχωρισμός και Εξαγωγή (Splitting & Regex)

Εδώ μετατρέπουμε μια στήλη σε πολλές, εξάγοντας την ουσία της πληροφορίας.

- **split(expand=True)**: Αν έχεις μια στήλη "Ονοματεπώνυμο", με αυτή τη μέθοδο τη σπας σε δύο αυτόνομα πεδία.
- **extract()**: **Πανίσχυρο εργαλείο.** Χρησιμοποιώντας Κανονικές Εκφράσεις (Regex), μπορείς να "τραβήξεις" συγκεκριμένα μοτίβα, όπως emails ή τηλέφωνα, μέσα από μεγάλα κείμενα.
- **get_dummies()**: Μετατρέπει κείμενο (π.χ. "Κόκκινο, Μπλε") σε στήλες 0 και 1. Είναι η βάση για την προετοιμασία δεδομένων για **Machine Learning** (One-Hot Encoding).

3. Αναζήτηση και Έλεγχος (Search & Validation)

Χρησιμοποιούνται κυρίως για το φιλτράρισμα των δεδομένων.

- **contains()**: Η πιο συχνή μέθοδος αναζήτησης.

Tip: Χρησιμοποίησε πάντα na=False (π.χ. str.contains('abc', na=False)) για να αποφύγεις σφάλματα αν η στήλη περιέχει κενά κελιά (NaN).

- **isalnum(), isdigit(), κ.λπ.**: Ιδανικές για **Data Validation**. Μπορείς να βρεις γρήγορα αν σε μια στήλη που έπρεπε να έχει μόνο αριθμούς, κάποιος εισήγαγε κατά λάθος γράμματα.

4. Indexing και Slicing

Λειτουργούν ακριβώς όπως το κλασικό slicing της Python ([start:stop]), αλλά εφαρμόζονται σε ολόκληρη τη στήλη ταυτόχρονα.

- **slice(0, 4)**: Πολύ χρήσιμο για ημερομηνίες που είναι αποθηκευμένες ως κείμενο (π.χ. "2023-10-05") για να πάρεις μόνο το έτος.
- **get(i)**: Αν έχεις κάνει split, το get(0) παίρνει το πρώτο κομμάτι, το get(-1) το τελευταίο.

```
import pandas as pd
import numpy as np

# ---------------------------------------------------------------------
# Δημιουργία δείγματος DataFrame με διάφορες στήλες συμβολοσειρών
# ---------------------------------------------------------------------
print("ΔΗΜΙΟΥΡΓΙΑ ΔΕΙΓΜΑΤΟΣ DataFrame")
print("-" * 70)

data = {
    'Name': ['  Anna ', 'GEORGE', 'maria', '  NIKOS  ', 'Eleni', None, '
john  '],
    'City': ['Athens ', 'thessaloniki', ' PATRAS', 'heraklion  ', ' Volos
', 'Chania', 'Larisa'],
    'Code': ['001', '002', '003', '004', '005', '006', '007'],
    'Address': ['Main St 10, Athens', 'Park Ave 5, Thessaloniki', 'Ocean
Blvd, Patras',
                'Central Square, Heraklion', 'El. Venizelou 15, Volos',
'Akti 7, Chania', 'Hillside, Larisa'],
    'Date': ['2023-01-15', '2023/02/20', '2023.03.25', '2023-04-10',
'2023/05/05', '2023.06.18', '2023-07-30'],
    'Email': ['anna@example.com', 'george@example.org', 'maria@example.gr',
'nikos@example.com',
              'eleni@example.gr', None, 'john@example.com'],
    'Text': ['Το προϊόν είναι εξαιρετικό!', 'Καλή ποιότητα, προτείνεται.',
             'Μέτριο αποτέλεσμα', 'Τέλειο! 100% ικανοποιημένος',
```

```
            'Δεν άξιζε τα λεφτά του', 'Γρήγορη παράδοση, ευχαριστώ', 'Θα
το ξαναγοράσω'],
    'Mixed': ['abc123', 'XYZ456', '123', 'abc', '!@#', 'A1B2C3', '    '],
    'Tags': ['tech,gadget,new', 'home,kitchen', 'clothing,mens,women',
             'books,education', 'sports,outdoor', 'toys,kids',
'automotive']
}
df = pd.DataFrame(data)
print("Αρχικό DataFrame (πρώτες 5 γραμμές):")
print(df.head())
print("\nΠληροφορίες στηλών:")
print(df.dtypes)
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 1. ΜΕΤΑΤΡΟΠΗ ΚΕΦΑΛΑΙΩΝ/ΠΕΖΩΝ (CASE CONVERSION)
# ----------------------------------------------------------------------
print("1. ΜΕΤΑΤΡΟΠΗ ΚΕΦΑΛΑΙΩΝ/ΠΕΖΩΝ")
print("-" * 50)

# lower() - όλοι πεζοί
print("lower() στην 'Name':")
print(df['Name'].str.lower())
print()

# upper() - όλοι κεφαλαίοι
print("upper() στην 'City':")
print(df['City'].str.upper())
print()

# capitalize() - πρώτος κεφαλαίος, υπόλοιποι πεζοί
print("capitalize() στην 'Text':")
print(df['Text'].str.capitalize())
print()

# title() - κάθε λέξη με κεφαλαίο πρώτο γράμμα
print("title() στην 'Address':")
print(df['Address'].str.title())
print()

# swapcase() - αντιστροφή πεζών/κεφαλαίων
print("swapcase() στην 'Name':")
print(df['Name'].str.swapcase())
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 2. ΑΦΑΙΡΕΣΗ & ΣΥΜΠΛΗΡΩΣΗ ΚΕΝΩΝ (WHITESPACE REMOVAL & PADDING)
# ----------------------------------------------------------------------
print("2. ΑΦΑΙΡΕΣΗ & ΣΥΜΠΛΗΡΩΣΗ ΚΕΝΩΝ")
print("-" * 50)

# strip() - αφαίρεση κενών από αρχή και τέλος
print("strip() στην 'Name' (πριν/μετά):")
print("Πριν:", repr(df['Name'].iloc[0]))
print("Μετά:", repr(df['Name'].str.strip().iloc[0]))
print()

# lstrip() - αφαίρεση από αριστερά
print("lstrip() στην 'City':")
print(df['City'].str.lstrip().tolist())
print()
```

```
# rstrip() - αφαίρεση από δεξιά
print("rstrip() στην 'City':")
print(df['City'].str.rstrip().tolist())
print()

# pad() - προσθήκη χαρακτήρων για σταθερό μήκος
print("pad() width=10, side='left', fillchar='0' στην 'Code':")
print(df['Code'].str.pad(width=10, side='left', fillchar='0'))
print()

# center() - κεντράρισμα με χαρακτήρες
print("center(15, '*') στην 'Name' (μετά από strip):")
print(df['Name'].str.strip().str.center(15, '*'))
print()

# zfill() - συμπλήρωση με μηδενικά αριστερά
print("zfill(5) στην 'Code' (ήδη έχουν μήκος 3):")
print(df['Code'].str.zfill(5))
print("\n" + "="*100 + "\n")

# ---------------------------------------------------------------------
# 3. ΔΙΑΧΩΡΙΣΜΟΣ & ΑΝΤΙΚΑΤΑΣΤΑΣΗ (SPLITTING & REPLACING)
# ---------------------------------------------------------------------
print("3. ΔΙΑΧΩΡΙΣΜΟΣ & ΑΝΤΙΚΑΤΑΣΤΑΣΗ")
print("-" * 50)

# split() - διαχωρισμός με expand=True για DataFrame
print("split() της 'Address' με διαχωριστικό ',', expand=True:")
address_split = df['Address'].str.split(',', expand=True)
address_split.columns = ['Street', 'City_From_Address']
print(address_split)
print()

# rsplit() - διαχωρισμός από δεξιά (π.χ. για τελευταία λέξη)
print("rsplit() της 'Address' με n=1, expand=True:")
print(df['Address'].str.rsplit(' ', n=1, expand=True))
print()

# replace() - αντικατάσταση χαρακτήρων
print("replace() της 'Date': αντικατάσταση '/' και '.' με '-'")
print(df['Date'].str.replace('/', '-', regex=False).str.replace('.', '-',
regex=False))
print()

# repeat() - επανάληψη συμβολοσειράς
print("repeat(2) στην 'Name' (μετά από strip):")
print(df['Name'].str.strip().str.repeat(2))
print()

# cat() - συνένωση όλων των στοιχείων μιας στήλης
print("cat(sep=', ') των 'Name' (αγνοώντας NaN):")
print(df['Name'].str.cat(sep=', ', na_rep='_'))
print()

# get_dummies() - ψευδομεταβλητές από διαχωρισμένες τιμές
print("get_dummies(sep=',') για 'Tags':")
dummies = df['Tags'].str.get_dummies(sep=',')
print(dummies)
print()
print("\n" + "="*100 + "\n")
```

```
# ---------------------------------------------------------------------
# 4. ΕΞΑΓΩΓΗ & ΑΝΑΖΗΤΗΣΗ ΜΕ ΚΑΝΟΝΙΚΕΣ ΕΚΦΡΑΣΕΙΣ (REGEX)
# ---------------------------------------------------------------------
print("4. ΕΞΑΓΩΓΗ & ΑΝΑΖΗΤΗΣΗ ΜΕ REGEX")
print("-" * 50)

# extract() - εξαγωγή ομάδων ως στήλες
print("extract() από 'Address': μοτίβο (.*), (.*)")
extracted = df['Address'].str.extract(r'(.*),\s*(.*)')
extracted.columns = ['Street_Extract', 'City_Extract']
print(extracted)
print()

# extractall() - εξαγωγή όλων των εμφανίσεων (εδώ από 'Text' βρίσκουμε
λέξεις)
print("extractall() από 'Text' (όλες οι λέξεις):")
# Απλό παράδειγμα: βρίσκει λέξεις (ακολουθίες γραμμάτων)
all_words = df['Text'].str.extractall(r'(\w+)')
print(all_words)
print()

# contains() - έλεγχος ύπαρξης μοτίβου (boolean)
print("contains('καλ', case=False) στην 'Text':")
print(df['Text'].str.contains('καλ', case=False, na=False))
print()

# match() - έλεγχος αν ξεκινά με μοτίβο
print("match(r'^[A-Z]') στην 'Name' (μετά από strip):")
print(df['Name'].str.strip().str.match(r'^[A-Z]', na=False))
print()

# findall() - εύρεση όλων των αριθμών
print("findall(r'\\d+') στην 'Mixed':")
print(df['Mixed'].str.findall(r'\d+'))
print()

# count() - πλήθος εμφανίσεων μοτίβου
print("count(r'[aeiou]') (φωνήεντα) στην 'Name' (πεζά):")
print(df['Name'].str.lower().str.count(r'[aeiou]'))
print("\n" + "="*100 + "\n")

# ---------------------------------------------------------------------
# 5. ΕΛΕΓΧΟΙ ΠΕΡΙΕΧΟΜΕΝΟΥ & ΜΗΚΟΣ
# ---------------------------------------------------------------------
print("5. ΕΛΕΓΧΟΙ ΠΕΡΙΕΧΟΜΕΝΟΥ & ΜΗΚΟΣ")
print("-" * 50)

# len() - μήκος συμβολοσειράς
print("len() στην 'Name' (μετά από strip):")
print(df['Name'].str.strip().str.len())
print()

# startswith() - ελέγχει αν ξεκινά με
print("startswith('A') στην 'Name' (strip):")
print(df['Name'].str.strip().str.startswith('A', na=False))
print()

# endswith() - ελέγχει αν τελειώνει με
print("endswith('.gr') στην 'Email':")
print(df['Email'].str.endswith('.gr', na=False))
```

```python
print()

# isalnum() - αλφαριθμητικοί χαρακτήρες
print("isalnum() στην 'Mixed':")
print(df['Mixed'].str.isalnum())
print()

# isalpha() - μόνο γράμματα
print("isalpha() στην 'Name' (μετά από strip):")
print(df['Name'].str.strip().str.isalpha())
print()

# isdigit() - μόνο αριθμοί
print("isdigit() στην 'Code':")
print(df['Code'].str.isdigit())
print()

# isnumeric() - αριθμητικοί χαρακτήρες (π.χ. και κλάσματα)
print("isnumeric() στην 'Mixed':")
print(df['Mixed'].str.isnumeric())
print()

# isspace() - μόνο κενά
print("isspace() στην 'Mixed':")
print(df['Mixed'].str.isspace())
print()

# islower() - όλοι πεζοί
print("islower() στην 'Name' (χωρίς strip):")
print(df['Name'].str.islower())
print()

# isupper() - όλοι κεφαλαίοι
print("isupper() στην 'Name':")
print(df['Name'].str.isupper())
print()

# istitle() - μορφή τίτλου (κάθε λέξη κεφαλαίο πρώτο)
print("istitle() στην 'Address':")
print(df['Address'].str.istitle())
print("\n" + "="*100 + "\n")

# ---------------------------------------------------------------------
# 6. ΑΝΑΖΗΤΗΣΗ ΜΕ ΕΥΡΕΤΗΡΙΟ (INDEXING)
# ---------------------------------------------------------------------
print("6. ΑΝΑΖΗΤΗΣΗ ΜΕ ΕΥΡΕΤΗΡΙΟ")
print("-" * 50)

# get(i) - παίρνει τον χαρακτήρα στη θέση i (σαν λίστα)
print("get(0) (πρώτος χαρακτήρας) από 'Name' (strip):")
print(df['Name'].str.strip().str.get(0))
print()

# find(sub) - θέση πρώτης εμφάνισης (ή -1)
print("find('a') στην 'Name' (πεζά):")
print(df['Name'].str.lower().str.find('a'))
print()

# rfind(sub) - θέση τελευταίας εμφάνισης
print("rfind('a') στην 'Name' (πεζά):")
print(df['Name'].str.lower().str.rfind('a'))
```

```
print()

# index(sub) - ίδιο με find, αλλά δίνει σφάλμα αν δεν βρεθεί (πρέπει να το χειριστούμε)
# Θα το εφαρμόσουμε σε μια στήλη που σίγουρα περιέχει το 'a' για αποφυγή σφάλματος.
print("index('a') στην 'Name' (πεζά) - μόνο όπου υπάρχει:")
try:
    print(df['Name'].str.lower().str.index('a'))
except ValueError as e:
    print("Σφάλμα ValueError λόγω έλλειψης του 'a' σε κάποιες γραμμές - χρησιμοποιούμε find() για ασφάλεια.")
print()

# rindex(sub) - αντίστοιχο rfind
# slice(start, stop) - απόκομμα
print("slice(0, 3) στην 'Code' (πρώτοι 3 χαρακτήρες):")
print(df['Code'].str.slice(0, 3))
print()

# slice_replace() - αντικατάσταση τμήματος
print("slice_replace(0, 3, 'XXX') στην 'Code':")
print(df['Code'].str.slice_replace(0, 3, 'XXX'))
print("\n" + "="*100 + "\n")

print("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
```

Κ. 3.4.14: Παράδειγμα για τις Μεθόδους επεξεργασίας αλφαριθμητικών (Κειμένου) για Πλαίσια Δεδομένων

3.4.14. Χρονικές Σειρές (dt accessor)

Μέθοδος	Περιγραφή	Επιστροφή
Μέθοδος/Ιδιότητα	Περιγραφή	Παράδειγμα
Ιδιότητες (Properties) - Εξαγωγή Συστατικών Ημερομηνίας/Ώρας		
.dt.year	Επιστρέφει το έτος ως ακέραιο αριθμό.	df['Ημερομηνία'].dt.year
.dt.month	Επιστρέφει τον μήνα (1-12).	df['Ημερομηνία'].dt.month
.dt.day	Επιστρέφει την ημέρα του μήνα (1-31).	df['Ημερομηνία'].dt.day
.dt.hour	Επιστρέφει την ώρα (0-23).	df['Ημερομηνία'].dt.hour
.dt.minute	Επιστρέφει τα λεπτά (0-59).	df['Ημερομηνία'].dt.minute
.dt.second	Επιστρέφει τα δευτερόλεπτα (0-59).	df['Ημερομηνία'].dt.second
.dt.microsecond	Επιστρέφει τα μικροδευτερόλεπτα.	df['Ημερομηνία'].dt.microsecond

Μέθοδος	Περιγραφή	Επιστροφή
.dt.nanosecond	Επιστρέφει τα νανοδευτερόλεπτα.	df['Ημερομηνία'].dt.nanosecond
.dt.weekday (ή .dt.dayofweek)	Επιστρέφει την ημέρα της εβδομάδας ως αριθμό (Δευτέρα=0, Κυριακή=6).	df['Ημερομηνία'].dt.weekday
.dt.dayofyear	Επιστρέφει την αύξουσα αριθμό της ημέρας μέσα στο έτος (1-366).	df['Ημερομηνία'].dt.dayofyear
.dt.quarter	Επιστρέφει το τρίμηνο του έτους (1-4).	df['Ημερομηνία'].dt.quarter
.dt.days_in_month	Επιστρέφει το πλήθος των ημερών του μήνα (π.χ. 28, 30, 31).	df['Ημερομηνία'].dt.days_in_month
.dt.is_leap_year	Επιστρέφει True αν το έτος είναι δίσεκτο, αλλιώς False.	df['Ημερομηνία'].dt.is_leap_year
Μέθοδοι (Functions) - Μετατροπές & Λογικές Ερωτήσεις		
.dt.normalize()	Μετατρέπει την ώρα σε "00:00:00" (μεσάνυχτα), κρατώντας μόνο την ημερομηνία. Επιστρέφει datetime64 τύπο.	df['Ημερομηνία'].dt.normalize()
.dt.date	Επιστρέφει μόνο το μέρος της ημερομηνίας (ημέρα-μήνας-έτος) ως object (όχι datetime).	df['Ημερομηνία'].dt.date
.dt.time	Επιστρέφει μόνο το μέρος της ώρας.	df['Ημερομηνία'].dt.time
.dt.day_name()	Επιστρέφει το όνομα της ημέρας (π.χ. "Monday", "Δευτέρα" ανάλογα με τα locale).	df['Ημερομηνία'].dt.day_name()
.dt.month_name()	Επιστρέφει το όνομα του μήνα.	df['Ημερομηνία'].dt.month_name()
.dt.is_month_start	Ελέγχει αν η ημερομηνία είναι η πρώτη ημέρα του μήνα (True/False).	df['Ημερομηνία'].dt.is_month_start
.dt.is_month_end	Ελέγχει αν η ημερομηνία είναι η τελευταία ημέρα του μήνα.	df['Ημερομηνία'].dt.is_month_end
.dt.is_year_start	Ελέγχει αν η ημερομηνία είναι η πρώτη ημέρα του χρόνου (1η Ιανουαρίου).	df['Ημερομηνία'].dt.is_year_start
.dt.is_year_end	Ελέγχει αν η ημερομηνία είναι η τελευταία ημέρα του χρόνου (31η Δεκεμβρίου).	df['Ημερομηνία'].dt.is_year_end
.dt.isocalendar()	Επιστρέφει ένα tuple με (ISO έτος, ISO εβδομάδα, ISO ημέρα). Ιδανικό για εβδομάδες.	df['Ημερομηνία'].dt.isocalendar().week

Μέθοδος	Περιγραφή	Επιστροφή
.dt.strftime(format)	Μετατρέπει την ημερομηνία σε string σύμφωνα με την μορφοποίηση format (π.χ. %d/%m/%Y).	df['Ημερομηνία'].dt.strftime('%d/%m/%Y')
.dt.floor(freq)	Στρογγυλοποιεί προς τα κάτω την ημερομηνία/ώρα στην καθορισμένη συχνότητα (freq).	df['Ημερομηνία'].dt.floor('H') (στρογγυλοποίηση στην πλησιέστερη ώρα)
.dt.ceil(freq)	Στρογγυλοποιεί προς τα πάνω την ημερομηνία/ώρα.	df['Ημερομηνία'].dt.ceil('D') (στρογγυλοποίηση στην επόμενη ημέρα)
.dt.round(freq)	Στρογγυλοποιεί στην πλησιέστερη συχνότητα.	df['Ημερομηνία'].dt.round('10min')
.dt.to_period(freq)	Μετατρέπει το datetime σε περίοδο (Period) βάσει συχνότητας (π.χ. 'M' για μήνα, 'D' για ημέρα).	df['Ημερομηνία'].dt.to_period('M')
.dt.tz_localize(None)	Αφαιρεί πληροφορία ζώνης ώρας (timezone) από ένα datetime.	df['Ημερομηνία'].dt.tz_localize(None)
.dt.tz_convert(tz)	Μετατρέπει ένα datetime-aware Series σε άλλη ζώνη ώρας.	df['Ημερομηνία'].dt.tz_convert('Europe/Athens')

Πίνακας 3.4.15: Μέθοδοι επεξεργασίας ημερομηνιών και ώρας για τα Πλαίσια Δεδομένων

Όπως οι μέθοδοι κειμένου χρησιμοποιούν το .str, έτσι και οι ημερομηνίες χρησιμοποιούν τον **.dt accessor**. Είναι το "μαγικό ραβδί" που ξεκλειδώνει πληροφορίες κρυμμένες μέσα σε μια σφραγίδα χρόνου (timestamp).

1. Εξαγωγή Συστατικών (The Anatomy of Time)

Όταν έχουμε μια στήλη datetime64, μπορούμε να την τεμαχίσουμε σε όποιο επίπεδο θέλουμε.

- **year, month, day**: Τα βασικά για να φιλτράρουμε δεδομένα ανά περίοδο.
- **weekday & day_name()**: Ιδανικά για να βρούμε αν μια αγορά έγινε Σαββατοκύριακο ή για να δούμε αν οι πωλήσεις αυξάνονται τις Παρασκευές.
- **quarter**: Απαραίτητο για οικονομικές αναφορές (Q1, Q2, κ.λπ.).

2. Λογικοί Έλεγχοι (Business Logic)

Οι ιδιότητες όπως η .is_month_end ή .is_year_start είναι εξαιρετικά χρήσιμες για τον υπολογισμό μηνιαίων μπόνους, κλεισίματα βιβλίων ή την παρακολούθηση της κίνησης σε συγκεκριμένες ημερομηνίες-ορόσημα. Αντί να γράφεις περίπλοκες συναρτήσεις, η Pandas σου απαντά με ένα απλό True/False.

3. Μορφοποίηση & Στρογγυλοποίηση

- **strftime()**: Η πιο σημαντική μέθοδος για την παρουσίαση των δεδομένων. Σου επιτρέπει να μετατρέψεις το 2024-05-15 σε "15 May, 2024" ή όποια άλλη μορφή επιθυμείς.
- **normalize(), floor(), ceil()**: Σκέψου τις σαν το round() των ημερομηνιών. Αν έχεις δεδομένα ανά δευτερόλεπτο αλλά θες να τα ομαδοποιήσεις ανά ώρα, η floor('H') θα τα "κουρέψει" όλα στην αρχή της ώρας.

4. Χειρισμός Ζωνών Ώρας (Timezones)

Αν δουλεύεις με παγκόσμια δεδομένα (π.χ. logs από servers), οι μέθοδοι tz_localize και tz_convert είναι η άμυνά σου απέναντι στο χάος των διαφορετικών ωρών. Σου επιτρέπουν να φέρεις όλα τα δεδομένα σε ένα κοινό σημείο αναφοράς (π.χ. UTC) ή να τα δεις στην τοπική ώρα της Αθήνας.

Πριν χρησιμοποιήσεις οποιαδήποτε από αυτές τις μεθόδους, βεβαιώσου ότι η στήλη σου είναι όντως τύπου ημερομηνίας. Αν η df.info() σου δείχνει object, τρέξε πρώτα: df['Ημερομηνία'] = pd.to_datetime(df['Ημερομηνία'])

```
import pandas as pd
import numpy as np

# ----------------------------------------------------------------------
# Δημιουργία δείγματος DataFrame με ημερομηνίες/ώρες
# ----------------------------------------------------------------------
print("ΔΗΜΙΟΥΡΓΙΑ ΔΕΙΓΜΑΤΟΣ DataFrame")
print("-" * 70)

# Δημιουργία μιας σειράς ημερομηνιών
date_rng = pd.date_range(start='2024-01-15 08:30:00', periods=10, freq='D')
# Προσθήκη μερικών ωρών και διαφορετικών ημερομηνιών για ποικιλία
dates = [
    '2024-02-28 14:45:30',
    '2024-03-01 00:00:00',
    '2024-12-31 23:59:59',
    '2023-01-01 12:00:00',
    '2024-07-04 18:20:10',
    '2024-11-30 06:15:00',
    '2024-02-29 10:10:10',  # δίσεκτο έτος
]

# Συνδυάζουμε σε ένα DataFrame
all_dates = list(date_rng) + [pd.Timestamp(d) for d in dates]
df = pd.DataFrame({'datetime': all_dates})
print("Αρχικό DataFrame με 17 ημερομηνίες:")
print(df)
print("\nΠληροφορίες:")
print(df.dtypes)
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 1. ΙΔΙΟΤΗΤΕΣ (PROPERTIES) - Εξαγωγή συστατικών ημερομηνίας/ώρας
# ----------------------------------------------------------------------
print("1. ΙΔΙΟΤΗΤΕΣ (PROPERTIES)")
print("-" * 60)

# .dt.year
print("df['datetime'].dt.year:")
print(df['datetime'].dt.year)
print()

# .dt.month
print("df['datetime'].dt.month:")
print(df['datetime'].dt.month)
print()

# .dt.day
print("df['datetime'].dt.day:")
print(df['datetime'].dt.day)
```

```python
print()

# .dt.hour
print("df['datetime'].dt.hour:")
print(df['datetime'].dt.hour)
print()

# .dt.minute
print("df['datetime'].dt.minute:")
print(df['datetime'].dt.minute)
print()

# .dt.second
print("df['datetime'].dt.second:")
print(df['datetime'].dt.second)
print()

# .dt.microsecond (τα περισσότερα είναι 0, αλλά υπάρχουν)
print("df['datetime'].dt.microsecond:")
print(df['datetime'].dt.microsecond)
print()

# .dt.nanosecond (συνήθως 0)
print("df['datetime'].dt.nanosecond:")
print(df['datetime'].dt.nanosecond)
print()

# .dt.weekday (ή dayofweek) - Δευτέρα=0, Κυριακή=6
print("df['datetime'].dt.weekday:")
print(df['datetime'].dt.weekday)
print()

# .dt.dayofweek (ίδιο)
print("df['datetime'].dt.dayofweek (ίδιο):")
print(df['datetime'].dt.dayofweek)
print()

# .dt.dayofyear
print("df['datetime'].dt.dayofyear:")
print(df['datetime'].dt.dayofyear)
print()

# .dt.quarter
print("df['datetime'].dt.quarter:")
print(df['datetime'].dt.quarter)
print()

# .dt.days_in_month
print("df['datetime'].dt.days_in_month:")
print(df['datetime'].dt.days_in_month)
print()

# .dt.is_leap_year
print("df['datetime'].dt.is_leap_year:")
print(df['datetime'].dt.is_leap_year)
print()

print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 2. ΜΕΘΟΔΟΙ (FUNCTIONS) - Μετατροπές & Λογικές Ερωτήσεις
```

```python
# ----------------------------------------------------------------------
print("2. ΜΕΘΟΔΟΙ (FUNCTIONS)")
print("-" * 60)

# .dt.normalize() - Μηδενίζει την ώρα (γίνεται 00:00:00)
print("df['datetime'].dt.normalize():")
print(df['datetime'].dt.normalize())
print()

# .dt.date - Επιστρέφει μόνο την ημερομηνία ως object (datetime.date)
print("df['datetime'].dt.date:")
print(df['datetime'].dt.date)
print("Τύπος:", df['datetime'].dt.date.apply(type).iloc[0])
print()

# .dt.time - Επιστρέφει μόνο την ώρα ως object (datetime.time)
print("df['datetime'].dt.time:")
print(df['datetime'].dt.time)
print()

# .dt.day_name() - Όνομα ημέρας (στα Αγγλικά από προεπιλογή, αλλά μπορεί να
αλλάξει locale)
print("df['datetime'].dt.day_name():")
print(df['datetime'].dt.day_name())
print()

# .dt.month_name() - Όνομα μήνα
print("df['datetime'].dt.month_name():")
print(df['datetime'].dt.month_name())
print()

# .dt.is_month_start
print("df['datetime'].dt.is_month_start:")
print(df['datetime'].dt.is_month_start)
print()

# .dt.is_month_end
print("df['datetime'].dt.is_month_end:")
print(df['datetime'].dt.is_month_end)
print()

# .dt.is_year_start
print("df['datetime'].dt.is_year_start:")
print(df['datetime'].dt.is_year_start)
print()

# .dt.is_year_end
print("df['datetime'].dt.is_year_end:")
print(df['datetime'].dt.is_year_end)
print()

# .dt.isocalendar() - Επιστρέφει DataFrame με έτος, εβδομάδα, ημέρα κατά
ISO
print("df['datetime'].dt.isocalendar():")
isocal = df['datetime'].dt.isocalendar()
print(isocal)
print("Πρόσβαση στην εβδομάδα:", isocal['week'])
print()

# .dt.strftime(format) - Μορφοποίηση ως string
print("df['datetime'].dt.strftime('%d/%m/%Y %H:%M'):")
```

```
print(df['datetime'].dt.strftime('%d/%m/%Y %H:%M'))
print()

# .dt.floor(freq) - Στρογγυλοποίηση προς τα κάτω (π.χ. στην ώρα)
print("df['datetime'].dt.floor('H'):")
print(df['datetime'].dt.floor('H'))
print()

# .dt.ceil(freq) - Στρογγυλοποίηση προς τα πάνω (π.χ. στην επόμενη ημέρα)
print("df['datetime'].dt.ceil('D'):")
print(df['datetime'].dt.ceil('D'))
print()

# .dt.round(freq) - Στρογγυλοποίηση στην πλησιέστερη συχνότητα (π.χ. 10
λεπτά)
print("df['datetime'].dt.round('10min'):")
print(df['datetime'].dt.round('10min'))
print()

# .dt.to_period(freq) - Μετατροπή σε περίοδο (π.χ. μήνα)
print("df['datetime'].dt.to_period('M'):")
print(df['datetime'].dt.to_period('M'))
print()

# ----------------------------------------------------------------------
# 3. ΠΑΡΑΔΕΙΓΜΑΤΑ ΜΕ TIMEZONE (για tz_localize και tz_convert)
# ----------------------------------------------------------------------
print("3. ΜΕΘΟΔΟΙ TIMEZONE")
print("-" * 60)

# Δημιουργία DataFrame με timezone-naive datetime
df_tz = df.copy()
print("Αρχικό (χωρίς timezone):")
print(df_tz['datetime'].head())
print()

# .dt.tz_localize - Προσθήκη timezone (π.χ. UTC)
df_tz['datetime_utc'] = df_tz['datetime'].dt.tz_localize('UTC')
print("Μετά από tz_localize('UTC'):")
print(df_tz['datetime_utc'].head())
print()

# .dt.tz_convert - Μετατροπή σε άλλη ζώνη (π.χ. Ελλάδα)
df_tz['datetime_athens'] =
df_tz['datetime_utc'].dt.tz_convert('Europe/Athens')
print("Μετά από tz_convert('Europe/Athens'):")
print(df_tz['datetime_athens'].head())
print()

# Αφαίρεση timezone με tz_localize(None)
df_tz['datetime_no_tz'] = df_tz['datetime_utc'].dt.tz_localize(None)
print("Μετά από tz_localize(None) (αφαίρεση timezone):")
print(df_tz['datetime_no_tz'].head())
print()

print("\n" + "="*100 + "\n")
print("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
```

Κ. 3.4.15: Παράδειγμα για τις Μεθόδους επεξεργασίας ημερομηνιών και ώρας των Πλαισίων Δεδομένων

3.4.15. Ομαδοποίηση (GroupBy)

Μέθοδος	Περιγραφή	Εφαρμογή
groupby()	Δημιουργεί ένα αντικείμενο GroupBy, χωρίζοντας τα δεδομένα σε ομάδες με βάση μία ή περισσότερες στήλες. Είναι η βάση για όλες τις επόμενες λειτουργίες ομαδοποίησης .	df.groupby('category')
agg() **ή** aggregate()	Εφαρμόζει μία ή περισσότερες συναρτήσεις συνάθροισης σε κάθε ομάδα, επιστρέφοντας **μία γραμμή ανά ομάδα**. Ιδανικό για υπολογισμό συνοπτικών στατιστικών (π.χ., μέσος όρος, άθροισμα) .	df.groupby('team').agg(avg_score=('score', 'mean'), total_score=('score', 'sum'))
transform()	Εφαρμόζει μια συνάρτηση σε κάθε ομάδα και επιστρέφει ένα αποτέλεσμα με το **ίδιο μήκος με το αρχικό DataFrame**. Χρησιμοποιείται για τη δημιουργία χαρακτηριστικών ανά γραμμή, όπως η κανονικοποίηση εντός της ομάδας ή η συμπλήρωση κενών τιμών .	df['score_pct'] = df['score'] / df.groupby('team')['score'].transform('sum')
apply()	Εφαρμόζει μια **προσαρμοσμένη συνάρτηση** σε κάθε ομάδα. Είναι πιο ευέλικτο αλλά και πιο αργό από τα agg και transform. Χρησιμοποιείται μόνο όταν αυτά δεν επαρκούν .	df.groupby('team').apply(lambda g: g.nlargest(2, 'score'))
filter()	Επιλέγει ή απορρίπτει ολόκληρες ομάδες με βάση μια συνθήκη που επιστρέφει True ή False για κάθε ομάδα .	df.groupby('team').filter(lambda g: len(g) >= 3)
size()	Επιστρέφει το μέγεθος (πλήθος γραμμών) κάθε ομάδας, συμπεριλαμβάνοντας και τις τιμές NaN .	df.groupby('team').size()
first() **/** last()	Επιστρέφει την πρώτη ή την τελευταία μη-κενή τιμή κάθε ομάδας .	df.groupby('team')['score'].first()
nth()	Επιστρέφει την ν-οστή γραμμή κάθε ομάδας .	df.groupby('team').nth(1) # Επιστρέφει τη 2η γραμμή κάθε ομάδας
cumcount()	Αριθμεί τις γραμμές κάθε ομάδας ξεκινώντας από το 0 .	df['obs_number'] = df.groupby('team').cumcount()
rank()	Υπολογίζει τη σειρά (rank) των τιμών μέσα σε κάθε ομάδα .	df['rank_in_team'] = df.groupby('team')['score'].rank(ascending=False)

Πίνακας 3.4.16: Μέθοδοι για ομαδοποίηση (groupby) Πλαισίων Δεδομένων

Η ομαδοποίηση (groupby) είναι η διαδικασία όπου χωρίζουμε ένα μεγάλο σύνολο δεδομένων σε μικρότερα κομμάτια για να εξάγουμε ουσιαστικά συμπεράσματα ανά κατηγορία.

1. Η Φιλοσοφία: Split-Apply-Combine

Όταν καλείς την groupby(), η Pandas ακολουθεί τρία βήματα:

1. **Split**: Χωρίζει το DataFrame σε ομάδες βάσει των τιμών μιας στήλης (π.χ. "Ομάδα Α", "Ομάδα Β").
2. **Apply**: Εφαρμόζει μια πράξη (άθροισμα, μέσο όρο, φίλτρο) σε κάθε ομάδα ξεχωριστά.

3. **Combine**: Ενώνει τα αποτελέσματα σε ένα νέο, συνοπτικό DataFrame.

2. Οι Τρεις Πυλώνες: Agg, Transform, Filter

Αυτές οι τρεις μέθοδοι καθορίζουν τι είδους αποτέλεσμα θα πάρεις:

- **agg() (Aggregation)**: Μειώνει τα δεδομένα. Αν η ομάδα έχει 100 γραμμές, η agg θα επιστρέψει **μία** τιμή (π.χ. το άθροισμα). Είναι η πιο συνηθισμένη μέθοδος για reports.
- **transform()**: Διατηρεί το αρχικό σχήμα. Αν η ομάδα έχει 100 γραμμές, η transform θα επιστρέψει **100** τιμές. Είναι ιδανική για να συγκρίνεις μια τιμή με τον μέσο όρο της ομάδας της (π.χ. "Πόσο απέχει ο μισθός αυτού του υπαλλήλου από τον μέσο όρο του τμήματός του;").
- **filter()**: Λειτουργεί σαν "κόσκινο" για ομάδες. Δεν φιλτράρει γραμμές, αλλά **ολόκληρες κατηγορίες**.

Παράδειγμα: "Δείξε μου δεδομένα μόνο για τις ομάδες που έχουν παίξει πάνω από 5 παιχνίδια". Αν μια ομάδα έχει 4, εξαφανίζεται όλη από το αποτέλεσμα.

3. Πλοήγηση μέσα στις Ομάδες

- **first() / last()**: Χρήσιμα σε χρονοσειρές για να δεις την πρώτη και την τελευταία καταγραφή μιας περιόδου.
- **nth()**: Σου δίνει πρόσβαση σε συγκεκριμένη σειρά. Πολύ χρήσιμο αν θες π.χ. να βρεις τον "δεύτερο καλύτερο" κάθε κατηγορίας.
- **cumcount()**: Δημιουργεί ένα "id" μέσα στην ομάδα. Αν έχεις πελάτες και τις επισκέψεις τους, το cumcount θα αριθμήσει την 1η, 2η, 3η επίσκεψη για κάθε πελάτη ξεχωριστά.

4. Κατάταξη εντός Ομάδας (rank)

Η rank() είναι πανίσχυρη για διαγωνισμούς ή πωλήσεις. Μπορείς να βρεις τη θέση ενός πωλητή όχι στο σύνολο της εταιρείας, αλλά **μέσα στην περιοχή του**. Αν δύο έχουν την ίδια τιμή, η rank διαθέτει παραμέτρους (όπως method='dense') για να αποφασίσεις πώς θα χειριστείς τις ισοπαλίες.

```
import pandas as pd
import numpy as np

# ---------------------------------------------------------------------
# Δημιουργία δείγματος DataFrame με δεδομένα πωλήσεων ανά ομάδα
# ---------------------------------------------------------------------
print("ΔΗΜΙΟΥΡΓΙΑ ΔΕΙΓΜΑΤΟΣ DataFrame")
print("-" * 70)

np.random.seed(42)  # για αναπαραγωγιμότητα

data = {
    'Team': ['A', 'A', 'B', 'B', 'B', 'C', 'C', 'C', 'C', 'D'],
    'Player': ['Anna', 'George', 'Maria', 'Nikos', 'Eleni', 'Dimitris',
'Sofia', 'John', 'Alice', 'Bob'],
    'Score': [85, 92, 78, 88, 95, 67, 72, 90, 84, 79],
    'Age': [25, 30, 28, 35, 22, 27, 24, 40, 31, 29],
    'Year': [2023, 2023, 2023, 2023, 2023, 2024, 2024, 2024, 2024, 2024]
}
df = pd.DataFrame(data)
print("Αρχικό DataFrame:")
print(df)
print("\n" + "="*100 + "\n")
```

```python
# ---------------------------------------------------------------------
# 1. groupby() - Δημιουργία ομάδων
# ---------------------------------------------------------------------
print("1. groupby() - Δημιουργία αντικειμένου GroupBy")
print("-" * 60)

grouped = df.groupby('Team')
print("Ομάδες που δημιουργήθηκαν:")
for name, group in grouped:
    print(f"Ομάδα {name}:")
    print(group)
    print()
print("\n" + "="*100 + "\n")

# ---------------------------------------------------------------------
# 2. agg() ή aggregate() - Συνάθροιση με συναρτήσεις
# ---------------------------------------------------------------------
print("2. agg() - Πολλαπλές συναρτήσεις συνάθροισης")
print("-" * 60)

# Μία συνάρτηση σε μία στήλη
agg1 = df.groupby('Team')['Score'].agg('mean')
print("Μέσος όρος Score ανά ομάδα:")
print(agg1)
print()

# Πολλές συναρτήσεις σε μία στήλη
agg2 = df.groupby('Team')['Score'].agg(['mean', 'max', 'min'])
print("Στατιστικά Score ανά ομάδα:")
print(agg2)
print()

# Διαφορετικές συναρτήσεις σε διαφορετικές στήλες
agg3 = df.groupby('Team').agg(
    avg_score=('Score', 'mean'),
    total_score=('Score', 'sum'),
    avg_age=('Age', 'mean'),
    max_age=('Age', 'max')
)
print("Προσαρμοσμένες συναθροίσεις:")
print(agg3)
print("\n" + "="*100 + "\n")

# ---------------------------------------------------------------------
# 3. transform() - Επιστροφή αποτελέσματος ίδιου μήκους
# ---------------------------------------------------------------------
print("3. transform() - Μετασχηματισμός εντός ομάδας")
print("-" * 60)

# Ποσοστό του score ως προς το άθροισμα της ομάδας
df['Score_pct'] = df.groupby('Team')['Score'].transform(lambda x: x /
x.sum() * 100)
print("Ποσοστό Score εντός ομάδας:")
print(df[['Team', 'Player', 'Score', 'Score_pct']])
print()

# Κανονικοποίηση (z-score) εντός ομάδας
df['Score_z'] = df.groupby('Team')['Score'].transform(lambda x: (x -
x.mean()) / x.std())
print("Z-score του Score εντός ομάδας:")
```

```
print(df[['Team', 'Player', 'Score', 'Score_z']])
print()

# Συμπλήρωση κενών (αν υπήρχαν) με τον μέσο όρο της ομάδας - εδώ δεν
υπάρχουν NaN, αλλά για επίδειξη
df_filled = df.copy()
df_filled.loc[2, 'Score'] = np.nan  # τεχνητό NaN
df_filled['Score_filled'] =
df_filled.groupby('Team')['Score'].transform(lambda x: x.fillna(x.mean()))
print("Συμπλήρωση NaN με τον μέσο όρο της ομάδας:")
print(df_filled[['Team', 'Player', 'Score', 'Score_filled']])
print("\n" + "="*100 + "\n")

# ------------------------------------------------------------------
# 4. apply() - Εφαρμογή custom συνάρτησης σε κάθε ομάδα
# ------------------------------------------------------------------
print("4. apply() - Προσαρμοσμένη συνάρτηση ανά ομάδα")
print("-" * 60)

# Επιστροφή των δύο κορυφαίων σκόρερ κάθε ομάδας
top2 = df.groupby('Team').apply(lambda g: g.nlargest(2, 'Score')[['Player',
'Score']])
print("Οι 2 κορυφαίοι σκόρερ ανά ομάδα:")
print(top2)
print()

# Εφαρμογή συνάρτησης που επιστρέφει DataFrame (π.χ. κανονικοποίηση)
def normalize(group):
    group['Score_norm'] = (group['Score'] - group['Score'].min()) /
(group['Score'].max() - group['Score'].min())
    return group

normalized = df.groupby('Team').apply(normalize)
print("Κανονικοποίηση min-max εντός ομάδας:")
print(normalized[['Team', 'Player', 'Score', 'Score_norm']])
print("\n" + "="*100 + "\n")

# ------------------------------------------------------------------
# 5. filter() - Φιλτράρισμα ομάδων
# ------------------------------------------------------------------
print("5. filter() - Κρατά ομάδες που ικανοποιούν συνθήκη")
print("-" * 60)

# Κρατάμε μόνο ομάδες με τουλάχιστον 3 μέλη
filtered = df.groupby('Team').filter(lambda g: len(g) >= 3)
print("Ομάδες με >=3 μέλη:")
print(filtered)
print()

# Κρατάμε ομάδες όπου το μέγιστο score είναι > 90
filtered_max = df.groupby('Team').filter(lambda g: g['Score'].max() > 90)
print("Ομάδες με max score > 90:")
print(filtered_max)
print("\n" + "="*100 + "\n")

# ------------------------------------------------------------------
# 6. size() - Μέγεθος κάθε ομάδας
# ------------------------------------------------------------------
print("6. size() - Πλήθος γραμμών ανά ομάδα")
print("-" * 60)
```

```
group_sizes = df.groupby('Team').size()
print("Μέγεθος ομάδων:")
print(group_sizes)
print()

# reset_index για πιο όμορφη παρουσίαση
group_sizes_df = group_sizes.reset_index(name='Count')
print("Ως DataFrame:")
print(group_sizes_df)
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 7. first(), last(), nth() - Πρώτη, τελευταία, ν-οστή γραμμή
# ----------------------------------------------------------------------
print("7. first(), last(), nth() - Επιλογή συγκεκριμένων γραμμών")
print("-" * 60)

# Πρώτη γραμμή κάθε ομάδας (βάσει σειράς εμφάνισης)
first = df.groupby('Team').first()
print("first():")
print(first[['Player', 'Score']])
print()

# Τελευταία γραμμή
last = df.groupby('Team').last()
print("last():")
print(last[['Player', 'Score']])
print()

# N-οστή γραμμή (π.χ. 1 = δεύτερη γραμμή, ξεκινά από 0)
nth = df.groupby('Team').nth(1)
print("nth(1) - δεύτερη γραμμή:")
print(nth[['Player', 'Score']])
print()

# nth με fallback (αν δεν υπάρχει, μπορεί να επιστρέψει NaN)
nth_fallback = df.groupby('Team').nth(2, default=pd.NA)
print("nth(2, default=pd.NA) - τρίτη γραμμή ή NA:")
print(nth_fallback[['Player', 'Score']])
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 8. cumcount() - Αρίθμηση γραμμών εντός ομάδας
# ----------------------------------------------------------------------
print("8. cumcount() - Αύξων αριθμός εντός ομάδας")
print("-" * 60)

df['Obs_in_team'] = df.groupby('Team').cumcount() + 1  # +1 για να ξεκινά
από 1
print("Αύξων αριθμός παίκτη στην ομάδα:")
print(df[['Team', 'Player', 'Obs_in_team']])
print()
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 9. rank() - Κατάταξη τιμών εντός ομάδας
# ----------------------------------------------------------------------
print("9. rank() - Σειρά κατάταξης (ranking)")
print("-" * 60)

# Κατάταξη βάσει Score (υψηλότερο = 1)
```

```
df['Rank_score'] = df.groupby('Team')['Score'].rank(ascending=False,
method='dense')
print("Κατάταξη score εντός ομάδας (1=κορυφαίος):")
print(df[['Team', 'Player', 'Score', 'Rank_score']])
print()

# Διάφορες μέθοδοι: 'min', 'max', 'first', 'dense'
df['Rank_method'] = df.groupby('Team')['Score'].rank(method='min',
ascending=False)
print("Κατάταξη με method='min':")
print(df[['Team', 'Player', 'Score', 'Rank_method']])
print("\n" + "="*100 + "\n")

print("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
```

Κ. 3.4.16: Παράδειγμα των μεθόδων για ομαδοποίηση (groupby) Πλαισίων Δεδομένων

3.4.16. Κινητοί Μέσοι (Rolling/EWM/Expanding)

Μέθοδος	Περιγραφή	Εφαρμογή
rolling()	Δημιουργεί ένα παράθυρο κυλιόμενου υπολογισμού σταθερού μεγέθους (π.χ., 3 γραμμές) ή σταθερού χρονικού διαστήματος (π.χ., "7D"). Στη συνέχεια, εφαρμόζεται μια συνάρτηση (π.χ., mean()) .	df['moving_avg'] = df['sales'].rolling(window=3).mean()
.agg() **πάνω σε rolling**	Εφαρμόζει πολλαπλές συναρτήσεις συνάθροισης στο ίδιο κυλιόμενο παράθυρο ταυτόχρονα .	df.rolling(window=3).agg(['sum', 'std', 'max'])
.apply() **πάνω σε rolling**	Εφαρμόζει μια προσαρμοσμένη συνάρτηση σε κάθε κυλιόμενο παράθυρο .	df['sales_range'] = df['sales'].rolling(4).apply(lambda x: x.max() - x.min(), raw=True)
expanding()	Δημιουργεί ένα **διευρυνόμενο** παράθυρο που ξεκινά από την αρχή των δεδομένων και μεγαλώνει μέχρι την τρέχουσα γραμμή. Χρήσιμο για υπολογισμό αθροιστικών στατιστικών .	df['cumulative_avg'] = df['sales'].expanding(min_periods=2).mean()
ewm()	Εφαρμόζει **εκθετική κινητή μείωση βάρους (Exponential Weighted Moving Average)**. Δίνει μεγαλύτερη βαρύτητα σε πιο πρόσφατες παρατηρήσεις .	df['ewm_avg'] = df['sales'].ewm(span=4, adjust=False).mean()
min_periods **(παράμετρος)**	Καθορίζει τον ελάχιστο αριθμό παρατηρήσεων με μη-NaN τιμές που πρέπει να έχει ένα παράθυρο για να εμφανιστεί αποτέλεσμα. Ελέγχει την εμφάνιση NaN στην αρχή της περιόδου .	df['moving_avg'] = df['sales'].rolling(window=5, min_periods=2).mean()
center **(παράμετρος)**	Αν center=True, τοποθετεί το αποτέλεσμα του παραθύρου στην **κεντρική** γραμμή (αντί για τη δεξιά, που είναι η προεπιλογή) .	df['moving_avg_centered'] = df['sales'].rolling(window=3, center=True).mean()

Πίνακας 3.4.17: Μέθοδοι για Κινητοί Μέσοι (Rolling/EWM/Expanding) των Πλαισίων Δεδομένων

Οι μέθοδοι **Window Operations** (Κυλιόμενα και Διευρυνόμενα Παράθυρα) είναι το "κλειδί" για την ανάλυση χρονοσειρών, καθώς μας επιτρέπουν να εξομαλύνουμε τον θόρυβο και να εντοπίσουμε τάσεις (trends) που δεν φαίνονται με μια απλή ματιά.

1. Rolling: Το "Κλασικό" Κινητό Παράθυρο

Η rolling() ορίζει ένα παράθυρο σταθερού μεγέθους που "γλιστράει" πάνω στα δεδομένα σας.

- **Window Size:** Αν ορίσετε window=7, κάθε αποτέλεσμα θα είναι ο υπολογισμός των τελευταίων 7 ημερών/εγγραφών.
- **Smoothing:** Είναι ο βασικός τρόπος για να δημιουργήσετε έναν **Κινητό Μέσο Όρο (Moving Average)**, ο οποίος εξαφανίζει τις απότομες καθημερινές διακυμάνσεις.
- **Center:** Από προεπιλογή, το αποτέλεσμα μπαίνει στην τελευταία γραμμή του παραθύρου. Με το center=True, το αποτέλεσμα μπαίνει στη μεσαία γραμμή, κάτι που είναι χρήσιμο για να ευθυγραμμιστεί ο μέσος όρος οπτικά με τα πραγματικά δεδομένα σε ένα γράφημα.

2. Expanding: Το "Ιστορικό" Παράθυρο

Σε αντίθεση με το rolling, η expanding() δεν αφήνει πίσω της τα παλιά δεδομένα. Το παράθυρο ξεκινά από την 1η γραμμή και μεγαλώνει συνεχώς.

- **Cumulative Insights:** Χρησιμοποιείται όταν θέλετε να βλέπετε πώς εξελίσσεται ένα στατιστικό μέγεθος (π.χ. η μέση απόδοση) από την αρχή των χρόνων μέχρι σήμερα.
- **Stability:** Είναι ιδανική για να δείτε αν ένας μέσος όρος "σταθεροποιείται" όσο μαζεύονται περισσότερα δεδομένα.

3. EWM: Εκθετική Βαρύτητα (Exponentially Weighted)

Η ewm() είναι η πιο εξελιγμένη μέθοδος. Αντί να δίνει την ίδια σημασία σε όλες τις ημέρες (όπως η rolling), δίνει **μεγαλύτερη βαρύτητα στις πιο πρόσφατες τιμές**.

- **Γιατί να τη χρησιμοποιήσετε;** Στα χρηματοοικονομικά, το τι έγινε σήμερα είναι συχνά πιο σημαντικό από το τι έγινε πριν 10 μέρες. Η ewm αντιδρά πολύ πιο γρήγορα στις αλλαγές της τάσης απ' ό,τι ένας απλός κινητός μέσος όρος.

4. Διαχείριση των NaN με την min_periods

Όταν ξεκινάει ένα rolling παράθυρο (π.χ. μεγέθους 10), οι πρώτες 9 γραμμές θα είναι αναγκαστικά NaN γιατί δεν υπάρχουν αρκετά δεδομένα.

- Με την **min_periods**, μπορείτε να πείτε στην Pandas: "Αν βρεις έστω και 2 τιμές, δώσε μου αποτέλεσμα, μην περιμένεις να μαζευτούν και οι 10". Αυτό βοηθάει να έχετε δεδομένα διαθέσιμα νωρίτερα στο dataset σας.

```
import pandas as pd
import numpy as np

# ---------------------------------------------------------------
# Δημιουργία δείγματος DataFrame με χρονοσειρά πωλήσεων
# ---------------------------------------------------------------
print("ΔΗΜΙΟΥΡΓΙΑ ΔΕΙΓΜΑΤΟΣ DataFrame")
print("-" * 70)

# Δημιουργία ημερομηνιών (καθημερινά)
```

```
dates = pd.date_range(start='2024-01-01', periods=15, freq='D')
np.random.seed(1)
sales = np.random.randint(100, 200, size=15).astype(float)

# Εισαγωγή μερικών NaN για επίδειξη min_periods
sales[2] = np.nan
sales[7] = np.nan
sales[12] = np.nan

df = pd.DataFrame({'date': dates, 'sales': sales})
print("Αρχικό DataFrame (πωλήσεις ανά ημέρα):")
print(df)
print("\n" + "="*100 + "\n")

# ---------------------------------------------------------------------
# 1. rolling() - Κυλιόμενο παράθυρο σταθερού μεγέθους
# ---------------------------------------------------------------------
print("1. rolling() - Κυλιόμενος μέσος όρος (window=3)")
print("-" * 60)

# Απλός κυλιόμενος μέσος όρος 3 ημερών
df['rolling_mean_3'] = df['sales'].rolling(window=3).mean()
print(df[['date', 'sales', 'rolling_mean_3']])
print()

# Πα настоящий: min_periods - ελάχιστος αριθμός μη-NaN τιμών για εμφάνιση
αποτελέσματος
df['rolling_mean_min2'] = df['sales'].rolling(window=3,
min_periods=2).mean()
print("με min_periods=2 (επιτρέπονται αποτελέσματα με 2 τιμές):")
print(df[['date', 'sales', 'rolling_mean_min2']])
print()

# center=True - τοποθέτηση αποτελέσματος στην κεντρική γραμμή
df['rolling_mean_center'] = df['sales'].rolling(window=3,
center=True).mean()
print("με center=True (το αποτέλεσμα τοποθετείται στην κεντρική ημέρα του
παραθύρου):")
print(df[['date', 'sales', 'rolling_mean_center']])
print()

# Κυλιόμενη τυπική απόκλιση
df['rolling_std_3'] = df['sales'].rolling(window=3).std()
print("Κυλιόμενη τυπική απόκλιση 3 ημερών:")
print(df[['date', 'sales', 'rolling_std_3']])
print("\n" + "="*100 + "\n")

# ---------------------------------------------------------------------
# 2. .agg() σε rolling - Πολλαπλές συναρτήσεις ταυτόχρονα
# ---------------------------------------------------------------------
print("2. rolling().agg() - Πολλαπλές συναθροίσεις")
print("-" * 60)

rolling_agg = df['sales'].rolling(window=3).agg(['mean', 'std', 'sum',
'max'])
print("Πολλαπλές συναρτήσεις στο ίδιο παράθυρο:")
print(rolling_agg)
print()

# Μπορούμε να συνδυάσουμε με τα υπόλοιπα δεδομένα
df_agg = df.join(rolling_agg)
```

```
print("Με ενσωμάτωση στο αρχικό DataFrame:")
print(df_agg.head(10))
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 3. .apply() σε rolling - Προσαρμοσμένη συνάρτηση
# ----------------------------------------------------------------------
print("3. rolling().apply() - Προσαρμοσμένη συνάρτηση")
print("-" * 60)

# Συνάρτηση που υπολογίζει το εύρος (max-min) του παραθύρου
df['rolling_range'] = df['sales'].rolling(window=4).apply(lambda x: x.max()
- x.min(), raw=True)
print("Κυλιόμενο εύρος (max-min) με window=4:")
print(df[['date', 'sales', 'rolling_range']])
print()

# Συνάρτηση που υπολογίζει τον αριθμό των θετικών μεταβολών (δεν έχει νόημα
εδώ, απλό παράδειγμα)
def count_positives(series):
    return (series > 0).sum()

df['rolling_pos'] = df['sales'].rolling(window=3).apply(count_positives,
raw=True)
print("Πλήθος θετικών τιμών (όλες >0, εκτός NaN):")
print(df[['date', 'sales', 'rolling_pos']])
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 4. expanding() - Διευρυνόμενο παράθυρο (από την αρχή)
# ----------------------------------------------------------------------
print("4. expanding() - Αθροιστικά στατιστικά")
print("-" * 60)

# Αθροιστικός μέσος όρος
df['expanding_mean'] = df['sales'].expanding().mean()
print("Αθροιστικός μέσος όρος:")
print(df[['date', 'sales', 'expanding_mean']])
print()

# Αθροιστικό άθροισμα
df['expanding_sum'] = df['sales'].expanding().sum()
print("Αθροιστικό άθροισμα:")
print(df[['date', 'sales', 'expanding_sum']])
print()

# με min_periods (ελάχιστος αριθμός τιμών)
df['expanding_mean_min3'] = df['sales'].expanding(min_periods=3).mean()
print("Αθροιστικός μέσος με min_periods=3 (NaN έως ότου υπάρχουν 3
τιμές):")
print(df[['date', 'sales', 'expanding_mean_min3']])
print()

# Εφαρμογή πολλαπλών συναρτήσεων με agg
exp_agg = df['sales'].expanding().agg(['mean', 'sum', 'std'])
print("expanding().agg() - πολλαπλές συναρτήσεις:")
print(exp_agg)
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 5. ewm() - Εκθετικά κινητός μέσος (Exponential Weighted)
```

```
# ----------------------------------------------------------------------
print("5. ewm() - Εκθετικά κινητός μέσος")
print("-" * 60)

# Εκθετικός κινητός μέσος με span=4 (αντίστοιχο με window 4, αλλά με
εκθετική μείωση βαρών)
df['ewm_span4'] = df['sales'].ewm(span=4, adjust=False).mean()
print("EWM με span=4:")
print(df[['date', 'sales', 'ewm_span4']])
print()

# με alpha (παράγοντας εξομάλυνσης)
df['ewm_alpha03'] = df['sales'].ewm(alpha=0.3, adjust=False).mean()
print("EWM με alpha=0.3:")
print(df[['date', 'sales', 'ewm_alpha03']])
print()

# min_periods σε ewm (αντίστοιχα)
df['ewm_min2'] = df['sales'].ewm(span=4, min_periods=2,
adjust=False).mean()
print("EWM με min_periods=2:")
print(df[['date', 'sales', 'ewm_min2']])
print()

# ignore NaN; τα NaN επηρεάζουν τους υπολογισμούς. Μπορούμε να δούμε τη
διαφορά.
# Δημιουργούμε ένα αντίγραφο χωρίς NaN για σύγκριση.
df_no_nan = df.dropna(subset=['sales']).copy()
df_no_nan['ewm_clean'] = df_no_nan['sales'].ewm(span=4,
adjust=False).mean()
print("EWM χωρίς NaN (για σύγκριση):")
print(df_no_nan[['date', 'sales', 'ewm_clean']])
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 6. Παράδειγμα συνδυασμού rolling με groupby (προαιρετικό)
# ----------------------------------------------------------------------
print("6. rolling() με groupby (προαιρετικό παράδειγμα)")
print("-" * 60)

# Δημιουργούμε ένα DataFrame με δύο ομάδες
df2 = pd.DataFrame({
    'group': ['A']*10 + ['B']*10,
    'value': np.random.randint(50, 150, 20).astype(float)
})
df2['rolling_group'] =
df2.groupby('group')['value'].rolling(window=3).mean().reset_index(level=0,
drop=True)
print("Κυλιόμενος μέσος ανά ομάδα:")
print(df2.head(12))
print("\n" + "="*100 + "\n")

print("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
```

Κ. 3.4.17: Μέθοδοι για Κινητοί Μέσοι (Rolling/EWM/Expanding) των Πλαισίων Δεδομένων

3.4.17. Εισαγωγή Δεδομένων

Μέθοδος	Περιγραφή	Επιστροφή
pd.read_csv()	Διαβάζει δεδομένα από ένα αρχείο	import pandas as pd df = pd.read_csv('το_αρχείο_μου.csv')

	διαχωρισμένο με κόμματα (CSV) και τα φορτώνει σε ένα DataFrame.	
pd.read_excel()	Διαβάζει δεδομένα από ένα αρχείο Excel. Υποστηρίζει την επιλογή συγκεκριμένου φύλλου εργασίας.	df = pd.read_excel('το_αρχείο_μου.xlsx', sheet_name='Φύλλο1')
pd.read_json()	Φορτώνει δεδομένα από ένα αρχείο ή μια συμβολοσειρά μορφοποιημένη σε JSON.	df = pd.read_json('το_αρχείο_μου.json')
pd.read_sql()	Εκτελεί ένα ερώτημα SQL σε μια βάση δεδομένων και επιστρέφει τα αποτελέσματα ως DataFrame.	from sqlalchemy import create_engine engine = create_engine('mysql://user:pass@host/db') df = pd.read_sql('SELECT * FROM πίνακας', engine)
pd.read_html()	Διαβάζει όλους τους πίνακες HTML από μια ιστοσελίδα ή μια συμβολοσειρά και τους επιστρέφει ως λίστα από DataFrames.	λίστα_DF = pd.read_html('https://παράδειγμα.gr/σελίδα.html') πρώτος_πίνακας = λίστα_DF[0]
pd.read_clipboard()	Διαβάζει τα δεδομένα που βρίσκονται στο πρόχειρο του υπολογιστή σας (π.χ. από ένα αντίγραφο από Excel) και τα μετατρέπει σε DataFrame.	df = pd.read_clipboard()
pd.DataFrame()	Δημιουργεί ένα νέο DataFrame από διάφορες δομές Python, όπως λεξικά ή λίστες.	δεδομένα = {'Όνομα': ['Άννα', 'Νίκος'], 'Ηλικία': [25, 30]} df = pd.DataFrame(δεδομένα)
pd.json_normalize()	Μετατρέπει (απλοποιεί) ημιδομημένα δεδομένα JSON (π.χ. με εμφωλευμένα πεδία) σε ένα "επίπεδο" DataFrame.	import requests response = requests.get('https://api.example.com/data').json() df = pd.json_normalize(response['αποτελέσματα'])

Πίνακας 3.4.18: Μέθοδοι για εισαγωγή δεδομένων στα Πλαισίων Δεδομένων

Αυτές οι εντολές αποτελούν την «πύλη εισόδου» των δεδομένων σας στον κόσμο των Pandas. Κάθε μία είναι σχεδιασμένη για να μετατρέπει διαφορετικές πηγές πληροφοριών σε ένα ομοιόμορφο **DataFrame**, το οποίο είναι η βασική δομή ανάλυσης.

Ας τις αναλύσουμε μία προς μία:

1. pd.read_csv()

Είναι η πιο συνηθισμένη εντολή, καθώς τα αρχεία CSV (Comma Separated Values) είναι το στάνταρ στη μεταφορά δεδομένων.

- **Πώς λειτουργεί:** Διαβάζει το αρχείο γραμμή-γραμμή και χρησιμοποιεί το κόμμα (ή άλλο σύμβολο που θα ορίσετε, π.χ. ; με την παράμετρο sep=';') για να ξεχωρίσει τις στήλες.
- **Χρήσιμο tip:** Αν το αρχείο σας έχει ελληνικούς χαρακτήρες και δεν διαβάζεται σωστά, δοκιμάστε την παράμετρο encoding='utf-8' ή encoding='iso-8859-7'.

2. pd.read_excel()

Απαραίτητη για όσους εργάζονται με αρχεία γραφείου.

- **Πώς λειτουργεί:** Φορτώνει δεδομένα από αρχεία .xls ή .xlsx.
- **Δυνατότητες:** Μπορείτε να διαβάσετε μόνο ένα συγκεκριμένο φύλλο (sheet_name) ή ακόμα και συγκεκριμένες στήλες (usecols).

3. pd.read_json()

Το JSON είναι η κύρια μορφή δεδομένων στο διαδίκτυο και στα APIs.

- **Πώς λειτουργεί:** Μετατρέπει τα "κλειδιά" του JSON σε ονόματα στηλών και τις "τιμές" σε δεδομένα γραμμών.
- **Εφαρμογή:** Ιδανικό όταν παίρνετε δεδομένα από web services.

4. pd.read_sql()

Γεφυρώνει τον κόσμο των βάσεων δεδομένων με την Python.

- **Πώς λειτουργεί:** Χρειάζεται μια "σύνδεση" (engine) που δημιουργείται συνήθως με τη βιβλιοθήκη sqlalchemy. Του δίνετε μια εντολή SQL (π.χ. SELECT *) και η Pandas αναλαμβάνει να μετατρέψει το αποτέλεσμα του ερωτήματος σε πίνακα.
- **Πλεονέκτημα:** Δεν χρειάζεται να εξάγετε χειροκίνητα δεδομένα από τη βάση σε CSV.

5. pd.read_html()

Μία από τις πιο εντυπωσιακές εντολές για "web scraping".

- **Πώς λειτουργεί:** Σαρώνει τον κώδικα μιας ιστοσελίδας και εντοπίζει τα tags <table>. Επειδή μια σελίδα μπορεί να έχει πολλούς πίνακες, επιστρέφει **λίστα** από DataFrames.
- **Περιορισμός:** Λειτουργεί μόνο με στατικούς πίνακες που υπάρχουν στον HTML κώδικα της σελίδας.

6. pd.read_clipboard()

Ο πιο γρήγορος τρόπος για "πρόχειρη" ανάλυση.

- **Πώς λειτουργεί:** Αν έχετε κάνει αντιγραφή (Ctrl+C) μια περιοχή κελιών από το Excel ή έναν πίνακα από μια ιστοσελίδα, τρέχοντας αυτή την εντολή, η Pandas διαβάζει το πρόχειρο και το κάνει αυτόματα DataFrame.

7. pd.DataFrame()

Η εντολή για χειροκίνητη δημιουργία ή μετατροπή εσωτερικών δομών της Python.

- **Πώς λειτουργεί:** Παίρνει δομές όπως Λεξικά (Dictionaries) ή Λίστες.
- **Παράδειγμα:** Ένα λεξικό όπου τα κλειδιά είναι οι τίτλοι των στηλών και οι λίστες είναι τα περιεχόμενα, γίνεται αμέσως πίνακας.

8. pd.json_normalize()

Ο "εξομαλυντής" για σύνθετα δεδομένα.

- **Πώς λειτουργεί:** Πολλά αρχεία JSON είναι "εμφωλευμένα" (nested), δηλαδή μια τιμή μπορεί να περιέχει ένα άλλο λεξικό μέσα της. Η json_normalize "ανοίγει" αυτά τα επίπεδα και δημιουργεί έναν επίπεδο (flat) πίνακα.
- **Γιατί είναι χρήσιμη:** Μετατρέπει δομές όπως {'πελάτης': {'όνομα': 'Άννα', 'πόλη': 'Αθήνα'}} σε δύο στήλες: πελάτης.όνομα και πελάτης.πόλη.

```
import pandas as pd
import numpy as np
import json
import tempfile
import os
from sqlalchemy import create_engine
import requests  # Θα χρησιμοποιηθεί μόνο για επίδειξη (προαιρετικά)

# ---------------------------------------------------------------------
# 1. pd.DataFrame() - Δημιουργία DataFrame από λεξικό ή λίστα
# ---------------------------------------------------------------------
print("1. pd.DataFrame() - Δημιουργία από λεξικό")
print("-" * 60)
data_dict = {
    'Name': ['Anna', 'George', 'Maria'],
    'Age': [25, 30, 28],
    'City': ['Athens', 'Thessaloniki', 'Patras']
}
df_from_dict = pd.DataFrame(data_dict)
print("Δημιουργήθηκε DataFrame:")
print(df_from_dict)
print("\n" + "="*80 + "\n")

# ---------------------------------------------------------------------
# 2. pd.read_csv() - Ανάγνωση από αρχείο CSV
# ---------------------------------------------------------------------
print("2. pd.read_csv() - Ανάγνωση από CSV")
print("-" * 60)
# Δημιουργία προσωρινού CSV αρχείου
csv_content = """Name,Age,City
Eleni,22,Volos
Dimitris,27,Chania
Sofia,24,Larisa
"""
with tempfile.NamedTemporaryFile(mode='w', suffix='.csv', delete=False,
encoding='utf-8') as tmp_csv:
```

```
    tmp_csv.write(csv_content)
    tmp_csv_path = tmp_csv.name

df_csv = pd.read_csv(tmp_csv_path)
print("Δεδομένα από το προσωρινό CSV:")
print(df_csv)
os.unlink(tmp_csv_path)  # Διαγραφή προσωρινού αρχείου
print("\n" + "="*80 + "\n")

# ------------------------------------------------------------------
# 3. pd.read_excel() - Ανάγνωση από Excel
# ------------------------------------------------------------------
print("3. pd.read_excel() - Ανάγνωση από Excel")
print("-" * 60)
try:
    # Απαιτείται openpyxl για εγγραφή/ανάγνωση Excel
    with tempfile.NamedTemporaryFile(suffix='.xlsx', delete=False) as
tmp_xlsx:
        tmp_xlsx_path = tmp_xlsx.name
    # Αποθήκευση του df_from_dict σε Excel
    df_from_dict.to_excel(tmp_xlsx_path, index=False, engine='openpyxl')
    # Ανάγνωση από το Excel
    df_excel = pd.read_excel(tmp_xlsx_path, engine='openpyxl')
    print("Δεδομένα από το προσωρινό Excel:")
    print(df_excel)
    os.unlink(tmp_xlsx_path)
except ImportError:
    print("Η βιβλιοθήκη openpyxl δεν είναι εγκατεστημένη. Παραλείπεται το
παράδειγμα Excel.")
    print("Εγκαταστήστε την με: pip install openpyxl")
print("\n" + "="*80 + "\n")

# ------------------------------------------------------------------
# 4. pd.read_json() - Ανάγνωση από JSON
# ------------------------------------------------------------------
print("4. pd.read_json() - Ανάγνωση από JSON")
print("-" * 60)
json_data = """[
    {"Name": "Nikos", "Age": 35, "City": "Heraklion"},
    {"Name": "Anna", "Age": 25, "City": "Athens"},
    {"Name": "George", "Age": 30, "City": "Thessaloniki"}
]"""
with tempfile.NamedTemporaryFile(mode='w', suffix='.json', delete=False,
encoding='utf-8') as tmp_json:
    tmp_json.write(json_data)
    tmp_json_path = tmp_json.name

df_json = pd.read_json(tmp_json_path)
print("Δεδομένα από το προσωρινό JSON:")
print(df_json)
os.unlink(tmp_json_path)
print("\n" + "="*80 + "\n")

# ------------------------------------------------------------------
# 5. pd.read_sql() - Ανάγνωση από SQL βάση δεδομένων
# ------------------------------------------------------------------
print("5. pd.read_sql() - Ανάγνωση από SQL (SQLite in-memory)")
print("-" * 60)
# Δημιουργία in-memory SQLite βάσης
engine = create_engine('sqlite:///:memory:')
# Αποθήκευση του DataFrame σε πίνακα SQL
```

```
df_from_dict.to_sql('employees', engine, index=False, if_exists='replace')
# Ανάγνωση με SQL query
df_sql = pd.read_sql('SELECT * FROM employees WHERE Age > 25', engine)
print("Αποτέλεσμα ερωτήματος SQL:")
print(df_sql)
print("\n" + "="*80 + "\n")

# ----------------------------------------------------------------------
# 6. pd.read_html() - Ανάγνωση από HTML πίνακες
# ----------------------------------------------------------------------
print("6. pd.read_html() - Ανάγνωση πινάκων από HTML")
print("-" * 60)
html_string = """
<html>
<body>
<table>
    <tr><th>Name</th><th>Age</th></tr>
    <tr><td>Alice</td><td>28</td></tr>
    <tr><td>Bob</td><td>32</td></tr>
</table>
</body>
</html>
"""
with tempfile.NamedTemporaryFile(mode='w', suffix='.html', delete=False,
encoding='utf-8') as tmp_html:
    tmp_html.write(html_string)
    tmp_html_path = tmp_html.name

df_list = pd.read_html(tmp_html_path)  # Επιστρέφει λίστα από DataFrames
df_html = df_list[0]  # Ο πρώτος πίνακας
print("Πίνακας από HTML:")
print(df_html)
os.unlink(tmp_html_path)
print("\n" + "="*80 + "\n")

# ----------------------------------------------------------------------
# 7. pd.read_clipboard() - Ανάγνωση από πρόχειρο
# ----------------------------------------------------------------------
print("7. pd.read_clipboard() - Ανάγνωση από πρόχειρο")
print("-" * 60)
print("Για να λειτουργήσει, πρέπει να έχετε δεδομένα στο πρόχειρο (π.χ. από
Excel).")
print("Προσομοίωση: θα δημιουργήσουμε δεδομένα και θα τα αντιγράψουμε στο
πρόχειρο.")
print("Σε περιβάλλον χωρίς GUI, η read_clipboard() μπορεί να αποτύχει.")
print("Εδώ απλά δείχνουμε τη χρήση:")
print("df = pd.read_clipboard()")
# Για λόγους πληρότητας, προσπαθούμε να εκτελέσουμε αλλά με try/except
try:
    # Δημιουργούμε ένα string με δεδομένα και τοποθετούμε στο πρόχειρο (αν
υποστηρίζεται)
    # Σημείωση: Σε περιβάλλον χωρίς GUI, αυτό δεν θα λειτουργήσει.
    import pyperclip  # προαιρετική βιβλιοθήκη για clipboard
    sample_data = "Name\tAge\nEleni\t22\nDimitris\t27\n"
    pyperclip.copy(sample_data)
    df_clip = pd.read_clipboard()
    print("Δεδομένα από το πρόχειρο:")
    print(df_clip)
except ImportError:
    print("Η βιβλιοθήκη pyperclip δεν είναι εγκατεστημένη. Παραλείπεται η
εκτέλεση.")
```

```
    print("Μπορείτε να εγκαταστήσετε: pip install pyperclip")
except Exception as e:
    print(f"Δεν ήταν δυνατή η ανάγνωση από το πρόχειρο: {e}")
print("\n" + "="*80 + "\n")

# ----------------------------------------------------------------------
# 8. pd.json_normalize() - Μετατροπή ένθετου JSON σε επίπεδο DataFrame
# ----------------------------------------------------------------------
print("8. pd.json_normalize() - Επίπεδο (flatten) ένθετο JSON")
print("-" * 60)
nested_json = {
    'status': 'success',
    'data': [
        {'id': 1, 'name': 'Product A', 'details': {'price': 100, 'stock':
50}},
        {'id': 2, 'name': 'Product B', 'details': {'price': 200, 'stock':
30}},
        {'id': 3, 'name': 'Product C', 'details': {'price': 150, 'stock':
0}}
    ]
}
# Χρήση json_normalize στο πεδίο 'data' (η κύρια χρήση)
df_normalized = pd.json_normalize(nested_json['data'])
print("Επίπεδο DataFrame από το ένθετο JSON:")
print(df_normalized)
print("\n" + "="*80 + "\n")

print("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
```

Κ. 3.4.18: Παράδειγμα των μεθόδων για εισαγωγή δεδομένων στα Πλαισίων Δεδομένων

3.4.18. Εξαγωγή Δεδομένων

Μέθοδος	Περιγραφή	Επιστροφή
df.to_csv()	Εξάγει το DataFrame σε ένα αρχείο CSV. Με index=False αποφεύγεται η αποθήκευση των αριθμών γραμμών.	df.to_csv('το_αρχείο_εξαγωγής.csv', index=False)
df.to_excel()	Εξάγει το DataFrame σε ένα αρχείο Excel. Απαιτείται η εγκατάσταση της βιβλιοθήκης openpyxl.	df.to_excel('το_αρχείο_εξαγωγής.xlsx', sheet_name='Δεδομένα', index=False)
df.to_json()	Εξάγει το DataFrame σε ένα αρχείο μορφής JSON.	df.to_json('το_αρχείο_εξαγωγής.json', orient='records')
df.to_sql()	Εγγράφει τα δεδομένα του DataFrame σε έναν πίνακα μιας σχεσιακής βάσης δεδομένων.	from sqlalchemy import create_engine engine = create_engine('sqlite:///η_βάση_μου.db') df.to_sql('ο_πίνακας_μου', engine, if_exists='replace', index=False)
df.to_html()	Μετατρέπει το DataFrame σε έναν πίνακα HTML.	df.to_html('το_αρχείο_εξαγωγής.html')
df.to_clipboard()	Αντιγράφει το DataFrame στο πρόχειρο, ώστε να μπορείτε να το επικολλήσετε απευθείας σε άλλη εφαρμογή (π.χ. Excel).	df.to_clipboard()

Πίνακας 3.4.19: Μέθοδοι για εξαγωγή δεδομένων στα Πλαισίων Δεδομένων

Οι μέθοδοι to_... σάς επιτρέπουν να μοιραστείτε τα αποτελέσματά σας με άλλους χρήστες ή να τα αποθηκεύσετε για μελλοντική χρήση.

Ας αναλύσουμε τις εντολές του πίνακα:

1. df.to_csv()

Η πιο συνηθισμένη μέθοδος αποθήκευσης.

- **Πώς λειτουργεί:** Δημιουργεί ένα αρχείο κειμένου όπου οι τιμές χωρίζονται με κόμματα.
- **Η παράμετρος index=False:** Είναι εξαιρετικά σημαντική. Αν δεν τη χρησιμοποιήσετε, η Pandas θα προσθέσει μια επιπλέον στήλη στην αρχή με τους αριθμούς των γραμμών (0, 1, 2...), κάτι που συνήθως δεν χρειαζόμαστε στα αρχεία εξαγωγής.
- **Encoding:** Αν έχετε ελληνικά, η χρήση της παραμέτρου encoding='utf-8-sig' εξασφαλίζει ότι το Excel θα αναγνωρίσει σωστά τους χαρακτήρες όταν ανοίξετε το CSV.

2. df.to_excel()

Ιδανική για τη δημιουργία αναφορών που προορίζονται για μη προγραμματιστές.

- **sheet_name:** Σας επιτρέπει να ονομάσετε το φύλλο εργασίας (π.χ. "Πωλήσεις 2024").
- **Πολλαπλά φύλλα:** Αν θέλετε να γράψετε πολλά DataFrames στο ίδιο αρχείο Excel αλλά σε διαφορετικά φύλλα, θα χρειαστείτε το αντικείμενο pd.ExcelWriter.

3. df.to_json()

Χρησιμοποιείται κυρίως όταν τα δεδομένα πρόκειται να τροφοδοτήσουν μια ιστοσελίδα ή μια εφαρμογή.

- **orient='records':** Αυτή η παράμετρος είναι η πιο δημοφιλής, καθώς μετατρέπει το DataFrame σε μια λίστα από λεξικά (π.χ. [{"id":1, "name":"Anna"}, ...]), μορφή που είναι η στάνταρ για τα περισσότερα web APIs.

4. df.to_sql()

Η μέθοδος για την ενημέρωση βάσεων δεδομένων.

- **if_exists:** Πολύ κρίσιμη παράμετρος.
  - 'fail': Δεν κάνει τίποτα αν ο πίνακας υπάρχει ήδη.
  - 'replace': Διαγράφει τον παλιό πίνακα και φτιάχνει νέο.
  - 'append': Προσθέτει τα νέα δεδομένα στο τέλος του υπάρχοντος πίνακα.
- **Σύνδεση:** Όπως και στην ανάγνωση, απαιτεί ένα "engine" από τη βιβλιοθήκη SQLAlchemy.

5. df.to_html()

Μετατρέπει τα δεδομένα σε κώδικα ιστοσελίδας.

- **Εφαρμογή:** Χρήσιμη αν θέλετε να ενσωματώσετε έναν πίνακα σε ένα email (σε μορφή HTML) ή σε ένα web dashboard. Παράγει ολόκληρη τη δομή <table>...</table>.

6. df.to_clipboard()

Ο "μαγικός" τρόπος για μεταφορά δεδομένων χωρίς αρχεία.

- **Πώς λειτουργεί:** Αντιγράφει τα δεδομένα σας στο πρόχειρο του συστήματος.
- **Χρήση:** Τρέχετε την εντολή και μετά κάνετε απλώς "Επικόλληση" (Ctrl+V) σε ένα κενό φύλλο Excel ή σε ένα έγγραφο Word. Είναι ο ταχύτερος τρόπος για να στείλετε ένα δείγμα δεδομένων σε έναν συνάδελφο μέσω chat.

```
import pandas as pd
import numpy as np
import os
import tempfile
from sqlalchemy import create_engine
import webbrowser  # για προεπισκόπηση HTML

# ---------------------------------------------------------------------
# Δημιουργία δείγματος DataFrame για εξαγωγή
# ---------------------------------------------------------------------
print("ΔΗΜΙΟΥΡΓΙΑ ΔΕΙΓΜΑΤΟΣ DataFrame")
print("-" * 70)

data = {
    'Name': ['Anna', 'George', 'Maria', 'Nikos', 'Eleni'],
    'Age': [25, 30, 28, 35, 22],
    'Salary': [50000, 60000, 55000, 65000, 48000],
    'Department': ['Sales', 'IT', 'Sales', 'HR', 'IT'],
    'Start_Date': pd.date_range('2023-01-01', periods=5, freq='M')
}
df = pd.DataFrame(data)
print("Δείγμα DataFrame:")
print(df)
print("\n" + "="*100 + "\n")

# ---------------------------------------------------------------------
# 1. df.to_csv() - Εξαγωγή σε CSV
# ---------------------------------------------------------------------
print("1. df.to_csv() - Εξαγωγή σε CSV")
print("-" * 60)

# Δημιουργία προσωρινού αρχείου CSV
with tempfile.NamedTemporaryFile(mode='w', suffix='.csv', delete=False,
encoding='utf-8') as tmp_csv:
    csv_path = tmp_csv.name

df.to_csv(csv_path, index=False)
print(f"Το DataFrame αποθηκεύτηκε στο προσωρινό αρχείο: {csv_path}")
print("Περιεχόμενο του αρχείου CSV:")
with open(csv_path, 'r', encoding='utf-8') as f:
    print(f.read())

os.unlink(csv_path)  # Διαγραφή προσωρινού αρχείου
print("\n" + "="*100 + "\n")

# ---------------------------------------------------------------------
# 2. df.to_excel() - Εξαγωγή σε Excel
# ---------------------------------------------------------------------
print("2. df.to_excel() - Εξαγωγή σε Excel")
print("-" * 60)

try:
```

```
    with tempfile.NamedTemporaryFile(suffix='.xlsx', delete=False) as
tmp_xlsx:
        excel_path = tmp_xlsx.name

    df.to_excel(excel_path, sheet_name='Employees', index=False,
engine='openpyxl')
    print(f"Το DataFrame αποθηκεύτηκε στο προσωρινό αρχείο: {excel_path}")
    print("Μέγεθος αρχείου (bytes):", os.path.getsize(excel_path))
    # Προαιρετικά: διαβάζουμε πίσω για επιβεβαίωση
    df_read = pd.read_excel(excel_path, engine='openpyxl')
    print("Ανάγνωση από το Excel για επιβεβαίωση:")
    print(df_read)
    os.unlink(excel_path)
except ImportError:
    print("Η βιβλιοθήκη openpyxl δεν είναι εγκατεστημένη. Παραλείπεται το
παράδειγμα Excel.")
    print("Εγκαταστήστε την με: pip install openpyxl")
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 3. df.to_json() - Εξαγωγή σε JSON
# ----------------------------------------------------------------------
print("3. df.to_json() - Εξαγωγή σε JSON")
print("-" * 60)

with tempfile.NamedTemporaryFile(mode='w', suffix='.json', delete=False,
encoding='utf-8') as tmp_json:
    json_path = tmp_json.name

# Εξαγωγή με προσανατολισμό 'records' (λίστα αντικειμένων)
df.to_json(json_path, orient='records', indent=2)
print(f"Το DataFrame αποθηκεύτηκε στο προσωρινό αρχείο: {json_path}")
print("Περιεχόμενο JSON (orient='records'):")
with open(json_path, 'r', encoding='utf-8') as f:
    print(f.read())

os.unlink(json_path)
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 4. df.to_sql() - Εξαγωγή σε SQL βάση δεδομένων
# ----------------------------------------------------------------------
print("4. df.to_sql() - Εξαγωγή σε SQL (SQLite in-memory)")
print("-" * 60)

# Δημιουργία in-memory SQLite βάσης
engine = create_engine('sqlite:///:memory:')
# Εγγραφή του DataFrame σε πίνακα
df.to_sql('employees', engine, if_exists='replace', index=False)
print("Το DataFrame αποθηκεύτηκε στον πίνακα 'employees' της βάσης
SQLite.")

# Επιβεβαίωση με ανάγνωση
df_sql = pd.read_sql('SELECT * FROM employees', engine)
print("Ανάγνωση από τον πίνακα SQL για επιβεβαίωση:")
print(df_sql)
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 5. df.to_html() - Εξαγωγή σε HTML
# ----------------------------------------------------------------------
```

```
print("5. df.to_html() - Εξαγωγή σε HTML")
print("-" * 60)

with tempfile.NamedTemporaryFile(mode='w', suffix='.html', delete=False,
encoding='utf-8') as tmp_html:
    html_path = tmp_html.name

df.to_html(html_path, index=False, border=1)
print(f"Το DataFrame αποθηκεύτηκε στο προσωρινό αρχείο: {html_path}")
print("Περιεχόμενο HTML (πρώτες 5 γραμμές):")
with open(html_path, 'r', encoding='utf-8') as f:
    html_content = f.read()
    print(html_content[:500] + "...")  # Εμφάνιση αρχής του HTML

# Προαιρετικά: άνοιγμα στο browser (σχολιασμένο για αποφυγή)
# webbrowser.open(html_path)

os.unlink(html_path)
print("\n" + "="*100 + "\n")

# ----------------------------------------------------------------------
# 6. df.to_clipboard() - Αντιγραφή στο πρόχειρο
# ----------------------------------------------------------------------
print("6. df.to_clipboard() - Αντιγραφή στο πρόχειρο")
print("-" * 60)

print("Η εντολή df.to_clipboard() αντιγράφει το DataFrame στο πρόχειρο.")
print("Στη συνέχεια, μπορείτε να το επικολλήσετε (π.χ. σε Excel ή άλλο
πρόγραμμα).")

try:
    df.to_clipboard(index=False)
    print("Το DataFrame αντιγράφηκε στο πρόχειρο.")
    print("Δοκιμάστε να το επικολλήσετε σε ένα πρόγραμμα.")
except Exception as e:
    print(f"Δεν ήταν δυνατή η αντιγραφή στο πρόχειρο: {e}")
    print("Αυτό μπορεί να συμβαίνει σε περιβάλλον χωρίς GUI ή χωρίς
υποστήριξη clipboard.")
print("\n" + "="*100 + "\n")

print("ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ")
```

Κ. 3.4.19: Παράδειγμα των μεθόδων για εξαγωγή δεδομένων στα Πλαισίων Δεδομένων

3.4.19. Ολοκληρωμένο Παράδειγμα στα Πλαίσια Δεδομένων: Διαχείριση Προϊόντων

Τίτλος: **Σύστημα Διαχείρισης Προϊόντων με Pandas**

Το πρόγραμμα χρησιμοποιεί τη βιβλιοθήκη pandas για τη διαχείριση καταλόγου προϊόντων. Το σύστημα περιέχει τις εξής λειτουργίες:

1. Δημιουργία και αρχικοποίηση DataFrame με τα εξής πεδία:

- **product_name** (όνομα προϊόντος)
- **price** (τιμή)

- **manufacturer** (κατασκευαστής)

2. Βασικές λειτουργίες:

- Προσθήκη νέων προϊόντων
- Εμφάνιση όλων των προϊόντων
- Αναζήτηση προϊόντων βάσει κατασκευαστή
- Ενημέρωση τιμών προϊόντων
- Διαγραφή προϊόντων

3. Αποθήκευση και Ανάκτηση:

- Αποθήκευση δεδομένων σε CSV αρχείο
- Φόρτωση δεδομένων από CSV αρχείο
- Επαλήθευση ότι τα δεδομένα διατηρούνται σωστά

```
import pandas as pd
import os

class ProductManager:
    def __init__(self, filename='products.csv'):
        self.filename = filename
        self.products_df = self.load_products()

    def load_products(self):
        """Φόρτωση προϊόντων από αρχείο ή δημιουργία νέου DataFrame"""
        if os.path.exists(self.filename):
            try:
                df = pd.read_csv(self.filename)
                print("✅ Τα δεδομένα φορτώθηκαν επιτυχώς από το αρχείο")
                return df
            except Exception as e:
                print(f"❌ Σφάλμα φόρτωσης αρχείου: {e}")

        # Δημιουργία κενού DataFrame αν το αρχείο δεν υπάρχει
        empty_df = pd.DataFrame(columns=['product_name', 'price',
'manufacturer'])
        print("📂 Δημιουργήθηκε νέος κατάλογος προϊόντων")
        return empty_df

    def save_products(self):
        """Αποθήκευση προϊόντων σε αρχείο CSV"""
        try:
            self.products_df.to_csv(self.filename, index=False)
            print("✅ Τα δεδομένα αποθηκεύτηκαν επιτυχώς")
        except Exception as e:
            print(f"❌ Σφάλμα αποθήκευσης: {e}")

    def add_product(self, product_name, price, manufacturer):
```

```
        """Προσθήκη νέου προϊόντος"""
        new_product = pd.DataFrame({
            'product_name': [product_name],
            'price': [price],
            'manufacturer': [manufacturer]
        })

        self.products_df = pd.concat([self.products_df, new_product],
ignore_index=True)
        print(f"✅ Προστέθηκε το προϊόν: {product_name}")
        self.save_products()

    def display_all_products(self):
        """Εμφάνιση όλων των προϊόντων"""
        if self.products_df.empty:
            print("📭 Ο κατάλογος προϊόντων είναι άδειος")
            return

        print("\n📋 Κατάλογος Προϊόντων:")
        print("=" * 50)
        for index, row in self.products_df.iterrows():
            print(f"{index + 1}. {row['product_name']} - {row['price']}€ -
{row['manufacturer']}")
        print("=" * 50)
        print(f"Σύνολο προϊόντων: {len(self.products_df)}")

    def search_by_manufacturer(self, manufacturer):
        """Αναζήτηση προϊόντων βάσει κατασκευαστή"""
        filtered_df =
self.products_df[self.products_df['manufacturer'].str.lower() ==
manufacturer.lower()]

        if filtered_df.empty:
            print(f" Δεν βρέθηκαν προϊόντα του κατασκευαστή:
{manufacturer}")
            return

        print(f"\n Προϊόντα του κατασκευαστή '{manufacturer}':")
        print("=" * 40)
        for index, row in filtered_df.iterrows():
            print(f"- {row['product_name']} - {row['price']}€")
        print("=" * 40)

    def update_price(self, product_name, new_price):
        """Ενημέρωση τιμής προϊόντος"""
        mask = self.products_df['product_name'].str.lower() ==
product_name.lower()

        if mask.any():
            old_price = self.products_df.loc[mask, 'price'].values[0]
            self.products_df.loc[mask, 'price'] = new_price
            print(f"✅ Η τιμή του '{product_name}' ενημερώθηκε από
{old_price}€ σε {new_price}€")
            self.save_products()
        else:
            print(f"❌ Το προϊόν '{product_name}' δεν βρέθηκε")

    def delete_product(self, product_name):
        """Διαγραφή προϊόντος"""
        initial_count = len(self.products_df)
```

```
        self.products_df = self.products_df[
            self.products_df['product_name'].str.lower() !=
product_name.lower()
        ]

        if len(self.products_df) < initial_count:
            print(f"✅ Το προϊόν '{product_name}' διαγράφηκε")
            self.save_products()
        else:
            print(f"❌ Το προϊόν '{product_name}' δεν βρέθηκε")

    def display_statistics(self):
        """Εμφάνιση στατιστικών"""
        if self.products_df.empty:
            print("📊 Δεν υπάρχουν δεδομένα για στατιστικά")
            return

        print("\n📊 Στατιστικά Καταλόγου:")
        print(f"Σύνολο προϊόντων: {len(self.products_df)}")
        print(f"Μέση τιμή: {self.products_df['price'].mean():.2f}€")
        print(f"Υψηλότερη τιμή: {self.products_df['price'].max()}€")
        print(f{self.products_df['price'].min()}€")
        print(f"Αριθμός κατασκευαστών:
{self.products_df['manufacturer'].nunique()}")

        # Πιο δημοφιλείς κατασκευαστές
        manufacturer_counts =
self.products_df['manufacturer'].value_counts()
        print("\n🏆 Κατασκευαστές ανά αριθμό προϊόντων:")
        for manufacturer, count in manufacturer_counts.items():
            print(f"  {manufacturer}: {count} προϊόντα")

def main():
    # Δημιουργία διαχειριστή προϊόντων
    manager = ProductManager()

    while True:
        print("\n" + "=" * 50)
        print("🏢 ΣΥΣΤΗΜΑ ΔΙΑΧΕΙΡΙΣΗΣ ΠΡΟΪΟΝΤΩΝ")
        print("=" * 50)
        print("1. Προσθήκη νέου προϊόντος")
        print("2. Εμφάνιση όλων των προϊόντων")
        print("3. Αναζήτηση προϊόντος βάσει κατασκευαστή")
        print("4. Ενημέρωση τιμής προϊόντος")
        print("5. Διαγραφή προϊόντος")
        print("6. Στατιστικά")
        print("7. Έξοδος")
        print("=" * 50)

        choice = input("Επιλέξτε ενέργεια (1-7): ").strip()

        if choice == '1':
            print("\n➕ ΠΡΟΣΘΗΚΗ ΝΕΟΥ ΠΡΟΪΟΝΤΟΣ")
            product_name = input("Όνομα προϊόντος: ").strip()
            try:
                price = float(input("Τιμή (€): ").strip())
                manufacturer = input("Κατασκευαστής: ").strip()
                manager.add_product(product_name, price, manufacturer)
            except ValueError:
```

```
                print("❌ Λάθος μορφή τιμής. Παρακαλώ εισάγετε αριθμό.")

        elif choice == '2':
            manager.display_all_products()

        elif choice == '3':
            print("\n🔍 ΑΝΑΖΗΤΗΣΗ ΠΡΟΪΟΝΤΩΝ")
            manufacturer = input("Εισάγετε όνομα κατασκευαστή: ").strip()
            manager.search_by_manufacturer(manufacturer)

        elif choice == '4':
            print("\n💰 ΕΝΗΜΕΡΩΣΗ ΤΙΜΗΣ")
            product_name = input("Όνομα προϊόντος: ").strip()
            try:
                new_price = float(input("Νέα τιμή (€): ").strip())
                manager.update_price(product_name, new_price)
            except ValueError:
                print("❌ Λάθος μορφή τιμής. Παρακαλώ εισάγετε αριθμό.")

        elif choice == '5':
            print("\n🗑️ ΔΙΑΓΡΑΦΗ ΠΡΟΪΟΝΤΟΣ")
            product_name = input("Όνομα προϊόντος προς διαγραφή: ").strip()
            manager.delete_product(product_name)

        elif choice == '6':
            manager.display_statistics()

        elif choice == '7':
            print("👋 Έξοδος από το σύστημα. Αντίο!")
            break

        else:
            print("❌ Μη έγκυρη επιλογή. Παρακαλώ επιλέξτε 1-7.")

if __name__ == "__main__":
    main()
```

3.5. Ερωτήσεις Αυτοαξιολόγησης – Απαντήσεις και Εξήγηση

3.5.1. Test Γνώσεων: Pandas Series

1. Ποια μέθοδος επιστρέφει τις πρώτες 5 γραμμές μιας Σειράς;

α) .first()

β) .head()

γ) .top()

δ) .start()

2. Ποια είναι η βασική διαφορά μεταξύ .loc και .iloc;

α) Η .loc χρησιμοποιεί θέσεις (αριθμούς) και η .iloc ονόματα (labels).

β) Η .loc είναι πιο γρήγορη από την .iloc.

γ) Η .loc χρησιμοποιεί ονόματα (labels) και η .iloc αριθμητικές θέσεις.

δ) Δεν υπάρχει καμία διαφορά.

3. Αν θέλουμε να προσπελάσουμε πολύ γρήγορα μία μόνο τιμή (single scalar) με βάση τη θέση της, ποια μέθοδος είναι η προτιμότερη;

α) .iloc

β) .loc

γ) .at

δ) .iat

4. Τι επιστρέφει η μέθοδος .get('key', 0) αν το 'key' δεν υπάρχει στο index;

α) Σφάλμα (KeyError)

β) NaN

γ) Την τιμή 0

δ) Την πρώτη τιμή της Σειράς

5. Ποια μέθοδος χρησιμοποιείται για να κρατήσουμε μόνο τις τιμές που ικανοποιούν μια συνθήκη, μετατρέποντας τις υπόλοιπες σε NaN;

α) .mask()

β) .where()

γ) .filter()

δ) .drop()

6. Πώς ελέγχουμε αν οι τιμές μιας Σειράς βρίσκονται μέσα σε μια συγκεκριμένη λίστα;

α) .exists()

β) .contains()

γ) .isin()

δ) .where()

7. Ποια μέθοδος επιστρέφει True/False για κάθε στοιχείο, δείχνοντας αν είναι κενό (null);

α) .isnull()

β) .empty()

γ) .isna()

δ) Τα α και γ είναι σωστά

8. Η μέθοδος .dropna()...

α) Διαγράφει τις τιμές NaN από τη Σειρά.

β) Αντικαθιστά τα NaN με 0.

γ) Προσθέτει NaN σε κενά κελιά.

δ) Μετατρέπει τα NaN σε strings.

9. Τι κάνει η μέθοδος .fillna(method='ffill');

α) Γεμίζει τα κενά με τη μέση τιμή.

β) Γεμίζει τα κενά με την αμέσως επόμενη έγκυρη τιμή.

γ) Γεμίζει τα κενά με την αμέσως προηγούμενη έγκυρη τιμή.

δ) Διαγράφει τα κενά.

10. Ποια μέθοδος υπολογίζει τον μέσο όρο των τιμών;

α) .average()

β) .mean()

γ) .median()

δ) .sum()

11. Ποια εντολή δίνει μια πλήρη στατιστική σύνοψη (mean, std, min, max, κλπ);

α) .summary()

β) .stats()

γ) .describe()

δ) .info()

12. Τι υπολογίζει η μέθοδος .cumsum();

α) Το σωρευτικό γινόμενο.

β) Τη διαφορά από το προηγούμενο στοιχείο.

γ) Το σωρευτικό άθροισμα.

δ) Την ποσοστιαία μεταβολή.

13. Αν θέλουμε να περιορίσουμε όλες τις τιμές μιας Σειράς μεταξύ 0 και 100, ποια μέθοδος είναι η κατάλληλη;

α) .round()

β) .clip(0, 100)

γ) .between(0, 100)

δ) .set_range(0, 100)

14. Ποια μέθοδος υπολογίζει την ποσοστιαία μεταβολή μεταξύ τρέχουσας και προηγούμενης τιμής;

α) .diff()

β) .pct_change()

γ) .shift()

δ) .growth()

15. Πώς αλλάζουμε τον τύπο δεδομένων μιας Σειράς (π.χ. από float σε int);

α) .convert()

β) .to_type()

γ) .astype()

δ) .change_dtype()

16. Η μέθοδος .map() χρησιμοποιείται κυρίως για...

α) Ταξινόμηση δεδομένων.

β) Αντικατάσταση τιμών βάσει λεξικού ή συνάρτησης.

γ) Υπολογισμό μέσου όρου.

δ) Διαγραφή κενών τιμών.

17. Ποια μέθοδος επιστρέφει ένα NumPy array με τις μοναδικές τιμές της Σειράς;

α) .unique()

β) .nunique()

γ) .distinct()

δ) .value_counts()

18. Τι επιστρέφει η .value_counts();

α) Τον αριθμό των μοναδικών τιμών.

β) Τη συχνότητα εμφάνισης κάθε μοναδικής τιμής.

γ) Το άθροισμα όλων των τιμών.

δ) True αν υπάρχουν διπλοεγγραφές.

19. Πώς ελέγχουμε αν δύο Σειρές είναι πανομοιότυπες (ίδιο index και ίδιες τιμές);

α) s1 == s2

β) s1.compare(s2)

γ) s1.equals(s2)

δ) s1.is_same(s2)

20. Για να μετατρέψουμε όλα τα κείμενα μιας Σειράς σε κεφαλαία, χρησιμοποιούμε:

α) .upper()

β) .str.upper()

γ) .text.upper()

δ) .to_capital()

21. Ποια μέθοδος .str αφαιρεί τα κενά διαστήματα στην αρχή και στο τέλος κάθε κειμένου;

α) .str.clean()

β) .str.strip()

γ) .str.replace(' ', '')

δ) .str.cut()

22. Αν μια Σειρά περιέχει ημερομηνίες, πώς παίρνουμε μόνο το έτος (year);

α) .year

β) .dt.year

γ) .date.year

δ) .strftime('%Y')

23. Τι κάνει η μέθοδος .dt.is_month_end;

α) Επιστρέφει την τελευταία μέρα του μήνα.

β) Επιστρέφει True αν η ημερομηνία είναι η τελευταία μέρα του μήνα.

γ) Διαγράφει τις ημερομηνίες που δεν είναι τέλος μήνα.

δ) Υπολογίζει πόσες μέρες έμειναν μέχρι το τέλος του μήνα.

24. Οι "κατηγορικές" (Categorical) Σειρές βοηθούν στην...

α) Αύξηση της ακρίβειας των δεκαδικών.

β) Εξοικονόμηση μνήμης και ταχύτητα ταξινόμησης.

γ) Αυτόματη μετάφραση κειμένου.

δ) Σύνδεση με βάσεις δεδομένων SQL.

25. Ποια μέθοδος επιτρέπει τον υπολογισμό ενός "κινητού μέσου όρου" (moving average);

α) .expanding()

β) .rolling()

γ) .moving()

δ) .groupby()

26. Τι επιστρέφει η μέθοδος .to_dict();

α) Μια λίστα Python.

β) Ένα αρχείο JSON.

γ) Ένα λεξικό Python (index: value).

δ) Ένα DataFrame.

27. Η παράμετρος inplace=True στις μεθόδους των Pandas σημαίνει ότι...

α) Η αλλαγή θα γίνει μόνιμα στο υπάρχον αντικείμενο.

β) Θα δημιουργηθεί ένα νέο αντίγραφο της Σειράς.

γ) Η μέθοδος θα εκτελεστεί πιο αργά.

δ) Τα NaN θα αγνοηθούν.

28. Η παράμετρος skipna=True (που είναι συνήθως default)...

α) Διαγράφει τα NaN πριν την πράξη.

β) Αγνοεί τα NaN κατά τον υπολογισμό (π.χ. στο άθροισμα).

γ) Σταματάει την πράξη αν βρει NaN.

δ) Μετατρέπει τα NaN σε 1.

29. Ποια μέθοδος χρησιμοποιείται για να αλλάξουμε τα ονόματα του Index;

α) .set_index()

β) .rename()

γ) .reindex()

δ) .change_labels()

30. Ποια μέθοδος ελέγχει αν έστω και μία τιμή στη Σειρά είναι True (ή μη μηδενική);

α) .all()

β) .any()

γ) .exists()

δ) .check()

3.5.2. Test Γνώσεων: Pandas Series - Απαντήσεις & Επεξηγήσεις

1. **β) .head()**: Επιστρέφει τις n πρώτες γραμμές.
2. **γ)**: Η .loc βασίζεται σε ονόματα (labels), η .iloc σε ακέραιες θέσεις (0, 1, 2...).
3. **δ) .iat**: Είναι η ταχύτερη μέθοδος για πρόσβαση σε ένα μεμονωμένο κελί μέσω θέσης.
4. **γ) Την τιμή 0**: Η .get() επιτρέπει τον ορισμό default τιμής αν το κλειδί λείπει.
5. **β) .where()**: Διατηρεί τις τιμές όπου η συνθήκη ισχύει, αλλιώς βάζει NaN (ή ό,τι ορίσουμε).
6. **γ) .isin()**: Ελέγχει αν τα στοιχεία περιέχονται σε μια δομή (λίστα, set κλπ).
7. **δ) Τα α και γ είναι σωστά**: Και οι δύο μέθοδοι κάνουν την ίδια ακριβώς δουλειά.
8. **α)**: Αφαιρεί εντελώς τις γραμμές που έχουν NaN.
9. **γ)**: Το 'ffill' (forward fill) μεταφέρει την τελευταία έγκυρη τιμή προς τα εμπρός.
10. **β) .mean()**: Η στατιστική συνάρτηση για τον μέσο όρο.
11. **γ) .describe()**: Παρέχει count, mean, std, min, 25%, 50%, 75%, max.
12. **γ) Το σωρευτικό άθροισμα**: Προσθέτει κάθε τιμή στο τρέχον σύνολο.
13. **β) .clip(0, 100)**: "Κόβει" ό,τι είναι έξω από τα όρια $[0, 100]$.
14. **β) .pct_change()**: Υπολογίζει το $(V_{new} - V_{old}) / V_{old}$.
15. **γ) .astype()**: Η μέθοδος για "casting" τύπων δεδομένων.
16. **β)**: Χρησιμοποιείται για αντιστοίχιση τιμών (π.χ. 1 -> 'Male', 2 -> 'Female').
17. **α) .unique()**: Επιστρέφει τις μοναδικές τιμές χωρίς τις συχνότητές τους.
18. **β)**: Επιστρέφει μια Σειρά με τις συχνότητες κάθε τιμής ταξινομημένες φθίνουσα.
19. **γ) .equals(s2)**: Ελέγχει την απόλυτη ισότητα (Values + Index + Dtype).
20. **β) .str.upper()**: Πρέπει πάντα να καλούμε τον accessor .str για string functions.
21. **β) .str.strip()**: Αφαιρεί whitespace από τις άκρες.
22. **β) .dt.year**: Χρησιμοποιούμε τον accessor .dt για ημερομηνίες.
23. **β)**: Επιστρέφει Boolean Σειρά (True αν είναι η τελευταία μέρα του μήνα).
24. **β)**: Οι κατηγορίες αποθηκεύουν εσωτερικά ακέραιους κωδικούς αντί για μεγάλα strings.
25. **β) .rolling()**: Δημιουργεί ένα παράθυρο (window) για υπολογισμούς.
26. **γ)**: Μετατρέπει το Series σε Python Dictionary.
27. **α)**: Αν inplace=True, η αρχική μεταβλητή αλλάζει και δεν επιστρέφεται νέο αντικείμενο.
28. **β)**: Επιτρέπει τον υπολογισμό στατιστικών ακόμα και αν υπάρχουν κενά.
29. **β) .rename()**: Χρησιμοποιείται για αλλαγή ονομάτων σε index ή columns.
30. **β) .any()**: Επιστρέφει True αν υπάρχει τουλάχιστον ένα True στοιχείο.

3.5.3. Τεστ Γνώσεων Pandas DataFrames

Ενότητα 1: Εισαγωγή και Εξαγωγή Δεδομένων

1. Ποια εντολή χρησιμοποιούμε για να διαβάσουμε ένα αρχείο Excel;

α) pd.read_csv()

β) pd.read_json()

γ) pd.read_excel()

δ) pd.load_excel()

2. Ποια παράμετρος στην to_csv() αποτρέπει την αποθήκευση της στήλης των δεικτών (0, 1, 2...);

α) header=False

β) index=False

γ) save_index=No

δ) encoding='utf-8'

3. Τι επιστρέφει η εντολή pd.read_html();

α) Ένα DataFrame

β) Μια συμβολοσειρά (string)

γ) Μια λίστα από DataFrames

δ) Ένα αρχείο JSON

4. Ποια μέθοδος είναι η καταλληλότερη για να "ισιώσουμε" (flatten) ένα σύνθετο JSON με εμφωλευμένα πεδία;

α) pd.DataFrame()

β) pd.json_normalize()

γ) df.to_json()

δ) df.explode()

Ενότητα 2: Επισκόπηση και Καθαρισμός

5. Ποια μέθοδος δίνει μια γρήγορη στατιστική σύνοψη (mean, std, min, max) των αριθμητικών στηλών;

α) df.info()

β) df.head()

γ) df.describe()

δ) df.summary()

6. Πώς εντοπίζουμε τις ελλείπουσες τιμές (NaN) σε ένα DataFrame;

α) df.isna()

β) df.missing()

γ) df.find_null()

δ) df.empty()

7. Με ποια μέθοδο αφαιρούμε εντελώς τις γραμμές που περιέχουν τουλάχιστον μία τιμή NaN;

α) df.fillna()

β) df.dropna()

γ) df.remove_nan()

δ) df.clean()

8. Ποια ιδιότητα (property) μας επιστρέφει τον αριθμό των γραμμών και των στηλών (π.χ. (100, 5));

α) df.size

β) df.columns

γ) df.shape

δ) df.count()

Ενότητα 3: Φιλτράρισμα και Επιλογή

9. Ποια μέθοδος επιτρέπει την επιλογή δεδομένων με βάση τις ετικέτες (labels) των γραμμών και των στηλών;

α) df.iloc[]

β) df.loc[]

γ) df.select()

δ) df.filter()

10. Τι κάνει η εντολή df.iloc[0:5, :];

α) Επιλέγει τις πρώτες 5 στήλες

β) Επιλέγει τις πρώτες 5 γραμμές και όλες τις στήλες

γ) Επιλέγει όλες τις γραμμές όπου η τιμή είναι μεταξύ 0 και 5

δ) Επιλέγει τις τελευταίες 5 γραμμές

11. Ποιος είναι ο πιο ευανάγνωστος τρόπος να φιλτράρουμε δεδομένα χρησιμοποιώντας ένα string (π.χ. 'Age > 18');

α) df.loc()

β) df.query()

γ) df.filter()

δ) df.where()

12. Ποια μέθοδος ελέγχει αν οι τιμές μιας στήλης περιέχονται σε μια συγκεκριμένη λίστα;

α) df.isin()

β) df.match()

γ) df.contains()

δ) df.equals()

Ενότητα 4: Στατιστική και Ομαδοποίηση

13. Ποιο είναι το μοντέλο λειτουργίας της groupby();

α) Load-Transform-Save

β) Split-Apply-Combine

γ) Read-Filter-Write

δ) Join-Merge-Reshape

14. Ποια μέθοδος στο groupby επιστρέφει αποτέλεσμα με το ίδιο μήκος με το αρχικό DataFrame;

α) agg()

β) apply()

γ) transform()

δ) size()

15. Πώς υπολογίζουμε το αθροιστικό άθροισμα (running total) μιας στήλης;

α) df.sum()

β) df.cumsum()

γ) df.rolling().sum()

δ) df.expanding().total()

16. Ποια μέθοδος επιστρέφει τον αριθμό των μοναδικών τιμών σε μια στήλη;

α) unique()

β) value_counts()

γ) nunique()

δ) count()

Ενότητα 5: Χειρισμός Κειμένου (Strings)

17. Με ποιον accessor έχουμε πρόσβαση στις μεθόδους κειμένου;

α) .text

β) .str

γ) .txt

δ) .string

18. Ποια μέθοδος αφαιρεί τα κενά από την αρχή και το τέλος ενός string;

α) strip()

β) clean()

γ) lower()

δ) replace()

19. Τι επιστρέφει η df['name'].str.contains('Anna');

α) Μια λίστα με ονόματα

β) Ένα νέο DataFrame

γ) Μια boolean Series (True/False)

δ) Τον αριθμό των εμφανίσεων

20. Ποια μέθοδος χωρίζει ένα string σε δύο ή περισσότερες στήλες βάσει ενός διαχωριστικού;

α) slice()

β) split(expand=True)

γ) extract()

δ) partition()

Ενότητα 6: Χρονοσειρές (Datetimes)

21. Με ποιον accessor έχουμε πρόσβαση στις ιδιότητες ημερομηνίας;

α) .date

β) .time

γ) .dt

δ) .ts

22. Πώς παίρνουμε το όνομα της ημέρας (π.χ. "Monday") από μια στήλη datetime;

α) .dt.day

β) .dt.weekday

γ) .dt.day_name()

δ) .dt.isocalendar()

23. Τι κάνει η μέθοδος .dt.normalize();

α) Μετατρέπει την ημερομηνία σε string

β) Μηδενίζει την ώρα (00:00:00) κρατώντας μόνο την ημερομηνία

γ) Αλλάζει τη ζώνη ώρας

δ) Στρογγυλοποιεί στο πλησιέστερο έτος

Ενότητα 7: Παράθυρα (Rolling & Expanding)

24. Τι υπολογίζει η df['sales'].rolling(window=7).mean();

α) Τον μέσο όρο όλων των πωλήσεων

β) Τον κινητό μέσο όρο των τελευταίων 7 ημερών

γ) Τον μέσο όρο ανά εβδομάδα του έτους

δ) Τις πωλήσεις της 7ης ημέρας

25. Ποια μέθοδος δίνει μεγαλύτερη βαρύτητα στις πιο πρόσφατες τιμές;

α) rolling()

β) expanding()

γ) ewm()

δ) cumsum()

26. Τι καθορίζει η παράμετρος min_periods;

α) Το συνολικό μήκος του DataFrame

β) Τον ελάχιστο αριθμό μη-κενών τιμών για να βγει αποτέλεσμα στο παράθυρο

γ) Τη συχνότητα της χρονοσειράς

δ) Τον αριθμό των στηλών

Ενότητα 8: Σύγκριση και Συνδυασμός

27. Ποια μέθοδος δείχνει τις διαφορές μεταξύ δύο DataFrames δίπλα-δίπλα;

α) equals()

β) compare()

γ) diff()

δ) match()

28. Ποια μέθοδος ελέγχει αν δύο DataFrames είναι ακριβώς ίδια και επιστρέφει ένα μόνο True/False;

α) ==

β) eq()

γ) equals()

δ) all()

29. Πώς προσθέτουμε δύο DataFrames αντικαθιστώντας τα NaN με 0 κατά την πράξη;

α) df1 + df2

β) df1.add(df2, fill_value=0)

γ) df1.sum(df2)

δ) df1.append(df2)

30. Ποια μέθοδος επιτρέπει την εφαρμογή μιας προσαρμοσμένης συνάρτησης σε κάθε γραμμή ενός DataFrame;

α) apply()

β) map()

γ) transform()

δ) agg()

3.5.4. Τεστ Γνώσεων Pandas DataFrames - Απαντήσεις και Εξηγήσεις

1. **γ (pd.read_excel)**: Ειδική συνάρτηση για αρχεία .xlsx/.xls.
2. **β (index=False)**: Αν δεν οριστεί, η Pandas αποθηκεύει το index ως πρώτη στήλη.
3. **γ (Μια λίστα από DataFrames)**: Επειδή μια σελίδα HTML μπορεί να έχει πολλούς πίνακες (<table>).
4. **β (pd.json_normalize)**: Ξεδιπλώνει τα ιεραρχικά JSON σε επίπεδη μορφή.
5. **γ (df.describe)**: Παράγει στατιστικά (count, mean, std κ.λπ.).
6. **α (df.isna)**: Επιστρέφει True όπου υπάρχει NaN.
7. **β (df.dropna)**: "Πετάει" τις γραμμές με ελλιπή δεδομένα.
8. **γ (df.shape)**: Επιστρέφει πλειάδα (rows, columns).
9. **β (df.loc)**: Label-based επιλογή.
10. **β (Επιλέγει τις πρώτες 5 γραμμές...)**: Το 0:5 στο iloc είναι αποκλειστικό για το 5 (0 έως 4), το : σημαίνει όλες οι στήλες.
11. **β (df.query)**: Επιτρέπει σύνταξη τύπου SQL.
12. **α (df.isin)**: Πολύ γρήγορο για φιλτράρισμα βάσει λίστας.
13. **β (Split-Apply-Combine)**: Η θεμελιώδης αρχή της ομαδοποίησης.
14. **γ (transform)**: Επιστρέφει Series ίδιου μεγέθους με το αρχικό (χρήσιμο για κανονικοποίηση).
15. **β (df.cumsum)**: Cumulative Sum.
16. **γ (nunique)**: Number of Unique values.
17. **β (.str)**: Απαραίτητο πρόθεμα για string operations.
18. **α (strip)**: Αφαιρεί whitespace.
19. **γ (Μια boolean Series)**: Ελέγχει κάθε γραμμή και βγάζει True ή False.
20. **β (split(expand=True))**: Το expand=True δημιουργεί νέες στήλες αντί για λίστες.
21. **γ (.dt)**: Datetime accessor.
22. **γ (.dt.day_name())**: Επιστρέφει το όνομα της ημέρας ως κείμενο.
23. **β (Μηδενίζει την ώρα)**: Χρήσιμο όταν μας ενδιαφέρει μόνο η ημερομηνία.
24. **β (Κινητό μέσο όρο)**: Υπολογίζει τον μέσο όρο σε "παράθυρο" 7 θέσεων.
25. **γ (ewm)**: Exponential Weighted Moving average.
26. **β (Ελάχιστος αριθμός τιμών)**: Επιτρέπει υπολογισμούς ακόμα και αν το παράθυρο δεν είναι "γεμάτο".
27. **β (compare)**: Δείχνει τις αλλαγές (self/other) ανά κελί.
28. **γ (equals)**: Ελέγχει αντικείμενο προς αντικείμενο, όχι στοιχείο προς στοιχείο.

29. **β (df1.add(df2, fill_value=0))**: Οι μέθοδοι (add, sub κ.λπ.) είναι πιο ασφαλείς από τους τελεστές (+, -) σε αραιά δεδομένα.
30. **α (apply)**: Η πιο ευέλικτη μέθοδος για custom υπολογισμούς ανά γραμμή (axis=1).

ΚΕΦΑΛΑΙΟ 4: ΠΕΡΙΓΡΑΦΙΚΗ ΣΤΑΤΙΣΤΙΚΗ ΚΑΙ ΟΠΤΙΚΟΠΟΙΗΣΗ ΔΕΔΟΜΕΝΩΝ

Η **Περιγραφική Στατιστική (Descriptive Statistics)** αποτελεί το θεμέλιο της Επιστήμης Δεδομένων (Data Science). Πρόκειται για το σύνολο των μεθόδων που χρησιμοποιούνται για τη συλλογή, οργάνωση, σύνοψη και παρουσίαση δεδομένων με τρόπο εύληπτο και κατανοητό.

Σε αντίθεση με την Επαγωγική Στατιστική, η οποία προσπαθεί να κάνει προβλέψεις ή γενικεύσεις για έναν πληθυσμό, η Περιγραφική Στατιστική εστιάζει αποκλειστικά στο να περιγράψει "τι συμβαίνει" στα δεδομένα που έχουμε στα χέρια μας τώρα.

Στο περιβάλλον της Python, η διαδικασία αυτή γίνεται εξαιρετικά ισχυρή μέσω εξειδικευμένων βιβλιοθηκών:

- **Pandas:** Για τη φόρτωση, καθαρισμό και διαχείριση των δεδομένων (DataFrames).
- **NumPy:** Για γρήγορους μαθηματικούς υπολογισμούς.
- **Matplotlib & Seaborn & Plotly**: Για τη δημιουργία πλούσιων και ενημερωτικών γραφημάτων.

Το παρόν κεφάλαιο δομείται σε δύο κεντρικούς πυλώνες:

- **Περιγραφική στατιστική: Ποσοτική προσέγγιση** (Αριθμητική σύνοψη των δεδομένων).
- **Περιγραφική στατιστική: Οπτική προσέγγιση** (Αναγνώριση μοτίβων μέσω γραφημάτων).

Με την μελέτη αυτών των σημειώσεων, θα είστε σε θέση να:

- **Διακρίνετε τους τύπους δεδομένων**: Κατανόηση της διαφοράς μεταξύ ποιοτικών (κατηγορικών) και ποσοτικών (αριθμητικών) μεταβλητών και πώς η Python τις διαχειρίζεται.
- **Υπολογίζετε Μέτρα Κεντρικής Τάσης**: Χρήση εντολών για την εύρεση του μέσου όρου (mean), της διαμέσου (median) και της επικρατούσας τιμής (mode) για τον εντοπισμό του "κέντρου" των δεδομένων.
- **Αξιολογείτε τη Μεταβλητότητα**: Υπολογισμός μέτρων διασποράς, όπως η τυπική απόκλιση (standard deviation), η διακύμανση (variance) και το εύρος (range), για να κατανοήσετε την εξάπλωση των τιμών.
- **Αναγνωρίζετε τη Μορφή της Κατανομής**: Χρήση μέτρων όπως η λοξότητα (skewness) και η κύρτωση (kurtosis).

- **Δημιουργείτε Επεξηγηματικά Γραφήματα**: Κατασκευή ιστογραμμάτων, θηκογραμμάτων (boxplots) και διαγραμμάτων διασποράς (scatter plots) για την οπτική επαλήθευση των αριθμητικών συμπερασμάτων.
- **Εντοπίζετε Ακραίες Τιμές (Outliers)**: Χρήση στατιστικών και οπτικών μεθόδων για τον εντοπισμό τιμών που αποκλίνουν σημαντικά από τα υπόλοιπα δεδομένα.

4.1. Περιγραφική Στατιστική: Ποσοτική Προσέγγιση

Η Ποσοτική Προσέγγιση στην περιγραφική στατιστική είναι η διαδικασία σύνοψης ενός συνόλου δεδομένων μέσω αριθμητικών δεικτών. Όταν έχουμε χιλιάδες ή εκατομμύρια εγγραφές, είναι αδύνατον να βγάλουμε συμπέρασμα κοιτάζοντας απλώς τις "ωμές" τιμές (raw data). Χρειαζόμαστε έναν τρόπο να συμπυκνώσουμε την πληροφορία σε μερικούς, εύκολα διαχειρίσιμους αριθμούς που περιγράφουν την «ταυτότητα» του δείγματός μας.

Σε αυτό το κεφάλαιο, θα εστιάσουμε στην αριθμητική σύνοψη. Στόχος μας είναι να απαντήσουμε σε συγκεκριμένα ερωτήματα χρησιμοποιώντας την Python: «Ποια είναι η τυπική τιμή;», «Πόσο διαφέρουν οι τιμές μεταξύ τους;», «Υπάρχουν ακραίες τιμές που στρεβλώνουν την εικόνα;».

Η ποσοτική περιγραφή βασίζεται σε τρεις κύριους πυλώνες:

α) **Μέτρα Κεντρικής Τάσης (Measures of Central Tendency).** Αυτοί οι δείκτες προσπαθούν να εντοπίσουν το «κέντρο» της κατανομής. Μας λένε πού τείνουν να συγκεντρώνονται τα δεδομένα μας.

- **Μέσος Όρος (Mean):** Ο αριθμητικός μέσος, η πιο κοινή μέτρηση, αλλά ευαίσθητη σε ακραίες τιμές.
- **Διάμεσος (Median):** Η τιμή που χωρίζει τα δεδομένα ακριβώς στη μέση. Είναι πιο ανθεκτική (robust) σε ακραίες τιμές.
- **Επικρατούσα Τιμή (Mode):** Η τιμή που εμφανίζεται συχνότερα στο σύνολο δεδομένων.

β) **Μέτρα Διασποράς ή Μεταβλητότητας (Measures of Dispersion).** Η γνώση του κέντρου δεν αρκεί. Δύο τμήματα μαθητών μπορεί να έχουν τον ίδιο μέσο όρο βαθμολογίας (π.χ. 15), αλλά στο ένα όλοι να έγραψαν από 14 έως 16 και στο άλλο από 10 έως 20. Τα μέτρα διασποράς μας δείχνουν πόσο «απλωμένα» είναι τα δεδομένα.

- **Εύρος** (Range): Η απόσταση μεταξύ της μέγιστης και της ελάχιστης τιμής.
- **Διακύμανση** (Variance): Ο μέσος όρος των τετραγώνων των αποκλίσεων από τον μέσο όρο.

- **Τυπική Απόκλιση (Standard Deviation):** Το πιο σημαντικό μέτρο διασποράς, που μας δείχνει τη μέση απόσταση των παρατηρήσεων από τον μέσο όρο.
- **Ενδοτεταρτημοριακό Εύρος (IQR):** Το εύρος του μεσαίου 50% των δεδομένων μας.
- **Συντελεστής μεταβλητότητας (Coefficient of Variation – CV):** Τον χρησιμοποιούμε για να συγκρίνουμε τη μεταβλητότητα δύο διαφορετικών δεδομένων που έχουν διαφορετικές μονάδες μέτρησης ή πολύ διαφορετικούς μέσους όρους.

γ) **Μέτρα Θέσης και Σχήματος (Measures of Position & Shape)**

- **Εκατοστημόρια & Τεταρτημόρια (Percentiles & Quartiles):** Μας βοηθούν να καταλάβουμε τη σχετική θέση μιας τιμής (π.χ. «είσαι στο ανώτερο 10%»).
- **Λοξότητα (Skewness)** & **Κύρτωση (Kurtosis):** Αριθμητικοί δείκτες που περιγράφουν τη συμμετρία και την «οξύτητα» της καμπύλης των δεδομένων μας, χωρίς να χρειαστεί να τη ζωγραφίσουμε.

Στην Python, η βιβλιοθήκη Pandas μας προσφέρει εργαλεία για να υπολογίσουμε όλα τα παραπάνω είτε μεμονωμένα, είτε μαζικά με εντολές όπως η describe(), μετατρέποντας πολύπλοκους μαθηματικούς τύπους σε μία γραμμή κώδικα.

4.1.1. Μέτρα Κεντρικής Τάσης, Διασποράς και Θέσης

Ο **Μέσος Όρος (Mean)** είναι το άθροισμα όλων των τιμών διαιρεμένο με το πλήθος τους (Κ. 4.1.1).

- **Πλεονέκτημα:** Λαμβάνει υπόψη όλες τις τιμές.
- **Μειονέκτημα:** Είναι εξαιρετικά ευαίσθητος σε ακραίες τιμές (outliers). Μια πολύ μεγάλη τιμή μπορεί να τραβήξει τον μέσο όρο προς τα πάνω, δίνοντας ψευδή εικόνα.

```
import pandas as pd

# Παράδειγμα: Μισθοί υπαλλήλων (σε εκατοντάδες €)
# Παρατηρήστε την τιμή 100 (ακραία τιμή)
data = {'Μισθός': [10, 12, 12, 14, 15, 16, 100]}
df = pd.DataFrame(data)

# Υπολογισμός Μέσου Όρου
mean_val = df['Μισθός'].mean()

print(f"Μέσος Όρος: {mean_val:.2f}")
# Αποτέλεσμα: Μέσος Όρος: 25.57
# Παρατήρηση: Ο μέσος όρος (25.57) είναι πολύ μεγαλύτερος από τους
περισσότερους μισθούς (10-16)
# λόγω του 100.
```

Κ. 4.1.1 Υπολογισμός Μέσου Όρου

Η **Διάμεσος (Median)** είναι η μεσαία τιμή των παρατηρήσεων όταν αυτές διαταχθούν σε αύξουσα σειρά. Αν το πλήθος είναι ζυγό, είναι ο μέσος όρος των δύο μεσαίων (Κ. 4.1.2).

- **Πλεονέκτημα:** Δεν επηρεάζεται από ακραίες τιμές (Robust).
- **Χρήση:** Προτιμάται όταν τα δεδομένα έχουν λοξότητα (π.χ. εισοδήματα, τιμές ακινήτων).

```
# Υπολογισμός Διαμέσου
median_val = df['Μισθός'].median()

print(f"Διάμεσος: {median_val}")
# Αποτέλεσμα: Διάμεσος: 14.0
# Παρατήρηση: Η διάμεσος (14) αντιπροσωπεύει καλύτερα τον "τυπικό" υπάλληλο
# σε σχέση με τον μέσο όρο (25.57).
```

Κ. 4.1.2 Υπολογισμός Διαμέσου

Η **Επικρατούσα Τιμή (mode)** είναι η τιμή που εμφανίζεται με τη μεγαλύτερη συχνότητα στο σύνολο δεδομένων (Κ. 4.1.3).

- **Ιδιαιτερότητα:** Μπορεί να υπάρχει μία, περισσότερες από μία (δίκορφα δεδομένα) ή καμία επικρατούσα τιμή.
- **Χρήση:** Είναι το μοναδικό μέτρο που μπορεί να χρησιμοποιηθεί και για ποιοτικά (κατηγορικά) δεδομένα (π.χ. "Ποιο είναι το πιο δημοφιλές χρώμα;").

```
# Υπολογισμός Επικρατούσας Τιμής
# Η .mode() επιστρέφει πάντα Series (λίστα), γιατί μπορεί να υπάρχουν
ισοπαλίες.
mode_val = df['Μισθός'].mode()

print(f"Επικρατούσα Τιμή:\n{mode_val}")
# Αποτέλεσμα:
# 0    12
# Name: Μισθός, dtype: int64
# Παρατήρηση: Το 12 εμφανίζεται πιο συχνά (2 φορές).

# Παράδειγμα με κατηγορικά δεδομένα
colors = pd.Series(['Κόκκινο', 'Μπλε', 'Κόκκινο', 'Πράσινο'])
print(f"Δημοφιλέστερο χρώμα: {colors.mode()[0]}")
# Αποτέλεσμα: Δημοφιλέστερο χρώμα: Κόκκινο
```

Κ. 4.1.3 Υπολογισμός Επικρατούσας Τιμής

Η Διακύμανση και η Τυπική Απόκλιση είναι τα πιο σημαντικά και ευρέως χρησιμοποιούμενα μέτρα διασποράς στην στατιστική. Μας δείχνουν πόσο "απλωμένες" είναι οι τιμές γύρω από τον μέσο όρο.

Η **Διακύμανση (Variance)** Είναι ο μέσος όρος των τετραγώνων των αποκλίσεων κάθε τιμής από τον μέσο όρο.

- **Γιατί τετράγωνα;** Για να μετατρέψουμε τις αρνητικές διαφορές σε θετικές (αλλιώς το άθροισμα θα ήταν μηδέν) και για να "τιμωρήσουμε" περισσότερο τις μεγάλες αποκλίσεις.

- **Το πρόβλημα:** Το αποτέλεσμα είναι σε τετραγωνισμένες μονάδες (π.χ. "τετραγωνικά ευρώ" ή "τετραγωνικά κιλά"), πράγμα που δεν βγάζει διαισθητικό νόημα.

Η **Τυπική Απόκλιση (Standard Deviation)** Είναι η τετραγωνική ρίζα της διακύμανσης. Ουσιαστικά, επαναφέρει τις μονάδες μέτρησης στην αρχική τους μορφή. Μας λέει, κατά μέσο όρο, πόσο μακριά βρίσκεται μια τυπική παρατήρηση από το κέντρο (τον μέσο όρο). Όταν η Τυπική Απόκλιση είναι μικρή τα δεδομένα συγκεντρώνονται κοντά στον μέσο όρο (σταθερότητα). Ενώ μεγάλη Τυπική Απόκλιση υποδηλώνει ότι τα δεδομένα είναι απλωμένα (αστάθεια/μεταβλητότητα).

Στο παρακάτω παράδειγμα (Κ.. 4.1.4) συγκρίνουμε τη θερμοκρασία σε δύο πόλεις. Και οι δύο έχουν τον ίδιο μέσο όρο (20°C), αλλά η αίσθηση του καιρού είναι τελείως διαφορετική. Το Pandas υπολογίζει εξ ορισμού την "Δειγματική" τυπική απόκλιση (διαιρεί με n-1), που είναι και το σωστό για περιγραφική στατιστική δείγματος.

```
import pandas as pd

data = {
    'City_A': [19, 20, 20, 21, 20],       # Σταθερός καιρός
    'City_B': [5, 35, 0, 40, 20]          # Ακραίος καιρός
}

df = pd.DataFrame(data)

# 1. Υπολογισμός Μέσου Όρου (για επιβεβαίωση)
mean_vals = df.mean()
print(f"Μέση Θερμοκρασία A: {mean_vals['City_A']}")
print(f"Μέση Θερμοκρασία B: {mean_vals['City_B']}")
# Αποτέλεσμα:
# Μέση Θερμοκρασία A: 20.0
# Μέση Θερμοκρασία B: 20.0
# (Οι μέσοι όροι είναι ίδιοι, άρα δεν δείχνουν τη διαφορά)


# 2. Υπολογισμός Διακύμανσης (Variance) - .var()
var_a = df['City_A'].var()
var_b = df['City_B'].var()

print(f"Διακύμανση A: {var_a}")
# Αποτέλεσμα: Διακύμανση A: 0.5 (Πολύ μικρή)

print(f"Διακύμανση B: {var_b}")
# Αποτέλεσμα: Διακύμανση B: 312.5 (Τεράστια)


# 3. Υπολογισμός Τυπικής Απόκλισης (Std Deviation) - .std()
std_a = df['City_A'].std()
std_b = df['City_B'].std()
```

```
print(f"Τυπική Απόκλιση Α: {std_a:.2f}")
# Αποτέλεσμα: Τυπική Απόκλιση Α: 0.71
# Ερμηνεία: Στην πόλη Α, η θερμοκρασία παίζει συνήθως +/- 0.7 βαθμούς από το 20.

print(f"Τυπική Απόκλιση Β: {std_b:.2f}")
# Αποτέλεσμα: Τυπική Απόκλιση Β: 17.68
# Ερμηνεία: Στην πόλη Β, η θερμοκρασία μπορεί να απέχει κατά μέσο όρο 17.7 βαθμούς από το
```

Κ. 4.1.4 Υπολογισμός Διακύμανσης και Τυπικής Απόκλισης

Το **Εύρος** είναι το πιο απλό μέτρο διασποράς. Μας δείχνει την απόσταση μεταξύ της μικρότερης και της μεγαλύτερης τιμής. Δυστυχώς, το Pandas δεν έχει έτοιμη εντολή .range(), οπότε το υπολογίζουμε αφαιρώντας το ελάχιστο από το μέγιστο (Κ. 4.1.5).

```
import pandas as pd

data = {'Ηλικία': [23, 25, 28, 30, 45, 22, 50]}
df = pd.DataFrame(data)

# Υπολογισμός Εύρους
range_val = df['Ηλικία'].max() - df['Ηλικία'].min()

print(f"Μέγιστη τιμή: {df['Ηλικία'].max()}")
# Αποτέλεσμα: Μέγιστη τιμή: 50

print(f"Ελάχιστη τιμή: {df['Ηλικία'].min()}")
# Αποτέλεσμα: Ελάχιστη τιμή: 22

print(f"Εύρος (Range): {range_val}")
# Αποτέλεσμα: Εύρος (Range): 28
```

Κ. 4.1.5 Υπολογισμός Εύρους

Ο Συντελεστής Μεταβλητότητας (CV) είναι ένα κρίσιμο εργαλείο όταν θέλουμε να συγκρίνουμε τη μεταβλητότητα δύο διαφορετικών δεδομένων που έχουν διαφορετικές μονάδες μέτρησης ή πολύ διαφορετικούς μέσους όρους.

Παράδειγμα: Ποιος έχει μεγαλύτερη διακύμανση βάρους; Οι ελέφαντες ή τα ποντίκια;

Αν δούμε την Τυπική Απόκλιση, οι ελέφαντες θα έχουν τεράστια (π.χ. 500 κιλά) και τα ποντίκια μικρή (π.χ. 0.05 κιλά). Αυτό όμως είναι παραπλανητικό γιατί οι ελέφαντες είναι εξαρχής βαρύτεροι. Ο **Συντελεστής Μεταβλητότητας (CV)** λύνει το πρόβλημα διαιρώντας την τυπική απόκλιση με τον μέσο όρο.

Τύπος: $CV = \frac{s}{m} \times 100\%$,

όπου s είναι η τυπική απόκλιση και m ο μέσος όρος. Ας δούμε ένα παράδειγμα εφαρμογής του Συντελεστή Μεταβλητότητας με μετοχές (Κ. 4.1.6):

```
# Έστω δύο μετοχές
stock_A = [100, 102, 98, 105, 95]  # Υψηλή τιμή, μικρή διακύμανση
```

```
stock_B = [5, 6, 4, 7, 3]          # Χαμηλή τιμή, μεγάλη διακύμανση
αναλογικά

df_stocks = pd.DataFrame({'Stock_A': stock_A, 'Stock_B': stock_B})

# Τύπος: (Τυπική Απόκλιση / Μέσος Όρος) * 100
cv_A = (df_stocks['Stock_A'].std() / df_stocks['Stock_A'].mean()) * 100
cv_B = (df_stocks['Stock_B'].std() / df_stocks['Stock_B'].mean()) * 100

print(f"CV Stock A: {cv_A:.2f}%")
# Αποτέλεσμα: CV Stock A: 3.81%
# (Συμπέρασμα: Σταθερή μετοχή)

print(f"CV Stock B: {cv_B:.2f}%")
# Αποτέλεσμα: CV Stock B: 31.62%
# (Συμπέρασμα: Πολύ ευμετάβλητη/ριψοκίνδυνη μετοχή)
```

Κ. 4.1.6 Υπολογισμός Συντελεστή Μεταβλητότητας

Ερμηνεία: Αν το CV της Stock_B είναι μεγαλύτερο (π.χ. 30%) από της Stock_A (π.χ. 5%), τότε η Stock_B είναι πιο "ασταθής" ή "ριψοκίνδυνη", ασχέτως αν η τιμή της είναι μικρότερη.

Τα **Εκατοστημόρια και Τεταρτημόρια (Percentiles & Quartiles)** χωρίζουν τα (ταξινομημένα) δεδομένα μας σε τμήματα. Μας βοηθούν να απαντήσουμε σε ερωτήσεις όπως: "Είμαι στο top 10% των φοιτητών;" ή "Ποιο είναι το όριο για να μπω στη μεσαία τάξη;".

Το k-οστό εκατοστημόριο (P_k) είναι η τιμή κάτω από την οποία βρίσκεται το k% των παρατηρήσεων. Αν το παιδί σας είναι στο 90ο εκατοστημόριο ύψους, σημαίνει ότι είναι ψηλότερο από το 90% των παιδιών της ηλικίας του.

Είναι τρία συγκεκριμένα εκατοστημόρια που χωρίζουν τα δεδομένα σε 4 ίσα μέρη (25% το καθένα).

1. **1ο Τεταρτημόριο (Q1 - 25%):** Το 25% των δεδομένων είναι μικρότερο από αυτό.
2. **2ο Τεταρτημόριο (Q2 - 50%):** Είναι ακριβώς η Διάμεσος. Το 50% είναι μικρότερο και το 50% μεγαλύτερο.
3. **3ο Τεταρτημόριο (Q3 - 75%):** Το 75% των δεδομένων είναι μικρότερο από αυτό (και το top 25% μεγαλύτερο).

Η εντολή-κλειδί εδώ είναι η .quantile(). Δέχεται έναν δεκαδικό αριθμό από 0 έως 1 (π.χ. 0.25 για το 25%) (Κ. 4.1.7).

```
import pandas as pd

# Δεδομένα: Βαθμολογίες 11 φοιτητών σε ένα τεστ (ταξινομημένες για ευκολία)
scores_data = [40, 52, 55, 60, 65, 70, 75, 82, 85, 90, 95]
df = pd.DataFrame(scores_data, columns=['Score'])

# 1. Υπολογισμός Τεταρτημορίων (Quartiles)
```

```
# Q1: Το σημείο που αφήνει πίσω του το 25% των φοιτητών
Q1 = df['Score'].quantile(0.25)

# Q2: Η Διάμεσος
Q2 = df['Score'].quantile(0.50)

# Q3: Το σημείο που αφήνει πίσω του το 75% των φοιτητών
Q3 = df['Score'].quantile(0.75)

print(f"1ο Τεταρτημόριο (Q1): {Q1}")
# Αποτέλεσμα: 1ο Τεταρτημόριο (Q1): 57.5
# (Βρίσκεται ανάμεσα στο 55 και το 60)

print(f"Διάμεσος (Q2): {Q2}")
# Αποτέλεσμα: Διάμεσος (Q2): 70.0
# (Είναι η μεσαία τιμή της λίστας)

print(f"3ο Τεταρτημόριο (Q3): {Q3}")
# Αποτέλεσμα: 3ο Τεταρτημόριο (Q3): 83.5
# (Βρίσκεται ανάμεσα στο 82 και το 85)


# 2. Υπολογισμός Ειδικού Εκατοστημορίου (Percentile)
# Θέλουμε να βρούμε το όριο για το Top 10% (δηλαδή το 90ο εκατοστημόριο)
top_10_threshold = df['Score'].quantile(0.90)

print(f"Όριο για το Top 10% (P90): {top_10_threshold}")
# Αποτέλεσμα: Όριο για το Top 10% (P90): 90.0
# Ερμηνεία: Όποιος έχει πάνω από 90, ανήκει στο κορυφαίο 10% της τάξης.
```

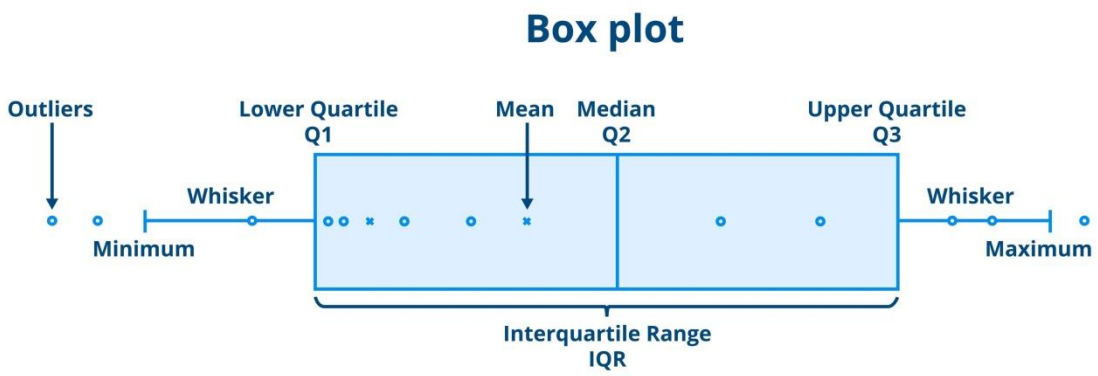
Κ. 4.1.7 Υπολογισμός Τεταρτημορίων

Μπορούμε επίσης να περάσουμε μια λίστα μέσα στην quantile() για να πάρουμε όλες τις τιμές ταυτόχρονα (Κ. 4.1.8).

```
# Υπολογισμός Q1, Q2, Q3 με μία εντολή
quartiles = df['Score'].quantile([0.25, 0.50, 0.75])

print(quartiles)
# Αποτέλεσμα:
# 0.25    57.5
# 0.50    70.0
# 0.75    83.5
# Name: Score, dtype: float64
```

Κ. 4.1.8 Υπολογισμός Τεταρτημορίων με λίστα τιμών

Το Εύρος (Range) που είδαμε νωρίτερα (Max - Min) έχει ένα μεγάλο μειονέκτημα: επηρεάζεται τρομερά από τις ακραίες τιμές. Αν σε μια τάξη όλοι πήραν 15-18 και ένας πήρε 0, το Εύρος θα είναι τεράστιο (18), δίνοντας λάθος εντύπωση. Το **Ενδοτεταρτημοριακό Εύρος (IQR)** λύνει αυτό το πρόβλημα εστιάζοντας στο μεσαίο 50% των δεδομένων. Το IQR δίδεται από τον τύπο IQR = Q3 - Q1. Στο σχήμα 4.1 απεικονίζεται το IQR με ένα box plot, το οποίον θα το εξηγήσουμε με λεπτομέρεια στο Κεφάλαιο 4.2.



Σ. 4.1 Απεικόνιση IQR

Στον κώδικα Κ. 4.1.9 Το μεσαίο 50% των υπαλλήλων έχει διαφορά μισθού μόλις 6.000€, παρόλο που υπάρχει κάποιος που παίρνει 150.000€. Το IQR μας δίνει μια πιο ρεαλιστική εικόνα της "τυπικής διακύμανσης" από το απλό Εύρος.

```
import pandas as pd

# Δεδομένα: Μισθοί σε μια εταιρεία (σε χιλιάδες €)
# Παρατηρήστε ότι το 150 είναι μια ακραία τιμή (το αφεντικό!)
salaries = [20, 22, 24, 25, 26, 28, 30, 35, 150]
df = pd.DataFrame(salaries, columns=['Salary'])

# Υπολογισμός Τεταρτημορίων
# Το 0.25 αντιστοιχεί στο 25% (Q1) και το 0.75 στο 75% (Q3)
Q1 = df['Salary'].quantile(0.25)
Q3 = df['Salary'].quantile(0.75)

# Υπολογισμός IQR
IQR = Q3 - Q1

print(f"1ο Τεταρτημόριο (Q1): {Q1}")
# Αποτέλεσμα: 1ο Τεταρτημόριο (Q1): 24.0

print(f"3ο Τεταρτημόριο (Q3): {Q3}")
# Αποτέλεσμα: 3ο Τεταρτημόριο (Q3): 30.0

print(f"Ενδοτεταρτημοριακό Εύρος (IQR): {IQR}")
# Αποτέλεσμα: Ενδοτεταρτημοριακό Εύρος (IQR): 6.0
```

Κ. 4.1.9 Υπολογισμός IQR

Στην επιστήμη δεδομένων, ο πιο κοινός αλγόριθμος για να βρούμε "αυτόματα" ποια τιμή είναι ακράια, είναι ο κανόνας του 1.5 x IQR.

Ορίζουμε δύο φράχτες (fences):

1. **Κάτω Όριο (Lower Bound):** Q1 - 1.5 x IQR
2. **Άνω Όριο (Upper Bound):** Q3 + 1.5 x IQR

Οτιδήποτε βρίσκεται έξω από αυτούς τους φράχτες, θεωρείται ακραία τιμή (Κ. 4.1.10).

```
# Συνέχεια από το προηγούμενο παράδειγμα...

# Υπολογισμός ορίων
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

print(f"Κάτω Όριο: {lower_bound}")
# Αποτέλεσμα: Κάτω Όριο: 15.0

print(f"Άνω Όριο: {upper_bound}")
# Αποτέλεσμα: Άνω Όριο: 39.0

# Εντοπισμός των Outliers
# Ψάχνουμε εγγραφές μικρότερες του κάτω ορίου ή μεγαλύτερες του άνω
outliers = df[(df['Salary'] < lower_bound) | (df['Salary'] > upper_bound)]

print("Εντοπίστηκαν οι εξής ακραίες τιμές:")
print(outliers)
# Αποτέλεσμα:
#    Salary
# 8     150
```

Κ. 4.1.10 Υπολογισμός ακραίων τιμών με χρήση IQR

Η **Λοξότητα (Skewness)** μετράει την ασυμμετρία της κατανομής γύρω από τον μέσο όρο. Μας δείχνει προς τα πού "τραβιέται" η ουρά των δεδομένων.

- **Λοξότητα περίπου στο 0 (Symmetrical):** Κανονική κατανομή. Τα δεδομένα μοιράζονται συμμετρικά (Μέσος = Διάμεσος).
- **Θετική Λοξότητα (Positive / Right Skewed):** Η ουρά εκτείνεται προς τα δεξιά (προς τις μεγάλες τιμές).
- **Αρνητική Λοξότητα (Negative / Left Skewed):** Η ουρά εκτείνεται προς τα αριστερά (προς τις μικρές τιμές).

Το παρακάτω παράδειγμα υπολογίζει τη Λοξότητα (Κ. 4.1.11) δύο συνόλων.

```
import pandas as pd

# Δεδομένα με Θετική Λοξότητα (εισοδήματα)
# Οι περισσότεροι παίρνουν λίγα, ένας παίρνει πολλά (100)
data_pos = [1, 2, 3, 4, 5, 100]
df_pos = pd.DataFrame(data_pos, columns=['Values'])

# Δεδομένα με Αρνητική Λοξότητα
# Οι περισσότεροι είναι ψηλά (95-100), ένας είναι χαμηλά (5)
data_neg = [5, 95, 96, 97, 98, 99, 100]
df_neg = pd.DataFrame(data_neg, columns=['Values'])

# Υπολογισμός Λοξότητας (.skew())
skew_pos = df_pos['Values'].skew()
```

```
skew_neg = df_neg['Values'].skew()

print(f"Θετική Λοξότητα: {skew_pos:.2f}")
# Αποτέλεσμα: Θετική Λοξότητα: 2.45
# (Η τιμή > 0 επιβεβαιώνει τη δεξιά ουρά)

print(f"Αρνητική Λοξότητα: {skew_neg:.2f}")
# Αποτέλεσμα: Αρνητική Λοξότητα: -2.64
# (Η τιμή < 0 επιβεβαιώνει την αριστερή ουρά)
```

Κ. 4.1.11 Υπολογισμός Λοξότητας

Η **Κύρτωση (Kurtosis)** μετράει την "αιχμηρότητα" της κορυφής και το "βάρος" των ουρών (πόσο συχνά εμφανίζονται ακραίες τιμές) σε σχέση με την κανονική κατανομή. Η Λεπτοκυρτωμένη εχει ψηλή, λεπτή κορυφή και παχιές ουρές (heavy tails). Εδώ υπάρχει μεγάλη πιθανότητα για ακραίες τιμές (outliers). Στα χρηματοοικονομικά αυτό σημαίνει "ρίσκο". Η Μεσοκυρτωμένη ομοιάζει με την κανονική κατανομή, ενώ η Πλατυκυρτωμένη έχει πλατιά, χαμηλή κορυφή και λεπτές ουρές. Εδώ τα δεδομένα είναι πιο ομοιόμορφα κατανεμημένα, λιγότερες ακραίες τιμές.

Το παρακάτω παράδειγμα υπολογίζει την Κύρτωση (Κ. 4.1.12) δύο συνόλων.

```
import pandas as pd

# Δεδομένα Λεπτοκυρτωμένα (Leptokurtic)
# Πολλά δεδομένα στο κέντρο (10-10.2) και ξαφνικά ακραίες τιμές (0, 20)
lepto_data = [0, 10, 10, 10, 10.1, 10.1, 10.2, 10.2, 20]
df_lepto = pd.DataFrame(lepto_data, columns=['Values'])

# Δεδομένα Πλατυκυρτωμένα (Platykurtic)
# Τα δεδομένα είναι απλωμένα ομοιόμορφα (Uniform distribution)
platy_data = [1, 2, 3, 4, 5, 6, 7, 8, 9]
df_platy = pd.DataFrame(platy_data, columns=['Values'])

# Υπολογισμός Κύρτωσης (.kurt())
kurt_lepto = df_lepto['Values'].kurt()
kurt_platy = df_platy['Values'].kurt()

print(f"Κύρτωση Λεπτοκυρτωμένης: {kurt_lepto:.2f}")
# Αποτέλεσμα: Κύρτωση Λεπτοκυρτωμένης: 3.42
# (Τιμή > 0: "Μυτερή" κατανομή με heavy tails)

print(f"Κύρτωση Πλατυκυρτωμένης: {kurt_platy:.2f}")
# Αποτέλεσμα: Κύρτωση Πλατυκυρτωμένης: -1.20
# (Τιμή < 0: "Πλατιά" κατανομή, χωρίς πολλές εκπλήξεις στις άκρες)
```

Κ. 4.1.12 Υπολογισμός Κύρτωσης

4.1.2. Ομαδοποίηση Δεδομένων

Η ομαδοποίηση μας επιτρέπει να τεμαχίσουμε τα δεδομένα μας σε κάδους (buckets) με βάση κάποια κατηγορία, να υπολογίσουμε στατιστικά στοιχεία για κάθε κάδο ξεχωριστά και να ενώσουμε τα αποτελέσματα ξανά. Η κεντρική εντολή στο Pandas είναι η: groupby(). Εδώ εφαρμόζεται η φιλοσοφία **Split-Apply-Combine**:

1. **Split (Διαχωρισμός)**: Τα δεδομένα χωρίζονται σε ομάδες με βάση τα κριτήρια που ορίζουμε (π.χ. ανά "Τμήμα").
2. **Apply (Εφαρμογή):** Μια συνάρτηση εφαρμόζεται σε κάθε ομάδα ανεξάρτητα (π.χ. υπολογισμός μέσου όρου, άθροισμα, ή τυπική απόκλιση).
3. **Combine (Συνδυασμός):** Τα αποτελέσματα ενώνονται ξανά σε μια νέα δομή δεδομένων (συνήθως Series ή DataFrame) για να παρουσιαστούν συνολικά.

Για να κατανοήσουμε τις έννοιες, ας υποθέσουμε ότι έχουμε ένα DataFrame με πωλήσεις καταστημάτων (Κ4.1.13):

```
import pandas as pd
import numpy as np

# Δημιουργία εικονικών δεδομένων
data = {
    'Κατάστημα': ['Αθήνα', 'Αθήνα', 'Θεσ/νίκη', 'Θεσ/νίκη', 'Πάτρα',
'Αθήνα', 'Πάτρα', 'Θεσ/νίκη'],
    'Προϊόν': ['Laptop', 'Mouse', 'Laptop', 'Keyboard', 'Mouse',
'Keyboard', 'Laptop', 'Mouse'],
    'Πωλητής': ['Γιώργος', 'Μαρία', 'Νίκος', 'Ελένη', 'Κώστας', 'Γιώργος',
'Κώστας', 'Νίκος'],
    'Πωλήσεις': [1200, 40, 1100, 80, 45, 90, 1150, 35],
    'Τεμάχια': [2, 5, 1, 10, 4, 8, 2, 3]
}

df = pd.DataFrame(data)
print(df)
```

Κ. 4.1.13 DataFrame με πωλήσεις καταστημάτων.

Η πιο απλή μορφή είναι να ομαδοποιήσουμε βάσει μίας στήλης και να υπολογίσουμε τον μέσο όρο (ή άθροισμα) για τις υπόλοιπες αριθμητικές στήλες.

Σύνταξη: `df.groupby('Στήλη_Ομαδοποίησης')['Αριθμητική_Στήλη'].συνάρτηση()`

Παράδειγμα: Ποιος είναι ο μέσος όρος πωλήσεων ανά πόλη (Κ. 4.1.14);

```
# Υπολογισμός μέσων πωλήσεων ανά Κατάστημα
mean_sales = df.groupby('Κατάστημα')['Πωλήσεις'].mean()
print(mean_sales)
```

Κ. 4.1.14 Μέσος όρος ανα πόλη

Ερμηνεία: Το Pandas χώρισε τις γραμμές σε "Αθήνα", "Θεσ/νίκη", "Πάτρα", βρήκε τον μέσο όρο για την κάθε μία και επέστρεψε το αποτέλεσμα. Μπορούμε να ομαδοποιήσουμε βάσει περισσότερων της μιας κατηγοριών. Αυτό δημιουργεί ένα MultiIndex (ιεραρχικό ευρετήριο).

Παράδειγμα: Πόσα τεμάχια πουλήθηκαν ανά Κατάστημα ΚΑΙ ανά Προϊόν (Κ. 4.1.15);

```
# Ομαδοποίηση με λίστα στηλών
sales_by_store_product =
df.groupby(['Κατάστημα','Προϊόν'])['Τεμάχια'].sum()
```

Κ. 4.1.15 Πώληση τεμαχίων ανά Κατάστημα ΚΑΙ ανά Προϊόν

Στην περιγραφική στατιστική, συχνά θέλουμε να βλέπουμε ταυτόχρονα τον μέσο όρο, τη διακύμανση και το πλήθος. Η μέθοδος aggregate() ή συντομογραφικά agg() είναι ιδανική για αυτό (Κ. 4.1.16). Μπορούμε να περάσουμε:

- Μια λίστα συναρτήσεων (π.χ. ['mean', 'std']).
- Ένα λεξικό (dictionary) για να ορίσουμε διαφορετικές πράξεις σε διαφορετικές στήλες.

```
# Εφαρμογή πολλών συναρτήσεων ταυτόχρονα στις Πωλήσεις
stats = df.groupby('Κατάστημα')['Πωλήσεις'].agg(['mean', 'median', 'std',
'max', 'count'])

# Εξειδικευμένη ομαδοποίηση: Άθροισμα στις Πωλήσεις, Μέσος όρος στα Τεμάχια
custom_agg = df.groupby('Κατάστημα').agg({
    'Πωλήσεις': 'sum',
    'Τεμάχια': 'mean'
})
```

Κ. *4.1.16 Μέθοδος agg*

Εκτός από την απλή σύνοψη (aggregation), υπάρχουν δύο πολύ σημαντικές λειτουργίες για την ανάλυση δεδομένων. Το φιλτράρισμα ομάδων (filter) μας επιτρέπει να κρατήσουμε ή να πετάξουμε ολόκληρες ομάδες βάσει ενός κριτηρίου (Κ. 4.1.17).

Παράδειγμα: Θέλουμε να κρατήσουμε δεδομένα μόνο από καταστήματα που έχουν κάνει συνολικό τζίρο άνω των 2000€.

```
# Η lambda function ελέγχει το άθροισμα της κάθε ομάδας (x)
high_performing_stores = df.groupby('Κατάστημα').filter(lambda x:
x['Πωλήσεις'].sum() > 2000)
```

Κ. 4.1.17 Μέθοδος filter

Η εντολή transform είναι χρήσιμη όταν θέλουμε να διατηρήσουμε το αρχικό σχήμα του DataFrame αλλά να προσθέσουμε μια στήλη με στατιστικά της ομάδας.

Παράδειγμα: Θέλουμε να δούμε τι ποσοστό του τζίρου του καταστήματος αντιπροσωπεύει η κάθε πώληση (Κ. 4.1.18).

```
# Υπολογίζουμε το άθροισμα πωλήσεων ΤΟΥ ΚΑΤΑΣΤΗΜΑΤΟΣ σε κάθε γραμμή
store_total = df.groupby('Κατάστημα')['Πωλήσεις'].transform('sum')

# Δημιουργούμε νέα στήλη
df['Ποσοστό_Τζίρου'] = (df['Πωλήσεις'] / store_total) * 100

# Τώρα βλέπουμε για κάθε πώληση, πόσο σημαντική ήταν για το κατάστημα
print(df.sort_values(by='Κατάστημα'))
```

Κ. 41.18 Μέθοδος transform

Σημείωση: Το transform επιστρέφει μια στήλη με το ίδιο μήκος με το αρχικό DataFrame, επιτρέποντας απευθείας πράξεις.

Μια εναλλακτική, πιο φιλική προς το μάτι (spreadsheet-style) προσέγγιση της ομαδοποίησης είναι τα Pivot Tables (Κ. 4.1.19).

```
# Δημιουργία πίνακα με γραμμές τα Καταστήματα, στήλες τα Προϊόντα και τιμές το άθροισμα Πωλήσεων
pivot = df.pivot_table(values='Πωλήσεις',
                       index='Κατάστημα',
                       columns='Προϊόν',
                       aggfunc='sum',
                       fill_value=0) # Βάζει 0 όπου δεν υπάρχει πώληση
```

Κ. 4.1.19 Δημιουργία Pivot Table

4.1.2.1 Case Study: Ανάλυση Επιδόσεων Φοιτητών

Σε αυτό το σενάριο, έχουμε δεδομένα από μια πανεπιστημιακή σχολή. Στόχος μας είναι να χρησιμοποιήσουμε την Περιγραφική Στατιστική για να κατανοήσουμε την απόδοση των φοιτητών, τη συνέπειά τους και τη σχέση των ωρών μελέτης με τον βαθμό τους. Θα κατασκευάσουμε ένα DataFrame με 100 φοιτητές (Κ. 4.1.20) το οποίο περιλαμβάνει:

- **Φύλο (Gender):** Κατηγορική μεταβλητή.
- **Τμήμα (Major):** CS (Πληροφορική), Math (Μαθηματικά), Physics (Φυσική).
- **Ώρες Μελέτης (Study_Hours):** Ποσοτική μεταβλητή (ώρες/εβδομάδα).
- **Τελικός Βαθμός (Grade):** Ποσοτική μεταβλητή (0-10).

```
import pandas as pd
import numpy as np

# Ρύθμιση seed για επαναληψιμότητα
np.random.seed(42)

# Δημιουργία δεδομένων
n = 100
data = {
    'Student_ID': range(1, n + 1),
    'Gender': np.random.choice(['Male', 'Female'], n),
    'Major': np.random.choice(['CS', 'Math', 'Physics'], n),
    'Study_Hours': np.random.randint(1, 30, n),  # Ώρες μελέτης από 1 έως 30
    'Grade': np.round(np.random.normal(6.5, 1.5, n), 1) # Κανονική κατανομή με μ=6.5, σ=1.5
}

df = pd.DataFrame(data)

# Διόρθωση βαθμών εκτός ορίων (0-10)
df['Grade'] = df['Grade'].clip(0, 10)
```

Κ. 4.1.20 DataFrame με 100 φοιτητές

Πριν σπάσουμε τα δεδομένα σε ομάδες, κοιτάμε τη "μεγάλη εικόνα" για να εντοπίσουμε κεντρικές τάσεις και πιθανά λάθη με την εντολή describe του Dataframe (print(df.describe()).

Τι παρατηρούμε εδώ:

1. **Mean vs Median (50%):** Αν ο Μέσος Όρος (mean) του Βαθμού είναι πολύ κοντά στη Διάμεσο (50%), η κατανομή είναι πιθανότατα συμμετρική. Αν έχουν μεγάλη διαφορά, έχουμε λοξότητα.

2. **Std (Τυπική Απόκλιση):** Μας δείχνει πόσο "απλωμένοι" είναι οι βαθμοί. Μια μεγάλη τυπική απόκλιση σημαίνει μεγάλες ανισότητες στην απόδοση.

3. **Min/Max:** Ελέγχουμε αν υπάρχουν παράλογες τιμές (π.χ. βαθμός -1 ή 15).

Το πιο ενδιαφέρον ερώτημα στο παράδειγμά μας είανι: «Ποιο τμήμα έχει τους καλύτερους φοιτητές και ποιο τους πιο συνεπείς;». Θα χρησιμοποιήσουμε την groupby μαζί με την agg για να πάρουμε ταυτόχρονα τον μέσο όρο (επίδοση) και την τυπική απόκλιση (συνέπεια) (Κ. 4.1.21).

```
dept_stats = df.groupby('Major')['Grade'].agg(['mean', 'median', 'std',
'count'])
print(dept_stats)
```

Κ. 4.1.21 Χρήση groupby μαζί με την agg

- Αν το CS έχει υψηλότερο mean αλλά και υψηλότερο std από το Math, σημαίνει ότι το CS έχει γενικά καλύτερους βαθμούς, αλλά έχει και μεγαλύτερο χάσμα μεταξύ των αριστούχων και των αδύναμων φοιτητών.

- Το τμήμα με το χαμηλότερο std είναι το πιο "ομοιογενές" (οι φοιτητές έχουν παρόμοιες επιδόσεις).

Οι "Ώρες Μελέτης" είναι συνεχής μεταβλητή. Για να βγάλουμε συμπέρασμα, βολεύει να τις ομαδοποιήσουμε σε κατηγορίες (Low, Medium, High). Αυτό λέγεται Binning (Κ. 4.1.22).

```
# Δημιουργία κατηγοριών μελέτης
bins = [0, 10, 20, 30]
labels = ['Λίγο (0-10)', 'Μέτρια (10-20)', 'Πολύ (20-30)']

df['Effort'] = pd.cut(df['Study_Hours'], bins=bins, labels=labels)

# Ομαδοποίηση βάσει προσπάθειας
effort_grade = df.groupby('Effort')['Grade'].mean()
print(effort_grade)
```

Κ. 4.1.22 Binning

Εδώ περιμένουμε (λογικά) να δούμε ότι όσο αυξάνεται η κατηγορία προσπάθειας, αυξάνεται και ο μέσος βαθμός. Αυτό επιβεβαιώνει ποσοτικά τη σχέση αιτίου-αποτελέσματος.

Ας φτιάξουμε τώρα έναν πίνακα (Pivot) που δείχνει τον μέσο βαθμό ανά Τμήμα (γραμμές) και ανά Φύλο (στήλες) (Κ. 4.1.23).

```
pivot = df.pivot_table(values='Grade',
                       index='Major',
                       columns='Gender',
                       aggfunc='mean')
print(pivot)
```
Κ. 4.1.23 Pivot Table

Αυτός ο πίνακας μας επιτρέπει να κάνουμε συγκρίσεις πολλαπλών επιπέδων με μια ματιά (π.χ. "Πώς τα πάνε οι γυναίκες στο τμήμα Φυσικής σε σχέση με τους άνδρες στο τμήμα Πληροφορικής;").

Στο ερώτημα "Ποιοι είναι οι φοιτητές που αποκλίνουν σημαντικά; ", θα ορίσουμε ως ακραίες τιμές (Outliers) τους βαθμούς πάνω ή κάτω από 2 τυπικές αποκλίσεις από τον μέσο όρο (Κ. 4.1.24).

```
mean_grade = df['Grade'].mean()
std_grade = df['Grade'].std()

# Όριο: Mean ± 2 * Std
upper_bound = mean_grade + 2 * std_grade
lower_bound = mean_grade - 2 * std_grade

outliers = df[(df['Grade'] > upper_bound) | (df['Grade'] < lower_bound)]

print(f"Όρια: {lower_bound:.2f} - {upper_bound:.2f}")
print("Φοιτητές με ακραίες επιδόσεις:\n", outliers[['Student_ID', 'Major',
'Grade']])
```
Κ. 4.1.24 Εύρεση ακράιων τιμών

4.1.3. Η Βιβλιοθήκη statistics

Μέχρι τώρα είδαμε πώς να κάνουμε στατιστική ανάλυση με το Pandas. Ωστόσο, η Python έρχεται με μια "πρότυπη βιβλιοθήκη" (Standard Library) που περιλαμβάνει το module statistics.

Είναι η ιδανική λύση όταν:

1. Δουλεύουμε με απλές λίστες δεδομένων και όχι με πολύπλοκα DataFrames.
2. Δεν θέλουμε ή δεν μπορούμε να εγκαταστήσουμε βαριές βιβλιοθήκες όπως το Pandas/NumPy.
3. Χρειαζόμαστε υψηλή ακρίβεια σε δεκαδικά ψηφία (η statistics διαχειρίζεται καλύτερα τα σφάλματα στρογγυλοποίησης από τα floats).

Ας δούμε πώς υπολογίζουμε τον μέσο όρο, την διάμεσο και την επιρατούσα τιμή (Κ. 4.1.25).

```
import statistics

# Ένα απλό σύνολο δεδομένων (λίστα)
data = [15, 18, 20, 20, 22, 25, 35]

# 1. Μέσος Όρος (Mean)
mean_val = statistics.mean(data)
print(f"Μέσος Όρος: {mean_val}")
```

```
# 2. Διάμεσος (Median)
median_val = statistics.median(data)
print(f"Διάμεσος: {median_val}")

# 3. Επικρατούσα Τιμή (Mode)
# Προσοχή: Αν υπάρχουν πολλές, επιστρέφει την πρώτη που συναντά (σε παλιότερες εκδόσεις)
# ή πετάει σφάλμα. Στις νέες εκδόσεις (Python 3.8+) υπάρχει το multimode.
mode_val = statistics.mode(data)
print(f"Επικρατούσα Τιμή: {mode_val}")
```

Κ. 4.1.25 Εύρεση μέσου όρου, διάμεσου και επικρατούσας τιμής

Η βιβλιοθήκη κάνει διάκριση μεταξύ δείγματος (Sample) και πληθυσμού (Population). Αυτό είναι σημαντικό γιατί ο μαθηματικός τύπος της τυπικής απόκλισης διαφέρει ελαφρώς, δηλαδή διαιρούμε με n-1 στο δείγμα και με n στον πληθυσμό, όπου n ο αριθμός των δειγμάτων. Στην περιγραφική στατιστική συνήθως δουλεύουμε με δείγματα, άρα χρησιμοποιούμε τις εντολές stdev για το υπολογισμό της τυπικής απόκλισης και variance για τον υπολογισμό της διακύμανσης (Κ. 4.1.26):

```
# 1. Τυπική Απόκλιση Δείγματος (Sample Standard Deviation)
st_dev = statistics.stdev(data)
print(f"Τυπική Απόκλιση: {st_dev:.2f}")

# 2. Διακύμανση Δείγματος (Sample Variance)
variance = statistics.variance(data)
print(f"Διακύμανση: {variance:.2f}")
```

Κ. 4.1.26 Υπολογισμός τυπικής απόκλισης και διακύμανσης

Σημείωση: Αν έχουμε δεδομένα για ολόκληρο τον πληθυσμό, χρησιμοποιούμε τις pstdev() και pvariance().

Συχνά μια κατανομή είναι "δικόρυφη" (bimodal), δηλαδή έχει δύο αριθμούς που εμφανίζονται εξίσου συχνά. Η κλασική mode() μπορεί να μπερδευτεί. Η multimode() λύνει αυτό το πρόβλημα επιστρέφοντας λίστα (Κ. 4.1.27).

```
bimodal_data = [1, 2, 2, 3, 4, 4, 5]

# Επιστρέφει λίστα με όλες τις επικρατούσες τιμές
modes = statistics.multimode(bimodal_data)
print(f"Επικρατούσες Τιμές: {modes}")  # Θα τυπώσει [2, 4]
```

Κ. 4.1.27 Χρήση της mutimode

4.2. Περιγραφική Στατιστική: Οπτική Προσέγγιση

Αν η Ποσοτική Προσέγγιση είναι η "λογική" της Στατιστικής, η Οπτική Προσέγγιση είναι η "διαίσθησή" της. Συχνά, οι αριθμητικοί δείκτες (Μέσος Όρος, Τυπική Απόκλιση) δεν αρκούν για να περιγράψουν την πραγματική εικόνα. Ένα κλασικό παράδειγμα είναι το Anscombe's Quartet: τέσσερα διαφορετικά σύνολα δεδομένων που έχουν ακριβώς τον ίδιο μέσο όρο και διακύμανση, αλλά αν τα σχεδιάσουμε σε γράφημα, εμφανίζουν εντελώς διαφορετική συμπεριφορά.

Η Οπτικοποίηση Δεδομένων (Data Visualization) μας επιτρέπει:

1. Να αντιληφθούμε την κατανομή και το σχήμα των δεδομένων.
2. Να εντοπίσουμε μοτίβα και συσχετίσεις που κρύβονται στους πίνακες.
3. Να ανακαλύψουμε ακραίες τιμές (outliers) και σφάλματα στα δεδομένα.

Στο οικοσύστημα της Python, έχουμε τρία βασικά εργαλεία, το καθένα με διαφορετικό ρόλο:

1. **Matplotlib (matplotlib.pyplot).** *Είναι η "μητέρα" όλων των γραφικών βιβλιοθηκών στην Python.Η βιβλιοθήκη είναι εξαιρετικά ευέλικτη και επιτρέπει τον έλεγχο κάθε λεπτομέρειας (pixel-perfect). Απαιτεί όμως αρκετές γραμμές κώδικα για να γίνει ένα γράφημα όμορφο και παρουσιάσιμο.*
2. **Seaborn (seaborn).** *Είναι χτισμένη "πάνω" στη Matplotlib, αλλά σχεδιασμένη ειδικά για στατιστική ανάλυση. Παράγει πανέμορφα χρώματα, συνεργάζεται άψογα με το Pandas, δημιουργεί πολύπλοκα στατιστικά γραφήματα με μία εντολή.*
3. **Plotly (plotly.express).** *Η βιβλιοθήκη της νέας εποχής που προσθέτει τη διαδραστικότητα (interactivity). Τα γραφήματα δεν είναι απλές εικόνες. Μπορούμε να κάνουμε Zoom, Pan, και Hover (να βλέπουμε τιμές περνώντας το ποντίκι). Η εν λόγω βιβλιοθήκη είναι ιδανική για Dashboards, παρουσιάσεις σε οθόνη και εξερεύνηση δεδομένων (Exploaratory Data Analysis) για τον εντοπισμό συγκεκριμένων εγγραφών.*

Στο κεφάλαιο αυτό θα μάθουμε να δημιουργούμε και να ερμηνεύουμε:

1. **Γραμμικά Διαγράμματα (Line Plots):** Για την απεικόνιση τάσεων και μεταβολών στο χρόνο (Time Series).
2. **Ιστογράμματα (Histograms):** Για την κατανόηση της κατανομής.

3. **Θηκογράμματα (Boxplots):** Για τη σύγκριση κατανομών και τον εντοπισμό Outliers.
4. **Ραβδογράμματα (Bar Plots):** Για την ανάλυση κατηγορικών δεδομένων.
5. **Διαγράμματα Διασποράς (Scatter Plots)**: Για τη διερεύνηση σχέσεων μεταξύ δύο μεταβλητών.
6. **Θερμικά Διαγράμματα Συσχέτισης:** Για την οπτικοποίηση πινάκων συσχέτισης.

4.2.1. Εισαγωγή στα Εργαλεία Οπτικοποίησης

Πριν ξεκινήσουμε την ανάλυση δεδομένων, πρέπει να προετοιμάσουμε το περιβάλλον μας. Στην Python δεν υπάρχει "ένας σωστός τρόπος" για να φτιάξεις ένα γράφημα. Υπάρχουν τρία κύρια εργαλεία, το καθένα με τη δική του φιλοσοφία. Σε αυτή την ενότητα θα φορτώσουμε τις βιβλιοθήκες και θα δημιουργήσουμε το ίδιο ακριβώς γράφημα και με τους τρεις τρόπους για να κατανοήσουμε τις διαφορές. Ο κώδικας Κ. 4.2.1 είναι το εισαγωγικό κομμάτι κώδικα που θα τρέχουμε πάντα στην αρχή κάθε σημειωματάριου για την εν λόγω ενότητα.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# Ρύθμιση του Seaborn για πιο όμορφα (modern) γραφήματα από προεπιλογή
sns.set_theme(style="whitegrid")

# Δημιουργία ενός απλού DataFrame για τα παραδείγματα
# Δεδομένα: Πωλήσεις ενός καταστήματος για 5 ημέρες
df = pd.DataFrame({
    'Ημέρα': ['Δευ', 'Τρι', 'Τετ', 'Πεμ', 'Παρ'],
    'Πωλήσεις': [100, 150, 130, 200, 180]
})
```

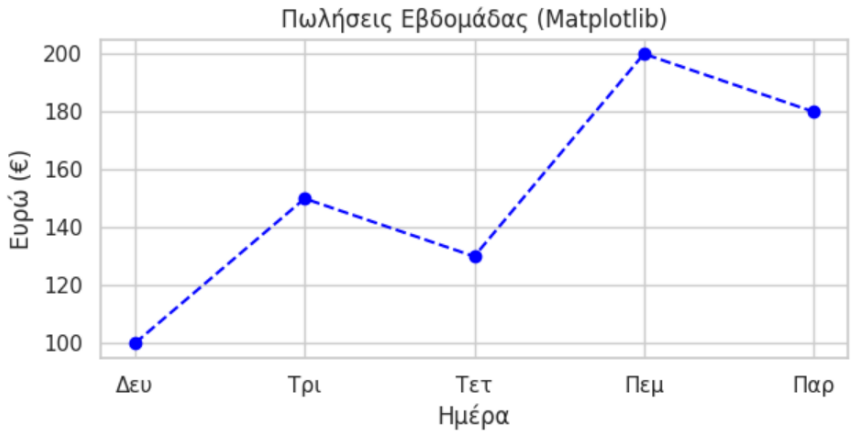
Κ. 4.2.1 Εισαγωγικό κομμάτι κώδικα για τα γραμμικά διαγράμματα

Χρησιμοποιώντας την βιβλιοθήκη matplotlib είναι η πιο κλασική μέθοδος για κατασκευή διαγραμμάτων. Παρατηρήστε στον κώδικα Κ.4.2.2, όπου δημιουργούμε ένα **γραμμικό διάγραμμα**, ότι πρέπει να ορίσουμε χειροκίνητα τους τίτλους και τις ετικέτες.

```
# 1. Δημιουργία του χώρου (Figure)
plt.figure(figsize=(8, 4))

# 2. Σχεδίαση (Plotting)
plt.plot(df['Ημέρα'], df['Πωλήσεις'], color='blue', marker='o',
linestyle='--')

# 3. Μορφοποίηση (Formatting)
plt.title('Πωλήσεις Εβδομάδας (Matplotlib)')
plt.xlabel('Ημέρα')
plt.ylabel('Ευρώ (€)')
```

```
# 4. Εμφάνιση
plt.show()
```

Κ. 4.2.2 Γραμμικό Διάγραμμα με matplotlib

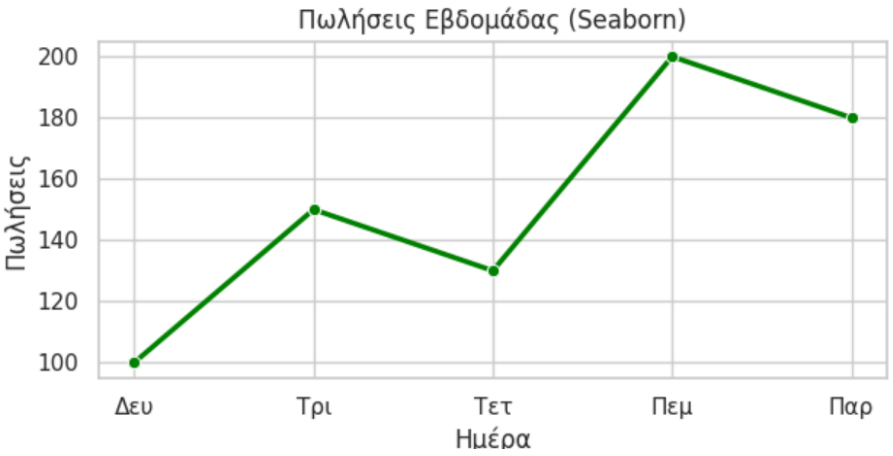


Ε. 4.2.2 Γραμμικό Διάγραμμα με matplotlib

Το διάγραμμα είναι ένα απλό, στατικό γράφημα. Οι γραμμές είναι λεπτές, το φόντο λευκό και η αισθητική θυμίζει κλασικά επιστημονικά διαγράμματα. Αντίθετα με τη βιβλιοθήκη seaborn (Κ. 4.2.3) χρησιμοποιούμε το DataFrame απευθείας. Ο κώδικας είναι πιο "καθαρός" και το αποτέλεσμα πιο μοντέρνο χωρίς κόπο.

```
# Δεν χρειάζεται να ορίσουμε figure size αν μας καλύπτει το default,
# αλλά μπορούμε αν θέλουμε:
plt.figure(figsize=(8, 4))

# Η εντολή lineplot κάνει τη δουλειά
sns.lineplot(data=df, x='Ημέρα', y='Πωλήσεις', marker='o', color='green',
linewidth=2.5)

plt.title('Πωλήσεις Εβδομάδας (Seaborn)')
plt.show()
```

Κ. 4.2.3 Γραμμικό Διάγραμμα με seaborn

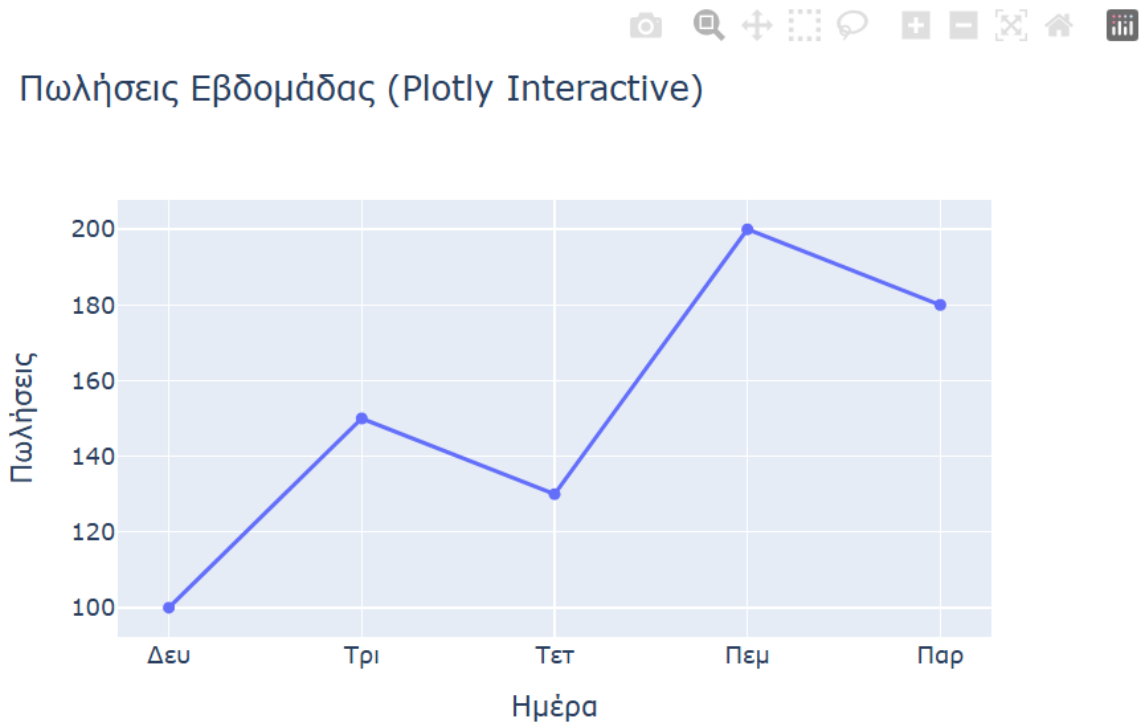


Ε. 4.2.3 Γραμμικό Διάγραμμα με seaborn

Τώρα έχουμε ένα πιο "ραφινάτο" γράφημα. Υπάρχει πλέγμα (grid) στο φόντο που βοηθάει στην ανάγνωση των τιμών, η γραμματοσειρά είναι πιο καθαρή και τα χρώματα πιο απαλά. Με το plotly η διαφορά συγκριτικα τις δύο προηγούμενες προσεγγίσεις είναι μεγάλη. Ο κώδικας Κ. 4.2.4 δημιουργεί ένα αντικείμενο fig που είναι ένα διαδραστικό HTML widget.

```
# Το Plotly Express (px) είναι σχεδιασμένο για ελάχιστο κώδικα
fig = px.line(df,
              x='Ημέρα',
              y='Πωλήσεις',
              title='Πωλήσεις Εβδομάδας (Plotly Interactive)',
              markers=True) # Ενεργοποίηση κουκκίδων

# Προαιρετικό: Αλλάζουμε το όνομα που φαίνεται στο ποντίκι
fig.update_traces(hovertemplate='Πωλήσεις: %{y}€')

fig.show()
```

Κ. 4.2.4 Γραμμικό Διάγραμμα με plotly



Ε. 4.2.4 Γραμμικό Διάγραμμα με plotly

Το διάγραμμα εμφανισιακά μοιάζει με αυτό του seaborn, αλλά:

1. Αν περάσετε το ποντίκι πάνω από τις κουκκίδες, εμφανίζεται ένα κουτάκι (Tooltip) με την ακριβή τιμή.

2. Πάνω δεξιά εμφανίζεται μια μπάρα εργαλείων για Zoom, Pan, Download image.

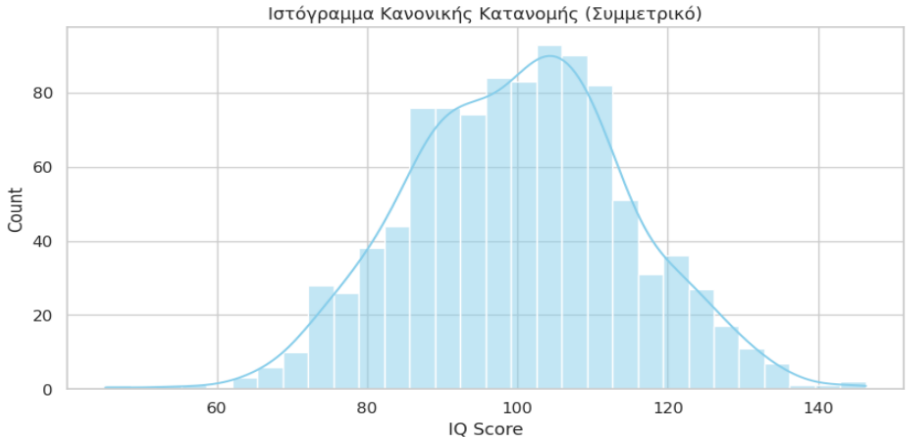
4.2.2. Ανάλυση Κατανομής

Η ανάλυση κατανομής είναι το πρώτο πράγμα που κάνουμε όταν λαμβάνουμε ένα νέο dataset. Μας βοηθάει να καταλάβουμε αν τα δεδομένα μας είναι συμμετρικά (κανονική κατανομή) ή αν έχουν λοξότητα, καθώς και να εντοπίσουμε ακραίες τιμές.

Το **ιστόγραμμα (histogram)** χωρίζει τα δεδομένα μας σε "κάδους" (bins) και μετράει πόσες παρατηρήσεις πέφτουν μέσα στον καθένα. Το KDE (Kernel Density Estimate) είναι η λεία γραμμή που ζωγραφίζεται πάνω από το ιστόγραμμα και μας δείχνει την τάση/καμπύλη της κατανομής. Θα δημιουργήσουμε δύο τυχαία σύνολα δεδομένων: ένα συμμετρικό (ηλικίες ενηλίκων) και ένα με λοξότητα (μισθοί, όπου οι περισσότεροι είναι χαμηλοί και λίγοι είναι τεράστιοι) (Κ. 4.2.5).

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_theme(style="whitegrid")

# 1. Δημιουργία Δεδομένων
# Κανονική κατανομή (π.χ. Ύψος ή IQ)
normal_data = np.random.normal(loc=100, scale=15, size=1000)

# Λοξή κατανομή (π.χ. Εισόδημα) - Χρησιμοποιούμε εκθετική κατανομή
skewed_data = np.random.exponential(scale=10, size=1000)

df = pd.DataFrame({
    'IQ_Score': normal_data,
    'Income': skewed_data
})

# 2. Δημιουργία Ιστογράμματος (Seaborn)
plt.figure(figsize=(10, 5))

# bins: Πόσους "κάδους" θα έχει το γράφημα.
# kde=True: Εμφανίζει την καμπύλη πυκνότητας.
sns.histplot(df['IQ_Score'], bins=30, kde=True, color='skyblue')

plt.title('Ιστόγραμμα Κανονικής Κατανομής (Συμμετρικό)')
plt.xlabel('IQ Score')
plt.show()
```

Κ. 4.2.5 Ιστόγραμμα με seaborn I

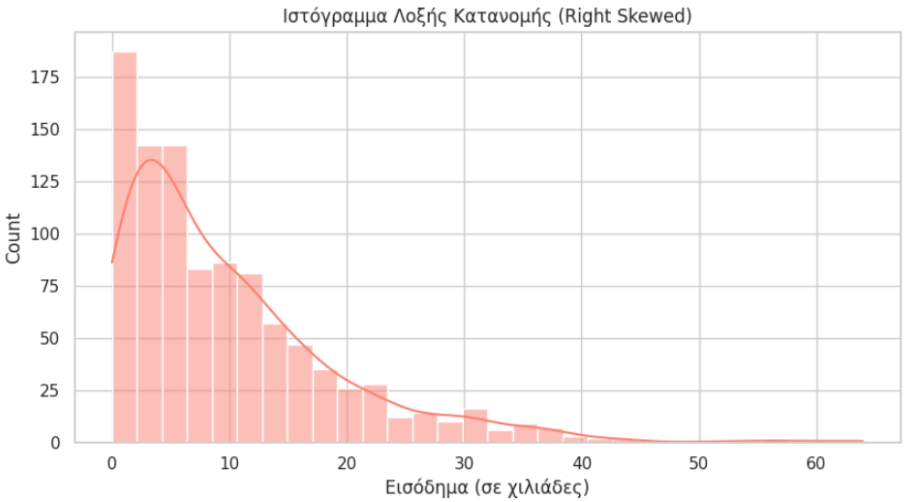


Ε. 4.2.5 Ιστόγραμμα με seaborn I

Στο Ε. 4.2.5 βλέπουμε το κλασικό σχήμα "καμπάνας" (Bell Curve). Η κορυφή της καμπύλης (KDE) θα είναι κοντά στο 100 (μέσος όρος) και οι ουρές θα σβήνουν συμμετρικά δεξιά και αριστερά. Τώρα ας δούμε την Λοξή Κατανομή (Εισόδημα) (Κ. 4.2.6):

```
plt.figure(figsize=(10, 5))
sns.histplot(df['Income'], bins=30, kde=True, color='salmon')

plt.title('Ιστόγραμμα Λοξής Κατανομής (Right Skewed)')
plt.xlabel('Εισόδημα (σε χιλιάδες)')
plt.show()
```

Κ. 4.2.6 Ιστόγραμμα με seaborn II



Ε. 4.2.5 Ιστόγραμμα με seaborn II

Όπως εμφάινεται στο Ε. 4.2.6, η κορυφή είναι τέρμα αριστερά (χαμηλά εισοδήματα) και υπάρχει μια μακριά ουρά που εκτείνεται προς τα δεξιά. Αυτό μας φωνάζει ότι "Ο μέσος όρος δεν είναι αντιπροσωπευτικός εδώ!".

Το **θηκόγραμμα (boxplot)** είναι η οπτική απεικόνιση των 5 Αριθμών που μάθαμε στην ποσοτική προσέγγιση (Min, Q1, Median, Q3, Max). Είναι το απόλυτο εργαλείο για τον εντοπισμό ακράιων τιμών.

Πώς διαβάζεται ένα θηκόγραμμα:

1. Το Κουτί (Box): Περιέχει το μεσαίο 50% των δεδομένων (από Q1 έως Q3). Το ύψος του είναι το IQR.

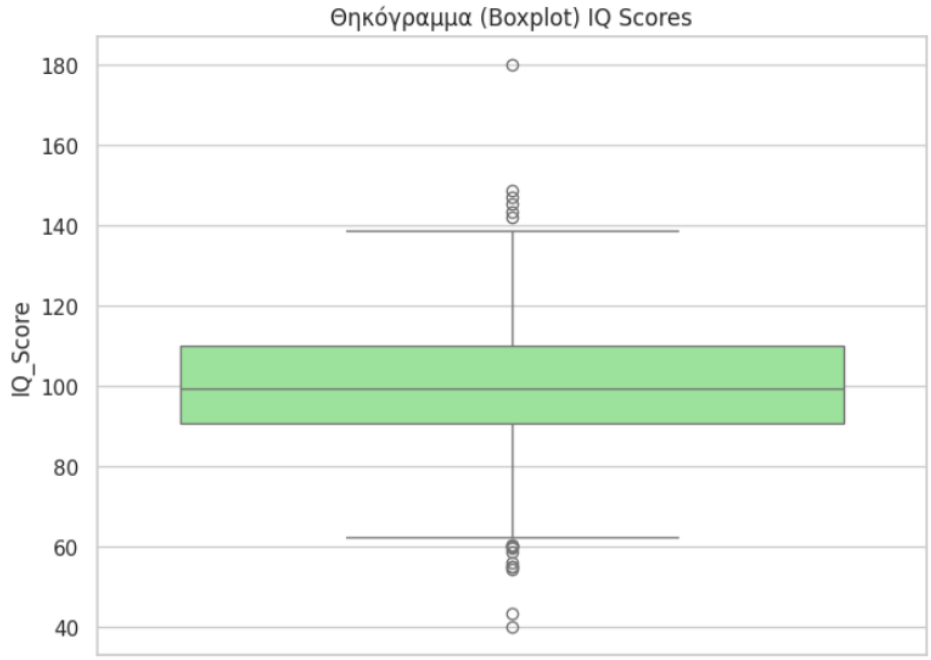
2. Η Γραμμή στη μέση: Είναι η Διάμεσος (όχι ο μέσος όρος!).

3. Τα "Μουστάκια" (Whiskers): Εκτείνονται μέχρι τις τιμές που δεν είναι outliers.

4. Οι Τελείες (Points): Κάθε τελεία έξω από τα μουστάκια είναι Ακραία Τιμή (Outlier).

```
# Ας προσθέσουμε μερικά ακραία outliers στα δεδομένα του IQ για να φανούν
df_box = df.copy()
df_box.loc[998] = 180 # Μια ιδιοφυΐα (Outlier ψηλά)
df_box.loc[999] = 40  # Μια πολύ χαμηλή τιμή (Outlier χαμηλά)

plt.figure(figsize=(8, 6))

# Δημιουργία Boxplot
sns.boxplot(y=df_box['IQ_Score'], color='lightgreen')

plt.title('Θηκόγραμμα (Boxplot) IQ Scores')
plt.show()
```

Κ. 4.2.6 Θηκόγραμμα με seaborn Ι

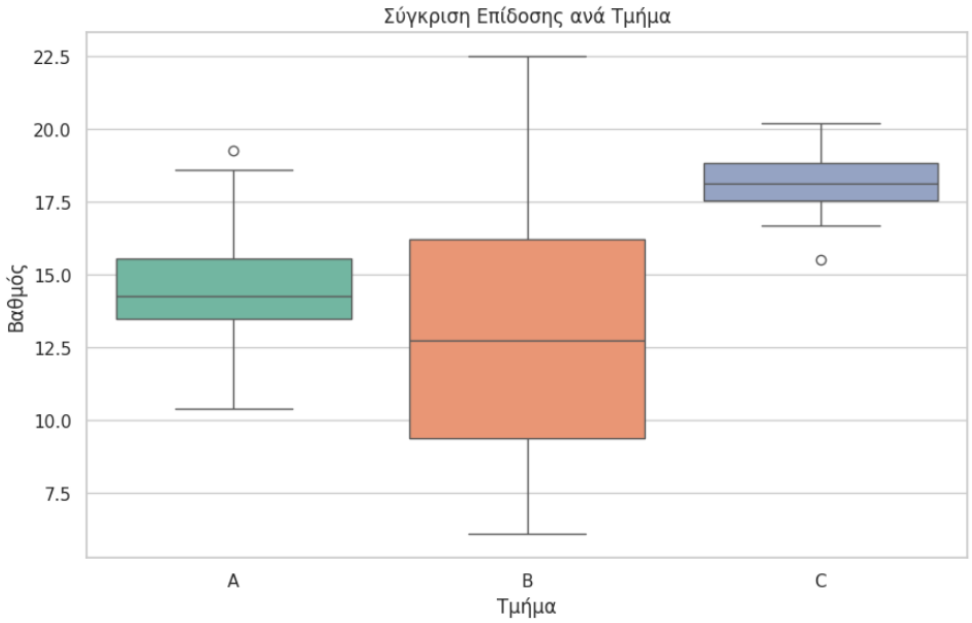


Ε. 4.2.6 Θηκόγραμμα με seaborn I

Στο Ε. 4.2.6 βλέπουμε ένα πράσινο ορθογώνιο στη μέση, όπως επίσης και κάποιες οριζόντιες γραμμές (whiskers) πάνω και κάτω. Οι μεμονωμένες τελίτσες πάνω από το 140-150 και κάτω από το 60, συμβολίζουν τις ακραίες τιμές που εντόπισε αυτόματα το seaborn. Η πραγματική αξία του θηκογράμματος φαίνεται όταν συγκρίνουμε πολλές ομάδες δίπλα-δίπλα (Κ. 4.2.7).

```
# Δημιουργία δεδομένων: Βαθμοί μαθητών σε 3 διαφορετικά τμήματα
class_data = pd.DataFrame({
    'Τμήμα': ['A']*50 + ['B']*50 + ['C']*50,
    'Βαθμός': np.concatenate([
        np.random.normal(15, 2, 50),  # Τμήμα A: Καλοί βαθμοί
        np.random.normal(12, 4, 50),  # Τμήμα B: Μεγάλη αστάθεια (μεγάλο
std)
        np.random.normal(18, 1, 50)   # Τμήμα C: Αριστούχοι (μικρό std)
    ])
})

plt.figure(figsize=(10, 6))

# x = Η κατηγορία (Τμήμα), y = Η ποσοτική μεταβλητή (Βαθμός)
sns.boxplot(data=class_data, x='Τμήμα', y='Βαθμός', palette="Set2")

plt.title('Σύγκριση Επίδοσης ανά Τμήμα')
plt.show()
```

Κ. 4.2.7 Θηκόγραμμα με seaborn II



Ε. 4.2.7 Θηκόγραμμα με seaborn II

Στο Ε. 4.2.7 βλέπουμε ότι για το Τμήμα C το κουτί είναι πολύ "στενό" (μικρό IQR) και ψηλά. Αυτό σημαίνει σταθερά καλοί βαθμοί. Τουναντίον, στο Τμήμα Β, το κουτί και τα μουστάκια είναι κατά πολύ μεγαλύτερα., πράγμα το οποίον συνεπάγεται ότι το εν λόγω τμήμα έχει και άριστους και πολύ αδύναμους φοιτητές (μεγάλη διασπορά).

4.2.3. Ανάλυση Εξέλιξης (Time Series Analysis)

Η ανάλυση χρονοσειρών αφορά δεδομένα που έχουν χρονική σειρά (π.χ. Τιμές Μετοχών, Θερμοκρασία ανά ώρα, Πωλήσεις ανά μήνα). Σε αντίθεση με τις απλές κατανομές, εδώ η σειρά των δεδομένων έχει κρίσιμη σημασία. Στα **γραφήματα εξέλιξης** ψάχνουμε κυρίως τρία πράγματα:

1. Τάση (Trend): Μακροπρόθεσμη άνοδος ή πτώση.
2. Εποχικότητα (Seasonality): Μοτίβα που επαναλαμβάνονται σε τακτά διαστήματα (π.χ. κάθε καλοκαίρι).
3. Θόρυβο (Noise): Τυχαίες αυξομειώσεις.

Για να δουλέψουμε σωστά με χρονοσειρές στην Python, πρέπει η στήλη της ημερομηνίας να είναι τύπου datetime και όχι απλό κείμενο (string). Για αυτή την ενότηταθα χρησιμοποιήσουμε το DataFrame όπως αποτυπώνεται στον Κ. 4.2.8.

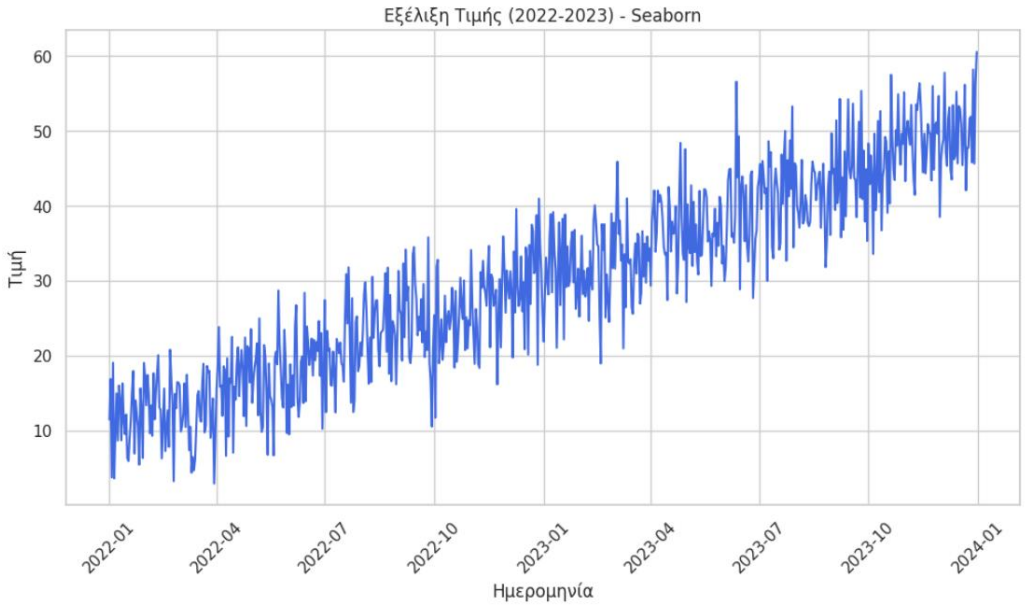
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px

# Δημιουργία δεδομένων για 2 χρόνια (730 ημέρες)
dates = pd.date_range(start='2022-01-01', periods=730, freq='D')

# Δημιουργούμε μια "μετοχή" που ανεβαίνει αλλά με σκαμπανεβάσματα
values = np.linspace(10, 50, 730) + np.random.normal(0, 5, 730)

df_time = pd.DataFrame({
    'Date': dates,
    'Price': values
})

print(df_time.head())
# Η στήλη Date είναι τύπου datetime64[ns]
```

Κ. 4.2.8 DataFrame για τα γραφήματα εξέλιξης

Το seaborn είναι ιδανικό για να δείξουμε τη γενική τάση σε μια αναφορά (report). Επίσης όταν έχουμε πολλές ημερομηνίες, τα γράμματα στον άξονα X "καβαλάει" το ένα το άλλο. Χρησιμοποιούμε το plt.xticks(rotation=45) για να τα γυρίσουμε πλαγιαστά (Κ. 4.2.9).

```
plt.figure(figsize=(12, 6))

# Lineplot
sns.lineplot(data=df_time, x='Date', y='Price', color='royalblue',
linewidth=1.5)

plt.title('Εξέλιξη Τιμής (2022-2023) - Seaborn')
plt.xlabel('Ημερομηνία')
plt.ylabel('Τιμή')

# Περιστροφή ημερομηνιών για να διαβάζονται
plt.xticks(rotation=45)

plt.show()
```

Κ. 4.2.9 Γράφημα εξέλιξης με seaborn

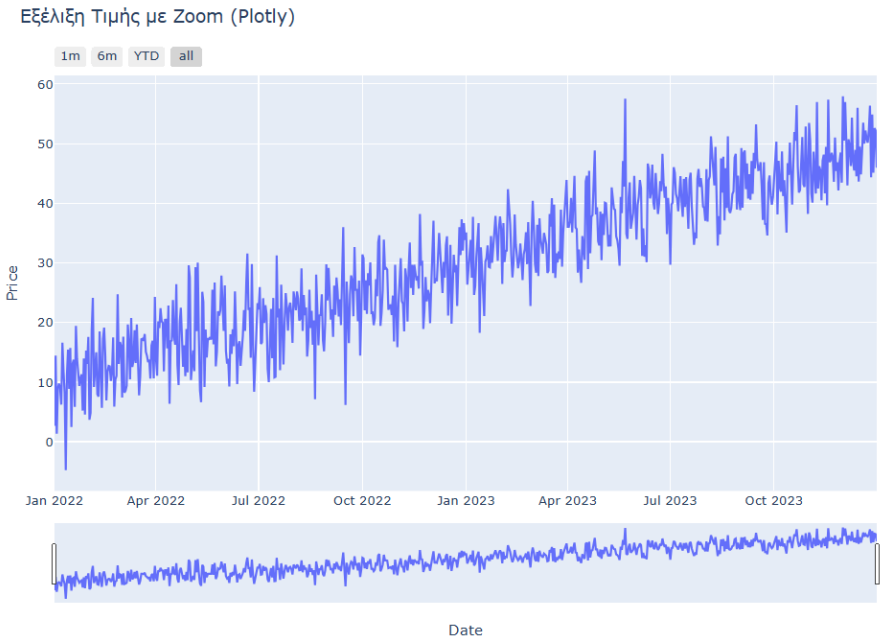


Ε. 4.2.9 Γράφημα εξέλιξης με seaborn

Στο Ε. 4.2.9 παρατηρούμε μία συνεχή μπλε γραμμή που δείχνει την ανοδική πορεία. Ωστόσο, είναι δύσκολο να δούμε τι ακριβώς έγινε π.χ. στις 15 Μαρτίου του 2022. Σε αυτές τις περιπτώσεις το plotly είναι μονόδρομος. Θα προσθέσουμε και ένα range slider (μπάρα κύλισης) στο κάτω μέρος, που επιτρέπει στον χρήστη να επιλέξει ποιο κομμάτι του χρόνου θέλει να δει (Κ. 4.2.10).

```
fig = px.line(df_time,
              x='Date',
              y='Price',
              title='Εξέλιξη Τιμής με Zoom (Plotly)')

# Προσθήκη Range Slider (Η μπάρα κάτω από το γράφημα)
fig.update_xaxes(rangeslider_visible=True)

# Προαιρετικά: Προσθήκη κουμπιών για γρήγορο zoom (1 μήνας, 6 μήνες, Όλα)
fig.update_xaxes(
    rangeselector=dict(
        buttons=list([
            dict(count=1, label="1m", step="month", stepmode="backward"),
            dict(count=6, label="6m", step="month", stepmode="backward"),
            dict(count=1, label="YTD", step="year", stepmode="todate"),
            dict(step="all")
        ])
    )
)

fig.show()
```

Κ. 4.2.10 Γράφημα εξέλιξης με plotly



Ε. 4.2.10 Γράφημα εξέλιξης με plotly

Με το plotly, ο χρήστης μπορεί να σύρει το ποντίκι και να κάνει zoom σε μια εβδομάδα. Επίσης κάτω από το γράφημα υπάρχει μια μικρογραφία. Σέρνοντας τις λαβές, αλλάζει το χρονικό παράθυρο. Μια άλλη χρήσιμη λειτουργία είναι αυτή που προσφέρεται από τα κουμπακια στο επάνω τμη΄μα του γραφήματος. Εδώ Πατώντας πχ. το "1m", το γράφημα εστιάζει αυτόματα στον τελευταίο μήνα.

4.2.4. Ανάλυση Συσχέτισης (Correlation Analysis)

Η συσχέτιση είναι η σχέση μεταξύ δύο ποσοτικών μεταβλητών. Έχουμε τρία είδη συσχετίσεων:

1. **Θετική Συσχέτιση:** Όταν ανεβαίνει το ένα, ανεβαίνει και το άλλο (π.χ. Ώρες Μελέτης -> Βαθμός).

2. **Αρνητική Συσχέτιση:** Όταν ανεβαίνει το ένα, το άλλο πέφτει (π.χ. Βάρος Αυτοκινήτου -> Οικονομία Καυσίμου).

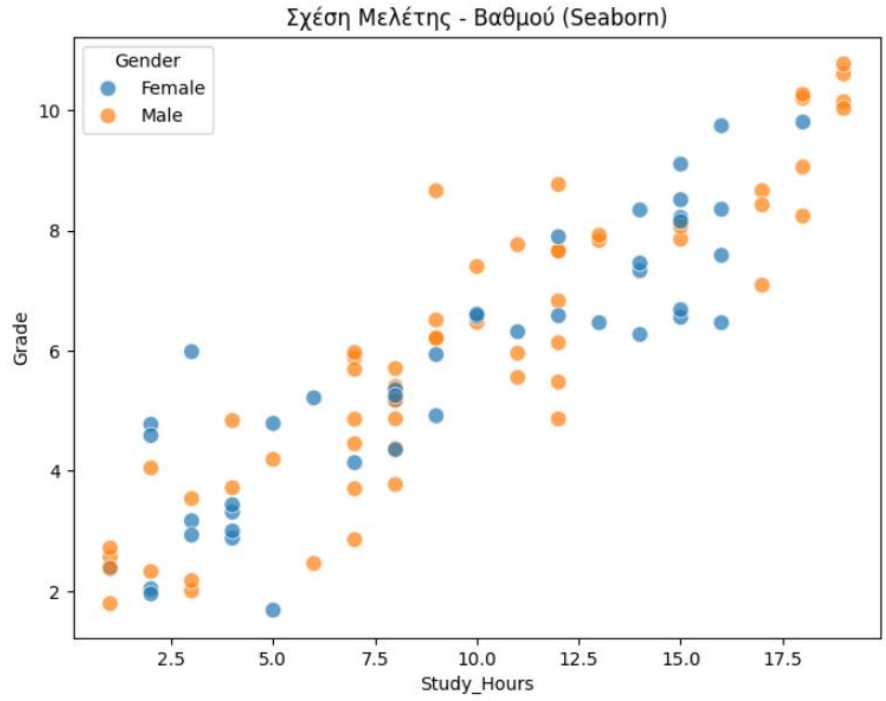
3. **Μηδενική Συσχέτιση:** Δεν υπάρχει εμφανές μοτίβο.

Το **διάγραμμα διασποράς (Scatter Plot)** είναι ένα νέφος σημείων. Κάθε τελεία αντιπροσωπεύει μία εγγραφή (π.χ. έναν φοιτητή). Παράδειγμα συσχέτισης είναι οι Ώρες Μελέτης (Study_Hours) vs Βαθμός (Grade), όπου θα προσθέσουμε και μια τρίτη διάσταση (κατηγορική), το Φύλο, χρησιμοποιώντας διαφορετικό χρώμα (Κ. 4.2.11).

```
import pandas as pd
import numpy as np
import seaborn as sns
```

```
import matplotlib.pyplot as plt
import plotly.express as px

# Δημιουργία Δεδομένων
np.random.seed(42)
n = 100
df_scatter = pd.DataFrame({
    'Study_Hours': np.random.randint(1, 20, n),
    'Gender': np.random.choice(['Male', 'Female'], n)
})
# Ο βαθμός εξαρτάται από τις ώρες μελέτης + λίγο θόρυβο
df_scatter['Grade'] = 2 + 0.4 * df_scatter['Study_Hours'] +
np.random.normal(0, 1, n)

# 1. Στατικό Scatter Plot με Seaborn
plt.figure(figsize=(8, 6))

# hue='Gender': Χρωματίζει τις τελείες ανάλογα με το φύλο
# s=100: Μέγεθος τελείας
sns.scatterplot(data=df_scatter, x='Study_Hours', y='Grade',hue='Gender',
s=80, alpha=0.7)

plt.title('Σχέση Μελέτης - Βαθμού (Seaborn)')
plt.show()
```

Κ. 4.2.11 Διάγραμμα Διασποράς με seaborn



Ε. 4.2.11 Διάγραμμα διασποράς με seaborn

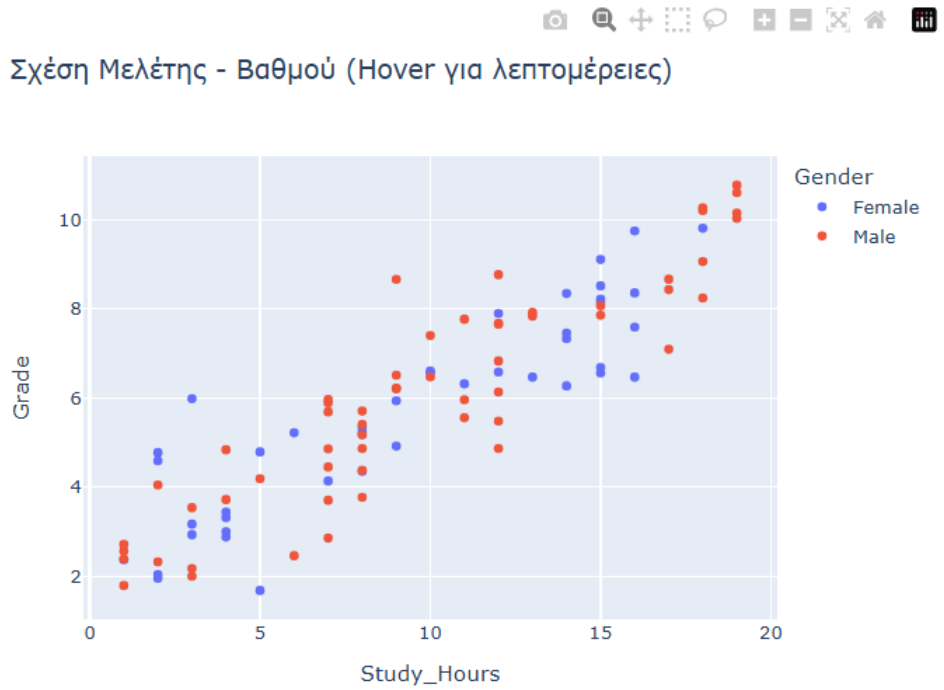
Η παράμετρος hue είναι πανίσχυρη. Μας επιτρέπει να δούμε αν μια ομάδα (π.χ. Male/Female) ομαδοποιείται σε συγκεκριμένη περιοχή του γραφήματος. Η παράμετρος alpha καθορίζει τη

διαφάνεια (alpha=0.7) και βοηθάει στην ενάργεια του διαγράμματος όταν πολλές τελείες πέφτουν η μία πάνω στην άλλη.

Στα Διαγράμματα Διασποράς, το plotly είναι εξαιρετικό γιατί συχνά θέλουμε να μάθουμε "Ποια είναι αυτή η τελεία που ξέφυγε;". Στο plotly, περνώντας το ποντίκι πάνω από μια "ακραία" τελεία, βλέπετε ακριβώς τις τιμές της, κάτι που είναι αδύνατον σε κάποιο στατικό γράφημα (Κ. 4.2.12).

```
# 2. Διαδραστικό Scatter Plot με Plotly
fig = px.scatter(df_scatter,
                 x='Study_Hours',
                 y='Grade',
                 color='Gender',
                 title='Σχέση Μελέτης - Βαθμού (Hover για λεπτομέρειες)',
                 hover_data=['Study_Hours', 'Grade'])
# Τι θα δείχνει το κουτάκι

fig.show()
```

Κ. 4.2.12 Διάγραμμα διασποράς με plotly



Ε. 4.2.12 Διάγραμμα διασποράς με plotly

Όταν έχουμε πολλές ποσοτικές μεταβλητές, είναι δύσκολο να δημιουργήσουμε διαγράμματα διασποράς για κάθε πιθανό ζευγάρι. Εδώ ακριβώς υπεισέρχεται ο **Πίνακας Συσχέτισης (Correlation Matrix),** ο οποίος υπολογίζει τον συντελεστή Pearson r για όλα τα ζευγάρια, όπου:

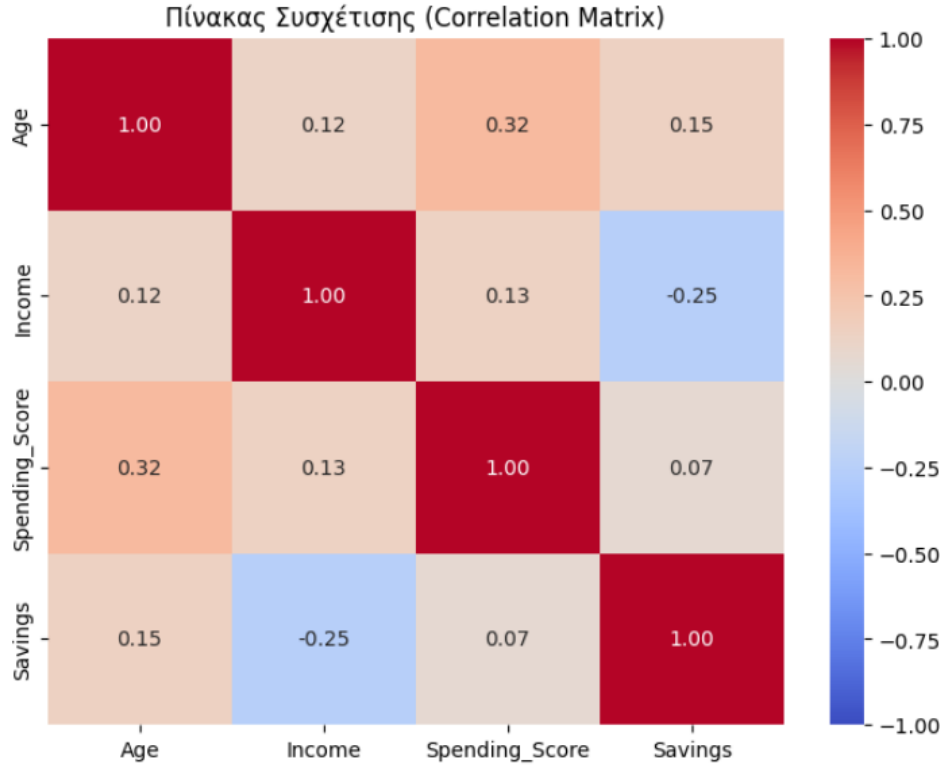
- r = 1: Τέλεια Θετική Συσχέτιση.

- r = -1: Τέλεια Αρνητική Συσχέτιση.
- r = 0: Καμία συσχέτιση.

Το **Θερμικό Διάγραμμα Συσχέτισης** χρωματίζει αυτόν τον πίνακα για να εντοπίσουμε οπτικά έντονες σχέσεις (Κ. 4.2.13).

```
# Δημιουργία ενός DataFrame με πολλαπλές μεταβλητές
data_multi = pd.DataFrame({
    'Age': np.random.randint(20, 60, 50),
    'Income': np.random.randint(20000, 80000, 50),
    'Spending_Score': np.random.randint(1, 100, 50),
    'Savings': np.random.randint(1000, 20000, 50)
})

# Υπολογισμός Πίνακα Συσχέτισης
corr_matrix = data_multi.corr()

plt.figure(figsize=(8, 6))

# Δημιουργία Heatmap
# annot=True: Εμφανίζει τα νούμερα μέσα στα κουτάκια
# cmap='coolwarm': Μπλε για αρνητική, Κόκκινο για θετική συσχέτιση
# fmt=".2f": Στρογγυλοποίηση σε 2 δεκαδικά
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", vmin=-1,
vmax=1)

plt.title('Πίνακας Συσχέτισης (Correlation Matrix)')
plt.show()
```

Κ. 4.2.13 Δημιουργία θερμικού διαγράμματος συσχέτισης με seaborn



Ε. 4.2.13 Δημιουργία θερμικού διαγράμματος συσχέτισης με seaborn

Το θερμικό διάγραμμα συσχέτισης στο Ε. 4.2.13 αποκαλύπτει κυρίως ασθενείς σχέσεις μεταξύ των μεταβλητών του δείγματος. Η πιο αξιοσημείωτη θετική συσχέτιση εντοπίζεται ανάμεσα στην Ηλικία (Age) και το Spending_Score (r=0.32), υποδηλώνοντας μια ελαφριά τάση όπου οι μεγαλύτεροι σε ηλικία έχουν υψηλότερο σκορ εξόδων. Αντίθετα, η πιο έντονη αρνητική συσχέτιση εμφανίζεται μεταξύ του Εισοδήματος (Income) και των Αποταμιεύσεων (Savings) (r=-0.25), που σημαίνει ότι καθώς αυξάνεται το εισόδημα, οι αποταμιεύσεις τείνουν να μειώνονται ελαφρώς στο συγκεκριμένο δείγμα. Οι υπόλοιπες συσχετίσεις (όπως Savings vs Spending_Score στο 0.07) κυμαίνονται πολύ κοντά στο μηδέν, δείχνοντας ότι δεν υπάρχει ουσιαστική γραμμική σχέση μεταξύ αυτών των ζευγών.

4.2.5. Σύγκριση Κατηγοριών (Categorical Comparison)

Σε αυτή την ενότητα ασχολούμαστε με δεδομένα που χωρίζονται σε ομάδες (π.χ. Φύλο, Πόλη, Τμήμα, Ημέρα της Εβδομάδας). Έχουμε δύο βασικά είδη ερωτημάτων:

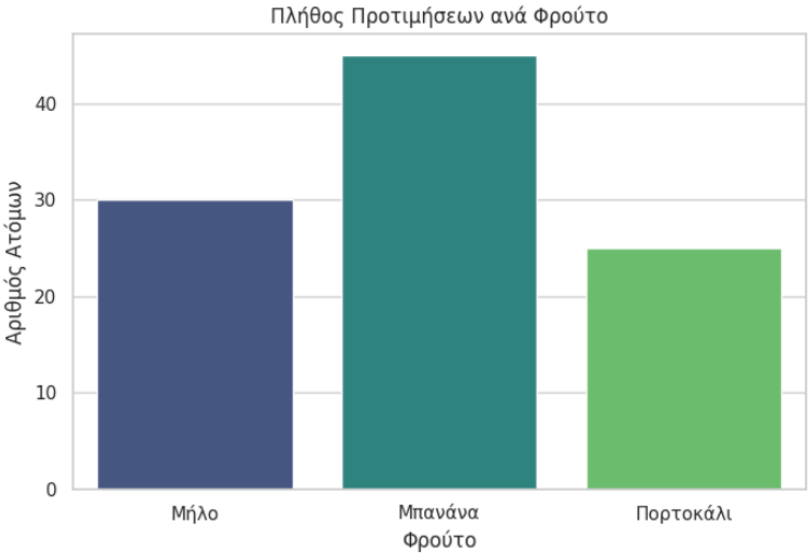
1. Πόσοι είναι; (Συχνότητα - Count Plot)

2. Ποιος είναι ο Μέσος Όρος τους; (Σύγκριση Μεγεθών - Bar Plot)

Το **ραβδόγραμμα** είναι το αντίστοιχο του Ιστογράμματος, αλλά για κατηγορίες. Μετράει απλώς πόσες εγγραφές υπάρχουν σε κάθε κατηγορία. Ένα παράδειγμα που εντάσσεται σε αυτή την κατηγορία είναι οι προτιμήσεις πελατών. Ας υποθέσουμε ότι ρωτήσαμε 100 άτομα ποιο είναι το αγαπημένο τους φρούτο (Κ. 4.2.14).

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Ρύθμιση στυλ
sns.set_theme(style="whitegrid")

# Δημιουργία Δεδομένων
data_cat = pd.DataFrame({
    'Φρούτο': ['Μήλο']*30 + ['Μπανάνα']*45 + ['Πορτοκάλι']*25,
    'Φύλο': ['Άνδρας', 'Γυναίκα'] * 50
})

plt.figure(figsize=(8, 5))

# countplot: Μετράει αυτόματα τις εγγραφές
sns.countplot(data=data_cat, x='Φρούτο', palette='viridis')

plt.title('Πλήθος Προτιμήσεων ανά Φρούτο')
plt.ylabel('Αριθμός Ατόμων')
plt.show()
```

Κ. 4.2.14 Δημιουργία ραβδογράμματος με seaborn



Ε. 4.2.14 Δημιουργία ραβδογράμματος με seaborn

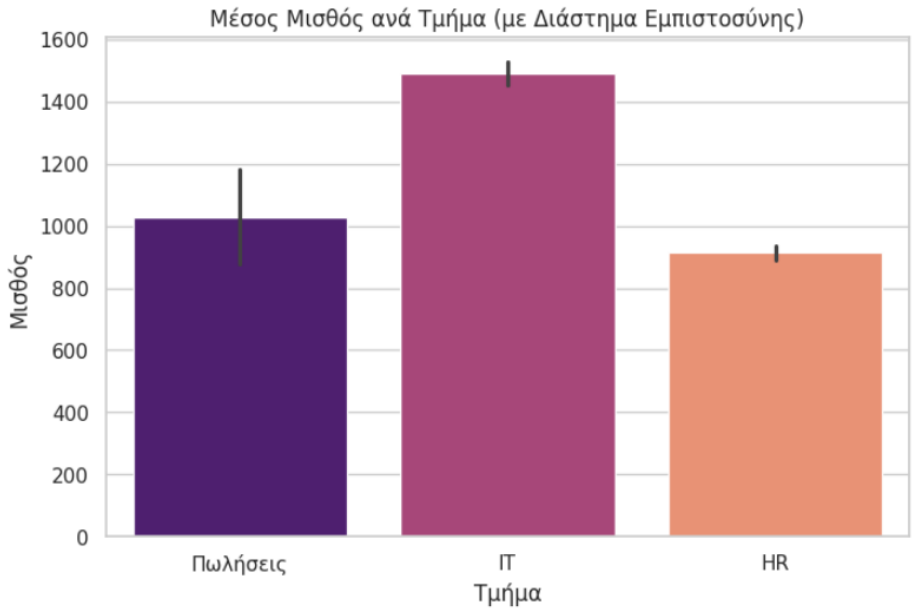
Στο Σ. 4.2.14 βλέπουμε αμέσως ότι η "Μπανάνα" είναι η δημοφιλέστερη επιλογή (ψηλότερη μπάρα). Στο επόμενο παράδειγμα (Κ. 4.2.15) με τις πωλήσεις τα πράγματα αλλάζουν. Εδώ δεν μετράμε απλά "πόσοι είναι", αλλά υπολογίζουμε έναν στατιστικό δείκτη (συνήθως τον μέσο όρο) μιας ποσοτικής μεταβλητής για κάθε κατηγορία.

Το sns.barplot του seaborn υπολογίζει αυτόματα τον μέσο όρο και σχεδιάζει μια μαύρη γραμμή στην κορυφή κάθε μπάρας. Αυτή η γραμμή είναι το διάστημα εμπιστοσύνης (Confidence Interval - 95%). Εάν η εν λόγω γραμμή είναι σχετικά μικρή, τότε είμαστε σίγουροι για τον μέσο όρο. Μία μεγάλη γραμμή υποδηλώνει μεγάλη αβεβαιότητα ή λίγα δεδομένα.

```
# Δημιουργία Δεδομένων: Μισθοί ανά Τμήμα
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

dept_data = pd.DataFrame({
    'Τμήμα': ['Πωλήσεις']*20 + ['IT']*20 + ['HR']*20,
    # Οι Πωλήσεις έχουν μεγάλη διακύμανση (κάποιοι παίρνουν πολλά, κάποιοι
λίγα)
    # Το IT έχει μεγάλους μισθούς αλλά σταθερούς
    'Μισθός': np.concatenate([
        np.random.normal(1000, 300, 20), # Πωλήσεις
        np.random.normal(1500, 100, 20), # IT
        np.random.normal(900, 50, 20)    # HR
    ])
})
```

```
plt.figure(figsize=(8, 5))

# barplot: Υπολογίζει τον Μέσο Όρο (estimator=mean από default)
sns.barplot(data=dept_data, x='Τμήμα', y='Μισθός', palette='magma')

plt.title('Μέσος Μισθός ανά Τμήμα (με Διάστημα Εμπιστοσύνης)')
plt.show()
```

Κ. 4.2.15 Δημιουργία ραβδογράμματος μέσουό όρου με διάστημα επμπιστοσύνης με seaborn

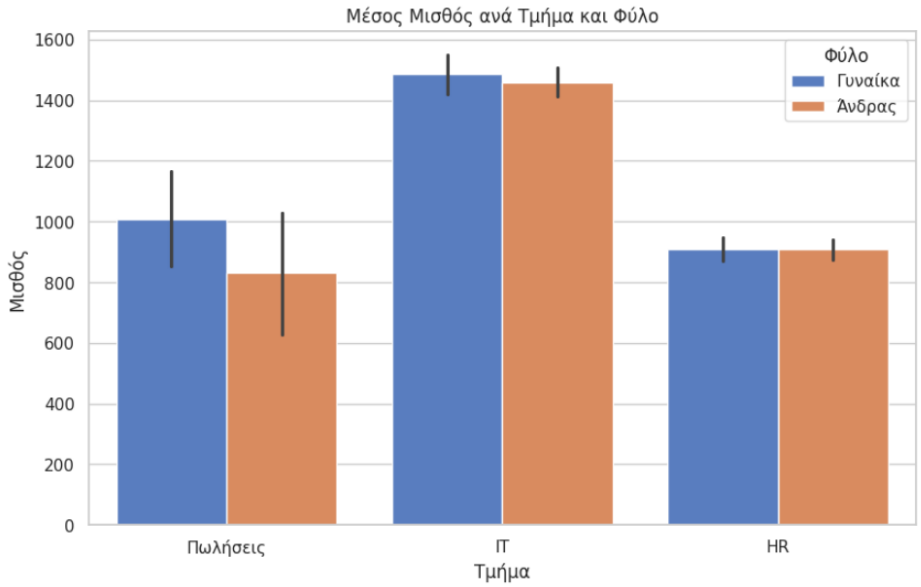


Ε. 4.2.15 Δημιουργία ραβδογράμματος μέσου όρου με διάστημα επμπιστοσύνης με seaborn

Στο Ε. 4.2.15 βλέπουμε ότι η Η μπάρα του IT είναι η πιο ψηλή (μεγαλύτερος μέσος μισθός). Η μαύρη γραμμή στις Πωλήσεις είναι πιθανότατα πιο μεγάλη από του HR, δείχνοντας ότι εκεί οι μισθοί διαφέρουν πολύ μεταξύ των υπαλλήλων. Όπως και στα διαγράμματα διασποράς, μπορούμε να προσθέσουμε μια τρίτη διάσταση χρησιμοποιώντας το hue. Αυτό θα δημιουργήσει ζευγάρια από μπάρες (Κ. 4.2.16).

```
# Ας δούμε τους μισθούς ανά τμήμα, αλλά χωρισμένους και ανά φύλο (τυχαία
δεδομένα)
dept_data['Φύλο'] = np.random.choice(['Άνδρας', 'Γυναίκα'], 60)

plt.figure(figsize=(10, 6))

sns.barplot(data=dept_data, x='Τμήμα', y='Μισθός', hue='Φύλο',
palette='muted')

plt.title('Μέσος Μισθός ανά Τμήμα και Φύλο')
plt.show()
```

Κ. 4.2.16 Δημιουργία σύνθετου ραβδογράμματος με seaborn



Ε. 4.2.16 Δημιουργία σύνθετου ραβδογράμματος με seaborn

4.2.6. Σύνθετες Απεικονίσεις και Υπογραφήματα

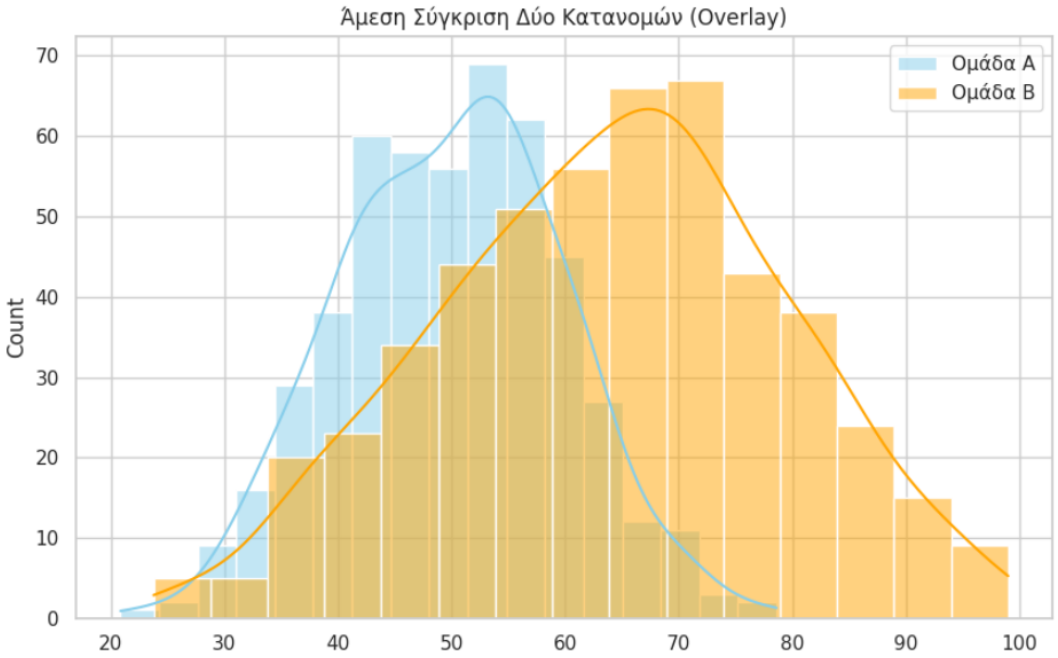
Μέχρι τώρα δημιουργούσαμε ένα γράφημα τη φορά. Στην πραγματική ανάλυση, όμως, η δύναμη βρίσκεται στη σύγκριση. Υπάρχουν δύο τρόποι να το πετύχουμε αυτό: η επικαλυψη και τα υπογραφήματα.

Η επικάλυψη χρησιμοποιείται όταν θέλουμε να συγκρίνουμε άμεσα δύο μεγέθη που έχουν την ίδια κλίμακα (π.χ. σύγκριση δύο κατανομών ή δύο μετοχών). Το μυστικό εδώ είναι η παράμετρος alpha (διαφάνεια), ώστε να βλέπουμε τι κρύβεται από πίσω. Στο παρακάτω παράδειγμα (Κ. 4.2.17) συγκρίνουμε δύο κατανομές με ιστογράμματα.

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

sns.set_theme(style="whitegrid")

# Δημιουργία δεδομένων: Δύο ομάδες με διαφορετικό μέσο όρο
data_a = np.random.normal(loc=50, scale=10, size=500) # Ομάδα Α
data_b = np.random.normal(loc=65, scale=15, size=500) # Ομάδα Β

plt.figure(figsize=(10, 6))

# Σχεδιάζουμε το πρώτο
```

```
sns.histplot(data_a, color="skyblue", label="Ομάδα Α", kde=True, alpha=0.5)
# Σχεδιάζουμε το δεύτερο ΣΤΟ ΙΔΙΟ "figure"
sns.histplot(data_b, color="orange", label="Ομάδα Β", kde=True, alpha=0.5)

plt.title("Άμεση Σύγκριση Δύο Κατανομών (Overlay)")
plt.legend() # Εμφάνιση υπομνήματος (απαραίτητο για να ξέρουμε ποιο είναι ποιο)
plt.show()
```

Κ. 4.2.17 Σύγκριση δύο κατανομών με ιστογράμματα του seaborn



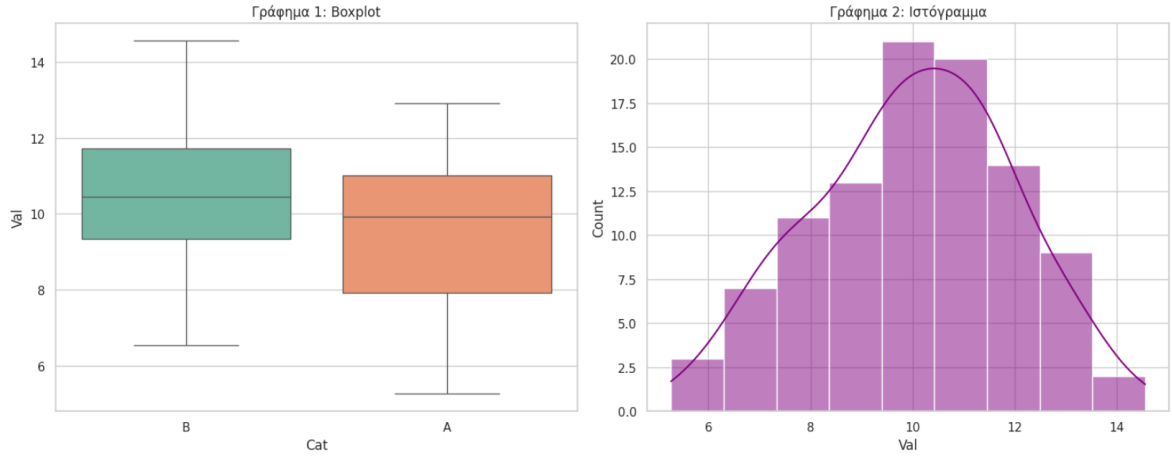
Ε. 4.2.17 Σύγκριση δύο κατανομών με ιστογράμματα του seaborn

Βλέπουμε αμέσως πού "τέμνονται" οι δύο ομάδες και πόσο διαφέρουν οι κορυφές τους. Αν τα βάζαμε σε ξεχωριστά γραφήματα, η σύγκριση θα ήταν δύσκολη. Όταν θέλουμε να δείξουμε διαφορετικά πράγματα δίπλα-δίπλα (π.χ. ένα θηκόγραμμα και ένα διάγραμμα διασποράς), χρησιμοποιούμε τη λειτουργία plt.subplots(). Η λειτουργία αυτή δημιουργεί ένα "πλέγμα" (grid) από άξονες (axes) (Κ. 4.2.18).

```
# Δημιουργία "ταμπλό" με 1 γραμμή και 2 στήλες (1x2)
# fig: Το συνολικό "χαρτί"
# axes: Μια λίστα που περιέχει τα επιμέρους γραφήματα (axes[0], axes[1])
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Δεδομένα (τυχαία)
df = pd.DataFrame({'Val': np.random.normal(10, 2, 100), 'Cat':
np.random.choice(['A', 'B'], 100)})

# Γράφημα 1 (Αριστερά) - Το τοποθετούμε στο ax=axes[0]
sns.boxplot(data=df, x='Cat', y='Val', ax=axes[0], palette="Set2")
axes[0].set_title("Γράφημα 1: Boxplot")

# Γράφημα 2 (Δεξιά) - Το τοποθετούμε στο ax=axes[1]
sns.histplot(data=df, x='Val', ax=axes[1], color='purple', kde=True)
axes[1].set_title("Γράφημα 2: Ιστόγραμμα")
```

```
plt.tight_layout() # Μαγική εντολή: Φτιάχνει αυτόματα τα κενά ώστε να μην πέφτει το ένα πάνω στο άλλο
plt.show()
```

Κ. 4.2.18 Δημιουργία υπογραφημάτων με seaborn

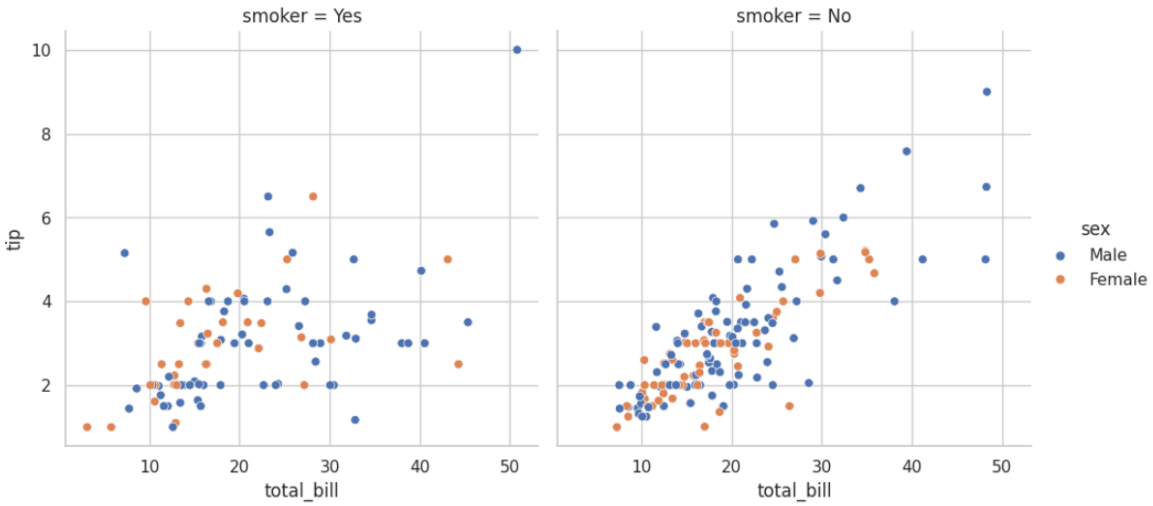


Ε. 4.2.18 Δημιουργία υπογραφημάτων με seaborn

Επίσης το seaborn έχει μια πολύ δυνατή λειτυγικότητα που λέγεται faceting. Αντί να φτιάχνουμε εμείς τα υπογραφήματα με το χέρι, του λέμε: "Φτιάξε μου ένα γράφημα για κάθε κατηγορία". Χρησιμοποιούμε τις εντολές relplot (για scatter/line) ή catplot (για bar/box) με την παράμετρο col (στήλη) ή row (γραμμή). Στο παρακάτω παράδειγμα (Κ. 4.2.19) έχουμε δεδομένα για λογαριασμούς σε εστιατόριο. Θέλουμε να δούμε τη σχέση Λογαριασμού-Φιλοδωρήματος, αλλά χωριστά για Καπνιστές και Μη Καπνιστές.

```
# Φόρτωση έτοιμου dataset του Seaborn
tips = sns.load_dataset("tips")

# col="smoker": Θα φτιάξει αυτόματα 2 γραφήματα δίπλα-δίπλα (ένα για Yes, ένα για No)
sns.relplot(
    data=tips,
    x="total_bill",
    y="tip",
    hue="sex",       # Χρώμα ανάλογα με το φύλο
    col="smoker",    # ΞΕΧΩΡΙΣΤΟ γράφημα ανάλογα με το κάπνισμα
    kind="scatter",
    height=5,
    aspect=1
)

plt.show()
```

Κ. 4.2.19 Δημιουργία υπογραφημάτων με το faceting του seaborn

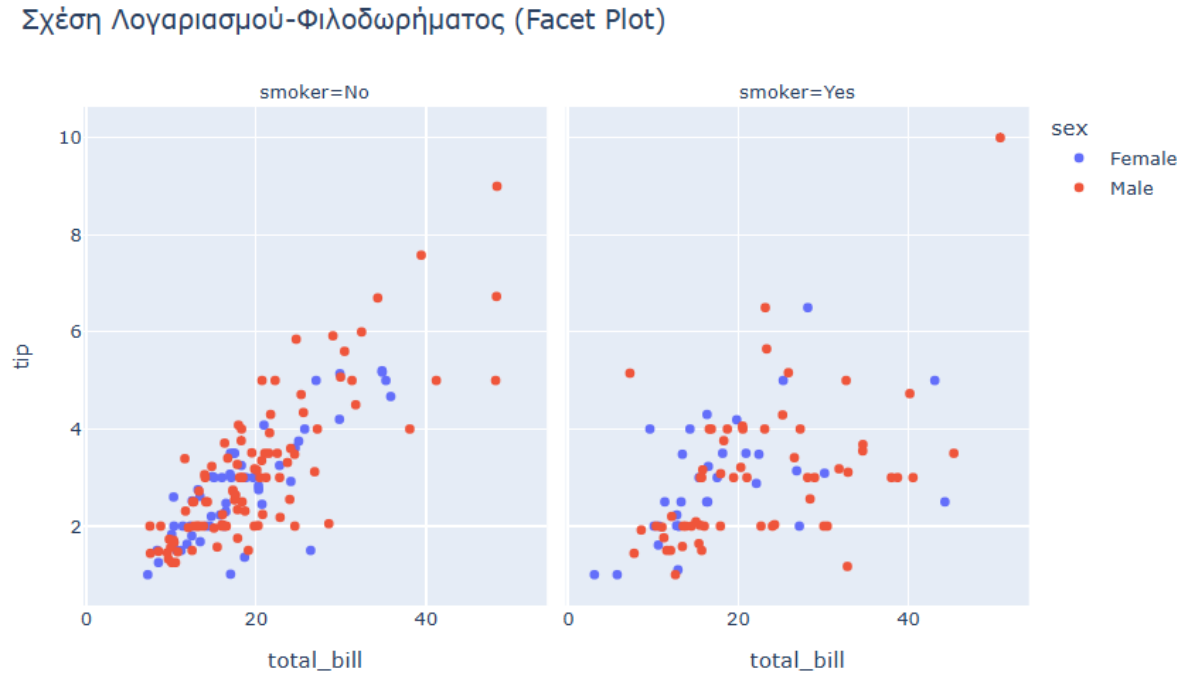


Ε. 4.2.19 Δημιουργία υπογραφημάτων με το faceting του seaborn

Και το Plotly υποστηρίζει faceting με ελάχιστο κόπο, χρησιμοποιώντας το facet_col (Κ. 4.2.20).

```
import plotly.express as px

# Ίδιο παράδειγμα με πριν (Καπνιστές vs Μη Καπνιστές)
fig = px.scatter(tips,
                 x="total_bill",
                 y="tip",
                 color="sex",
                 facet_col="smoker", # Η μαγική εντολή για subplots
                 title="Σχέση Λογαριασμού-Φιλοδωρήματος (Facet Plot)")

fig.show()
```

Κ. 4.2.20 Δημιουργία υπογραφημάτων με το faceting του plotly



Ε. 4.2.20 Δημιουργία υπογραφημάτων με το faceting του plotly

4.3. Εφαρμοσμένο Project: Ανάλυση Κερδοφορίας Καταστήματος

Μέχρι στιγμής εξετάσαμε τα εργαλεία μας μεμονωμένα. Σε αυτή την ενότητα, θα τα συνθέσουμε όλα μαζί σε ένα ρεαλιστικό σενάριο ανάλυσης δεδομένων. Σκοπός μας δεν είναι απλώς να γράψουμε κώδικα, αλλά να αφηγηθούμε μια ιστορία με τα δεδομένα (Data Storytelling).

Το Σενάριο: Εργάζεστε ως αναλυτής δεδομένων σε μια μεγάλη αλυσίδα λιανικής. Ο Οικονομικός Διευθυντής σάς παραδίδει τα δεδομένα πωλήσεων του τελευταίου έτους και θέτει ένα απλό αλλά κρίσιμο ερώτημα: *"Έχουμε υψηλό τζίρο, αλλά τα κέρδη μας δεν είναι τα αναμενόμενα. Πού χάνουμε χρήματα;"*

4.3.1. Δημιουργία και Επισκόπηση Δεδομένων

Ας ξεκινήσουμε δημιουργώντας ένα σύνολο δεδομένων που προσομοιώνει την πραγματικότητα (Κ. 4.3.1). Έχουμε καταγραφές για τρεις κατηγορίες προϊόντων: Τεχνολογία (Tech), Έπιπλα (Furniture) και Είδη Γραφείου (Office Supplies). Κάθε εγγραφή περιλαμβάνει την Ημερομηνία, το ποσό Πώλησης (Sales) και το Κέρδος (Profit).

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Ρυθμίσεις
np.random.seed(42)
sns.set_theme(style="whitegrid")

# Δημιουργία 500 εγγραφών
n = 500
categories = np.random.choice(['Tech', 'Furniture', 'Office'], n)
sales = np.random.exponential(scale=200, size=n) + 50  # Πωλήσεις με θετική
λοξότητα
dates = pd.date_range(start='2023-01-01', periods=n)

# Δημιουργία Κέρδους:
# Η Τεχνολογία έχει υψηλό κέρδος, τα Έπιπλα έχουν αστάθεια (συχνά ζημιές)
profit = []
for i in range(n):
    base_margin = 0.2 if categories[i] == 'Tech' else (0.05 if
categories[i] == 'Furniture' else 0.15)
    noise = np.random.normal(0, 40) # Τυχαίος παράγοντας
    profit.append(sales[i] * base_margin + noise)

df = pd.DataFrame({'Date': dates, 'Category': categories, 'Sales': sales,
'Profit': profit})

# Μια πρώτη ματιά στα στατιστικά
print(df.describe())
```

Κ. 4.3.1 Δημιουργία συνόλου δεδομένων

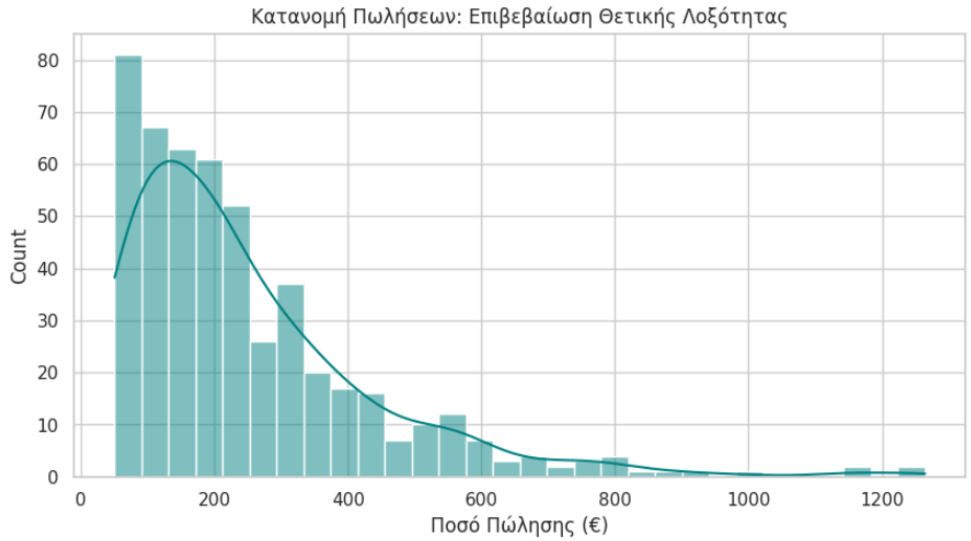
Κοιτάζοντας την εντολή describe(), παρατηρούμε αμέσως την πρώτη ένδειξη. Η μέση τιμή των Πωλήσεων είναι αισθητά μεγαλύτερη από τη διάμεσο (50%). Αυτό στην περιγραφική στατιστική υποδηλώνει θετική λοξότητα. Πρακτικά, σημαίνει ότι οι περισσότερες πωλήσεις μας είναι μικροποσά, αλλά υπάρχουν λίγες, πολύ μεγάλες παραγγελίες που τραβούν τον μέσο όρο προς τα πάνω. Αν βασιζόμασταν μόνο στον μέσο όρο για να κάνουμε προβλέψεις, θα είχαμε υπερεκτιμήσει τα έσοδα του "τυπικού" πελάτη.

4.3.2. Η Παγίδα των Πωλήσεων (Univariate Analysis)

Για να επιβεβαιώσουμε την υποψία μας για τη λοξότητα, οπτικοποιούμε την κατανομή των Πωλήσεων (Κ. 4.3.2). Αντί να κοιτάμε απλώς νούμερα, ένα Ιστόγραμμα θα μας δείξει το σχήμα των εσόδων μας.

```
plt.figure(figsize=(10, 5))
sns.histplot(df['Sales'], kde=True, color='teal', bins=30)
plt.title('Κατανομή Πωλήσεων: Επιβεβαίωση Θετικής Λοξότητας')
plt.xlabel('Ποσό Πώλησης (€)')
plt.show()
```

Κ. 4.3.2 Δημιουργία ιστογράμματος πωλήσεων



Ε. 4.3.2 Δημιουργία ιστογράμματος πωλήσεων

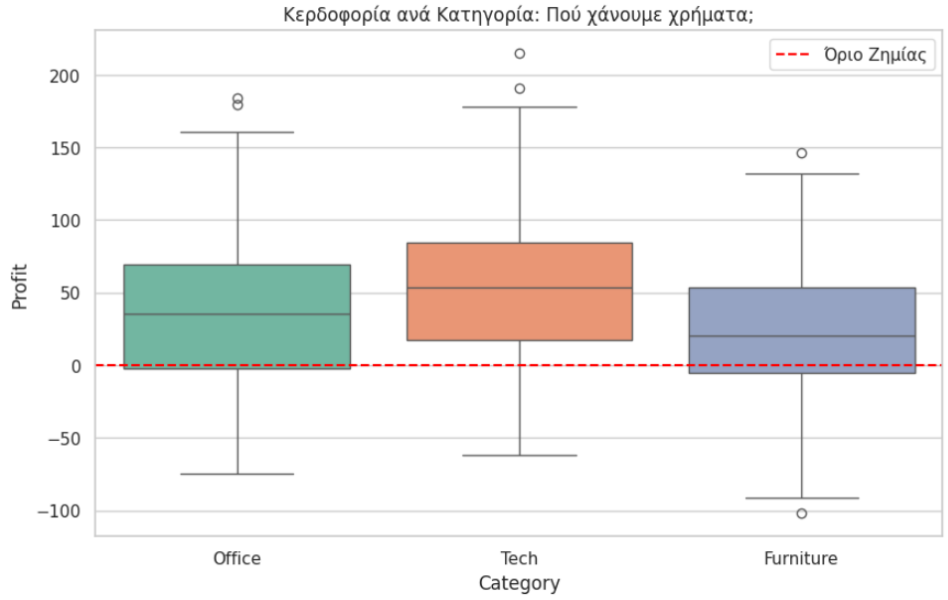
Το γράφημα επιβεβαιώνει την ανάλυσή μας. Βλέπουμε μια ξεκάθαρη "ουρά" προς τα δεξιά. Η πλειονότητα των συναλλαγών συγκεντρώνεται χαμηλά (κάτω από 200€), ενώ οι μεγάλες πωλήσεις άνω των 600€ είναι σπάνιες. Αυτό είναι μια κρίσιμη πληροφορία για το τμήμα Marketing: το προφίλ του πελάτη μας είναι ο "μικρός αγοραστής" και όχι ο μεγάλος επενδυτής.

4.3.3. Αναζητώντας τη Ζημία (Bivariate Analysis)

Ήρθε η ώρα να απαντήσουμε στο ερώτημα του Διευθυντή, *Γιατί τα κέρδη είναι χαμηλά;* Θα συγκρίνουμε την κερδοφορία ανά Κατηγορία χρησιμοποιώντας ένα θηκόγραμμα (Κ. 4.3.3). Το θηκόγραμμα είναι ιδανικό εδώ, διότι θα μας αποκαλύψει όχι μόνο τον μέσο όρο, αλλά και το ρίσκο (πόσο συχνά πέφτουμε σε αρνητικό κέρδος).

```
plt.figure(figsize=(10, 6))
# Σχεδιάζουμε το Κέρδος ανά Κατηγορία. Προσθέτουμε μια κόκκινη γραμμή στο
0.
sns.boxplot(data=df, x='Category', y='Profit', palette='Set2')
plt.axhline(0, color='red', linestyle='--', linewidth=1.5, label='Όριο
Ζημίας')
plt.title('Κερδοφορία ανά Κατηγορία: Πού χάνουμε χρήματα;')
plt.legend()
plt.show()
```

Κ. 4.3.3 Δημιουργία θηκογράμματος κερδοφορίας ανα κατηγορία



Ε. 4.3.3 Δημιουργία θηκογράμματος κερδοφορίας ανα κατηγορία

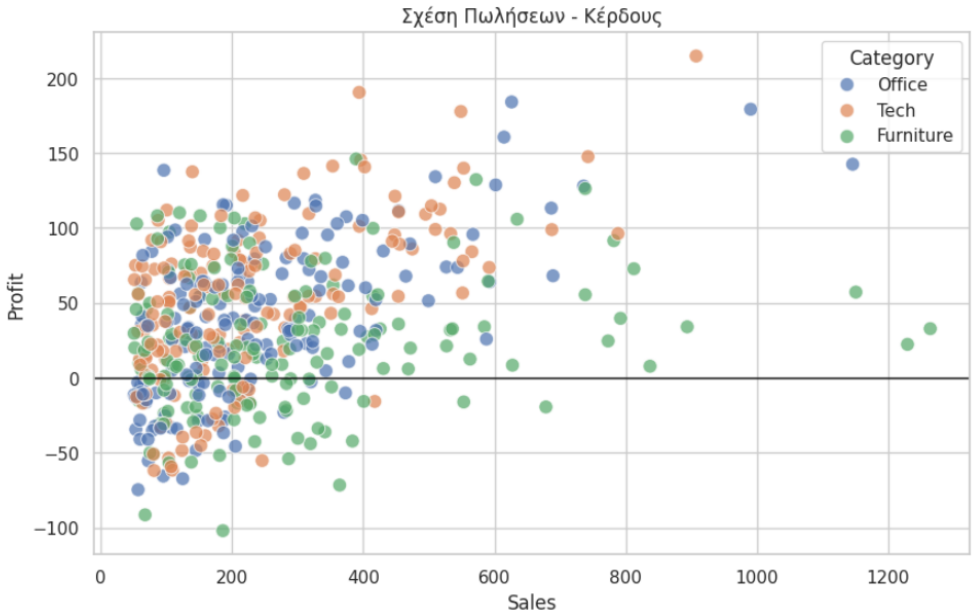
Η εικόνα είναι αποκαλυπτική. Ενώ η κατηγορία "Tech" (Τεχνολογία) έχει το κουτί της ξεκάθαρα πάνω από το μηδέν, η κατηγορία "Furniture" (Έπιπλα) αλλά και η κατηγορία “Office” (Είδη Γραφείου) παρουσιάζουν πρόβλημα. Ένα μέρος του "κουτιού" (που αντιπροσωπεύει το 50% των πωλήσεων) εκτείνονται κάτω από την κόκκινη γραμμή. Αυτό σημαίνει ότι στα Έπιπλα και Είδη Γραφείου, παρόλο που κάνουμε πωλήσεις, συχνά μπαίνουμε "μέσα" (πιθανώς λόγω υψηλών μεταφορικών ή εκπτώσεων). Η κατηγορία αυτή είναι ο υπαίτιος για τη μειωμένη κερδοφορία της εταιρείας.

4.3.4. Συσχέτιση Πωλήσεων και Κέρδους

Τέλος, ας εξετάσουμε αν η αύξηση του τζίρου φέρνει τελικά περισσότερα κέρδη. Θα χρησιμοποιήσουμε ένα διάγραμμα διασποράς (Κ. 4.3.4), χρωματίζοντας τα σημεία με βάση την κατηγορία για να δούμε τα μοτίβα συμπεριφοράς.

```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='Sales', y='Profit', hue='Category', alpha=0.7,
s=80)
plt.title('Σχέση Πωλήσεων - Κέρδους')
plt.axhline(0, color='black', linewidth=1) # Γραμμή μηδενικού κέρδους
plt.show()
```

Κ. 4.3.4 Δημιουργία διαγράμματος διασποράς κέρδους vs πωλήσεων



Ε. 4.3.4 Δημιουργία διαγράμματος διασποράς κέρδους vs πωλήσεων

Το διάγραμμα διασποράς αποκαλύπτει μια ξεκάθαρη, αλλά ταυτόχρονα ανησυχητική εικόνα για την οικονομική υγεία της επιχείρησης, με το πιο κρίσιμο σημείο αναφοράς να είναι η οριζόντια γραμμή του μηδενός που διαχωρίζει τις κερδοφόρες από τις ζημιογόνες συναλλαγές. Παρατηρώντας την κατηγορία της Τεχνολογίας (πορτοκαλί σημεία), βλέπουμε μια ισχυρή και σταθερή θετική συσχέτιση, όπου η αύξηση των πωλήσεων μεταφράζεται άμεσα και αναλογικά σε αύξηση του κέρδους. Τα σημεία σχηματίζουν μια απότομη ανοδική τάση και παραμένουν σταθερά πάνω από τη γραμμή του μηδενός, υποδηλώνοντας ένα υγιές περιθώριο κέρδους και χαμηλό επιχειρηματικό ρίσκο.

Στον αντίποδα, η κατηγορία των Επίπλων (πράσινα σημεία) παρουσιάζει σοβαρή δομική αδυναμία και αποτελεί την κύρια πηγή διαρροής εσόδων. Ένα σημαντικό πλήθος συναλλαγών εντοπίζεται κάτω από το όριο του μηδενός, γεγονός που σημαίνει ότι η εταιρεία συχνά χάνει χρήματα παρά την ολοκλήρωση της πώλησης, πιθανότατα λόγω του υψηλού κόστους μεταφοράς ή αποθήκευσης.

Ακόμη πιο ανησυχητικό είναι το φαινόμενο που παρατηρείται στο δεξί άκρο του γραφήματος: υπάρχουν περιπτώσεις πολύ υψηλού τζίρου που αποφέρουν ελάχιστο έως μηδαμινό κέρδος, μια συμπεριφορά που έρχεται σε πλήρη αντίθεση με την αποτελεσματικότητα του κλάδου της Τεχνολογίας.

Συνολικά, η οπτικοποίηση αυτή καταρρίπτει την πεποίθηση ότι ο υψηλός τζίρος συνεπάγεται αυτόματα και υψηλή κερδοφορία. Ενώ τα Είδη Γραφείου (μπλε σημεία) κινούνται σε ικανοποιητικά επίπεδα ακολουθώντας τον μέσο όρο, η στρατηγική της διοίκησης πρέπει να εστιάσει άμεσα στην εξυγίανση του κλάδου των Επίπλων. Είναι προφανές ότι τα λειτουργικά έξοδα ή η τιμολογιακή πολιτική στη συγκεκριμένη κατηγορία "κανιβαλίζουν" τα κέρδη, μειώνοντας τη συνολική απόδοση της εταιρείας παρά τον όγκο των πωλήσεων.

4.4. Ερωτήσεις Αυτοαξιολόγησης

1. **Ποιο μέτρο κεντρικής τάσης είναι το πιο κατάλληλο όταν τα δεδομένα παρουσιάζουν έντονη λοξότητα ή περιέχουν ακραίες τιμές;**

   α) Μέσος όρος

   β) Διάμεσος

   γ) Εύρος

   δ) Τυπική Απόκλιση

2. **Τι αντιπροσωπεύει το ενδοτεταρτημοριακό εύρος;**

   α) Τη διαφορά μεταξύ της μέγιστης και της ελάχιστης τιμής.

   β) Το εύρος στο οποίο βρίσκεται το μεσαίο 50% των δεδομένων.

   γ) Τον μέσο όρο των τετραγώνων των αποκλίσεων.

   δ) Το σημείο που χωρίζει τα δεδομένα σε δύο ίσα μέρη.

3. **Αν σε μια κατανομή ισχύει ότι μέσος όρος > διάμεσος, τι είδος λοξότητας έχουμε;**

   α) Αρνητική Λοξότητα

   β) Συμμετρική Κατανομή

   γ) Θετική Λοξότητα

   δ) Μηδενική Λοξότητα

4. **Ποιο είναι το βασικό μειονέκτημα της Διακύμανσης που λύνει η Τυπική Απόκλιση;**

   α) Δεν μπορεί να υπολογιστεί για αρνητικούς αριθμούς.

   β) Είναι πάντα μικρότερη από το μηδέν.

   γ) Οι μονάδες μέτρησης είναι τετραγωνισμένες (π.χ. cm^2), δυσκολεύοντας την ερμηνεία.

   δ) Δεν λαμβάνει υπόψη όλες τις τιμές του δείγματος.

5. **Ποιο γράφημα είναι το πλέον κατάλληλο για να δούμε την κατανομή μιας ποσοτικής μεταβλητής και το σχήμα της (π.χ. αν μοιάζει με καμπάνα);**

   α) Ραβδόγραμμα

   β) Διάγραμμα Διασποράς

   γ) Ιστόγραμμα

   δ) Θερμικό Διάγραμμα Συσχέτισης

6. **Σε ένα θηκόγραμμα, τι υποδηλώνουν οι μεμονωμένες τελείες που βρίσκονται έξω από τα "μουστάκια";**

   α) Τον μέσο όρο

   β) Τις ακραίες τιμές

   γ) Τα τεταρτημόρια

   δ) Τη διάμεσο

7. **Τι μας δείχνει ο Συντελεστής Μεταβλητότητας;**

   α) Την απόλυτη διαφορά μεταξύ μεγίστου και ελαχίστου.

   β) Τη σχετική διασπορά ως ποσοστό του μέσου όρου.

   γ) Τη συσχέτιση μεταξύ δύο μεταβλητών.

   δ) Την ασυμμετρία της κατανομής.

8. **Ποια βιβλιοθήκη της Python είναι ιδανική για τη δημιουργία διαδραστικών γραφημάτων με δυνατότητα zoom και hover;**

   α) Matplotlib

   β) Seaborn

   γ) Plotly

   δ) Pandas

9. **Σε ένα θερμικό διάγραμμα συσχέτισης, τι σημαίνει αν ένα τετράγωνο έχει έντονο κόκκινο χρώμα και τιμή κοντά στο 1;**

   α) Ισχυρή θετική συσχέτιση

   β) Ισχυρή αρνητική συσχέτιση

   γ) Καμία συσχέτιση

   δ) Ότι τα δεδομένα είναι λάθος

10. **Τι σημαίνει αν μια κατανομή είναι λεπτοκυρτωμένη;**

    α) Είναι πολύ πλατιά και επίπεδη.

    β) Έχει αιχμηρή κορυφή και "παχιές" ουρές.

    γ) Είναι απόλυτα συμμετρική όπως η κανονική κατανομή.

    δ) Ότι ο μέσος όρος είναι αρνητικός.

11. **Ποιο είδος γραφήματος είναι το καταλληλότερο για την ανάλυση χρονοσειρών;**

    α) Διάγραμμα πίτας

    β) Διάγραμμα διασποράς

    γ) Γραμμικό διάγραμμα

    δ) Θηκόγραμμα

12. **Στην εντολή sns.barplot του seaborn, τι αντιπροσωπεύει η μαύρη γραμμή στην κορυφή κάθε ράβδου;**

    α) Τη μέγιστη τιμή.

    β) Το διάστημα εμπιστοσύνης.

    γ) Τον αριθμό των εγγραφών.

    δ) Τη διάμεσο.

13. **Τι μας διδάσκει το παράδειγμα του "Anscombe's Quartet";**

α) Μια τεχνική υπολογισμού του ενδοτεταρτημοριακού εύρους.

β) Ότι τέσσερα σ’υνολα δεδομένων με ίδια στατιστικά μπορεί να έχουν εντελώς διαφορετικά γραφήματα.

γ) Μια μέθοδο για τον εντοπισμό ακράιων τιμών.

δ) Έναν αλγόριθμο μηχανικής μάθησης.

14. Αν θέλουμε να συγκρίνουμε το πλήθος των ανδρών και γυναικών σε ένα δείγμα, ποιο γράφημα θα χρησιμοποιήσουμε;

α) Γραμμικό διάγραμμα

β) Ραβδόγραμμα

γ) Διάγραμμα διασποράς

δ) KDE Plot

15. Τι είναι το 1ο Τεταρτημόριο (Q1);

α) Η τιμή κάτω από την οποία βρίσκεται το 25% των δεδομένων.

β) Η τιμή κάτω από την οποία βρίσκεται το 50% των δεδομένων.

γ) Η τιμή κάτω από την οποία βρίσκεται το 75% των δεδομένων.

δ) Η ελάχιστη τιμή του δείγματος.

16. Στο project του κεφαλαίου 4.3 (Ανάλυση Κερδοφορίας), ποιο ήταν το βασικό πρόβλημα με την κατηγορία "Furniture" (Έπιπλα);

α) Είχαν πολύ χαμηλές πωλήσεις.

β) Είχαν υψηλές πωλήσεις αλλά συχνά αρνητικό ή μηδαμινό κέρδος.

γ) Δεν υπήρχαν καθόλου δεδομένα για αυτή την κατηγορία.

δ) Είχαν την υψηλότερη κερδοφορία από όλες τις κατηγορίες.

17. **Πώς ονομάζεται η τεχνική όπου εμφανίζουμε πολλά μικρά γραφήματα δίπλα-δίπλα για διαφορετικές κατηγορίες (π.χ. col='smoker');**

α) Overlaying

β) Faceting (ή Subplots)

γ) Clustering

δ) Sampling

18. **Τι θα συμβεί αν χρησιμοποιήσουμε το όρισμα hue σε ένα διάγραμμα δισποράς του seaborn;**

α) Θα μεγαλώσει το μέγεθος των σημείων.

β) Θα αλλάξει το σχήμα των σημείων.

γ) Θα χρωματίσει τα σημεία βάσει μιας κατηγορικής μεταβλητής.

δ) Θα κάνει το γράφημα τρισδιάστατο.

19. **Ποιο από τα παρακάτω δεν είναι μέτρο διασποράς;**

α) Τυπική απόκλιση

β) Διακύμανση

γ) Επικρατούσα τιμή

δ) Ενδοτεταρτημοριακό εύρος

20. **Στην Python, πώς υπολογίζουμε το 90ο εκατοστημόριο (P90) μιας στήλης df['col'];**

α) df['col'].quantile(0.90)

β) df['col'].mean() * 0.90

γ) df['col'].percentile(90)

δ) df['col'].max() – 10

4.4.1. Απαντήσεις και Επεξηγήσεις

1. β) Η διάμεσος είναι ένα ανθεκτικό μέτρο που δεν επηρεάζεται από τις ακραίες τιμές, σε αντίθεση με τον μέσο όρο.

2. β) Το ενδοτεταρτημοριακό εύρος υπολογίζεται ως Q3 - Q1 και περιγράφει την εξάπλωση του κεντρικού κορμού των δεδομένων.

3. γ) Όταν υπάρχουν μεγάλες ακραίες τιμές (δεξιά ουρά), ο μέσος όρος "τραβιέται" προς τα πάνω και ξεπερνά τη διάμεσο.

4. γ) Η τυπική απόκλιση είναι η ρίζα της διακύμανσης, επαναφέροντας τις μονάδες στην αρχική τους μορφή για να βγάζουν νόημα.

5. γ) Το ιστόγραμμα είναι το βασικό εργαλείο για την οπτικοποίηση της συχνότητας εμφάνισης τιμών.

6. β) Οι τελείες πέρα από τα όρια (fences) του θηκογράμματος αναπαριστούν τιμές που θεωρούνται στατιστικά ακραίες.

7. β) Ο συντελεστής μταβλητότητας επιτρέπει τη σύγκριση μεταβλητότητας μεταξύ δεδομένων με διαφορετικές μονάδες μέτρησης.

8. γ) Το Plotly δημιουργεί γραφήματα που λειτουργούν στον browser και επιτρέπουν την αλληλεπίδραση.

9. α) Τιμή κοντά στο 1 σημαίνει ότι οι δύο μεταβλητές αυξάνονται ταυτόχρονα.

10. β) Η θετική κύρτωση δείχνει συγκέντρωση στο κέντρο αλλά και μεγαλύτερη πιθανότητα για ακραίες τιμές.

11. γ) Το γραμμικό διάγραμμα συνδέει τα σημεία με γραμμές, κάνοντας εύκολη την αναγνώριση τάσεων στο χρόνο.

12. β) Δείχνει το εύρος μέσα στο οποίο αναμένουμε να βρίσκεται ο πραγματικός μέσος όρος (συνήθως 95%).

13. β) Αποδεικνύει ότι δεν πρέπει να βασιζόμαστε μόνο σε αριθμούς, αλλά πρέπει πάντα να οπτικοποιούμε τα δεδομένα.

14. β) Το ραβδόγραμμα είναι ιδανικό για να μετρήσουμε πόσες εγγραφές ανήκουν σε κάθε κατηγορία.

15. α) Είναι το σημείο που αφήνει πίσω του το 25% των παρατηρήσεων.

16. β) Το διάγραμμα έδειξε ότι πολλές συναλλαγές ήταν κάτω από τη γραμμή του μηδενός (ζημιές).

17. β) Το faceting είναι η δημιουργία ενός πλέγματος γραφημάτων βάσει μιας κατηγορικής μεταβλητής.

18. γ) Το hue (απόχρωση) χρησιμοποιείται για να ομαδοποιήσει τα δεδομένα με διαφορετικά χρώματα.

19. γ) Η επικρατούσα τιμή είναι μέτρο κεντρικής τάσης.

20. α) Η μέθοδος .quantile() δέχεται ποσοστό από 0 έως 1.

ΚΕΦΑΛΑΙΟ 5: ΕΠΙΣΚΟΠΗΣΗ ΒΑΣΙΚΩΝ ΑΛΓΟΡΙΘΜΩΝ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ

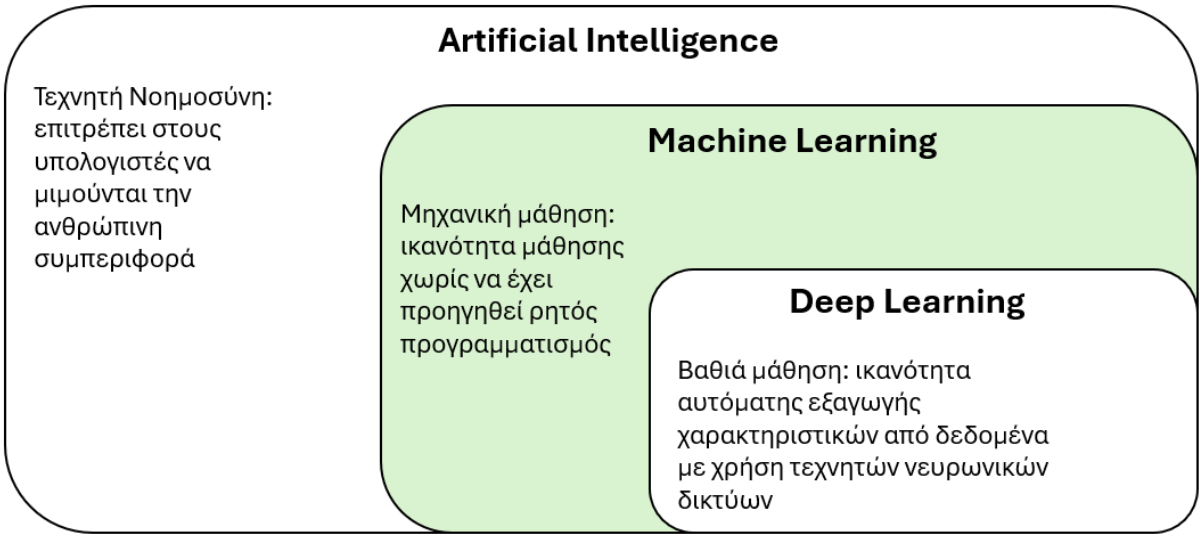
5.1. Τεχνητή νοημοσύνη, μηχανική μάθηση και βαθιά μάθηση

Η Τεχνητή Νοημοσύνη (Artificial Intelligence – AI) αποτελεί έναν ευρύ επιστημονικό και τεχνολογικό κλάδο της Πληροφορικής που μελετά και αναπτύσσει συστήματα ικανά να εκτελούν εργασίες οι οποίες, όταν εκτελούνται από ανθρώπους, απαιτούν νοημοσύνη. Τέτοιες εργασίες περιλαμβάνουν τη λήψη αποφάσεων, την επίλυση προβλημάτων, την κατανόηση φυσικής γλώσσας, την αναγνώριση εικόνων και προτύπων, καθώς και τη μάθηση που βασίζεται σε εμπειρία. Ο βασικός στόχος της Τεχνητής Νοημοσύνης δεν είναι απλώς η αυτοματοποίηση διαδικασιών, αλλά η δημιουργία υπολογιστικών συστημάτων που μπορούν να προσαρμόζονται σε νέες καταστάσεις και να εμφανίζουν συμπεριφορές που προσομοιάζουν την ανθρώπινη σκέψη.

Μέσα στο ευρύτερο πεδίο της Τεχνητής Νοημοσύνης εντάσσεται η Μηχανική Μάθηση (Machine Learning – ML), η οποία εστιάζει στη δυνατότητα των υπολογιστικών συστημάτων να μαθαίνουν από δεδομένα χωρίς να έχουν προγραμματιστεί ρητά για κάθε πιθανή περίπτωση. Αντί ο προγραμματιστής να καθορίζει αναλυτικά όλους τους κανόνες συμπεριφοράς ενός συστήματος, στη Μηχανική Μάθηση το σύστημα εκπαιδεύεται χρησιμοποιώντας παραδείγματα (δεδομένα εκπαίδευσης) και εξάγει μόνο του πρότυπα ή μοντέλα που του επιτρέπουν να κάνει προβλέψεις ή αποφάσεις για νέα, άγνωστα δεδομένα. Κλασικά παραδείγματα εφαρμογών της Μηχανικής Μάθησης είναι η πρόβλεψη τιμών, η κατηγοριοποίηση περιπτώσεων, τα συστήματα συστάσεων και ο εντοπισμός ανωμαλιών. Οι αλγόριθμοι μηχανικής μάθησης, όπως η γραμμική παλινδρόμηση, τα δέντρα αποφάσεων και οι μέθοδοι συσταδοποίησης, αποτελούν θεμέλιο για πολλές σύγχρονες εφαρμογές τεχνητής νοημοσύνης.

Η Βαθιά Μάθηση (Deep Learning – DL) αποτελεί ένα εξειδικευμένο υποσύνολο της Μηχανικής Μάθησης και βασίζεται στη χρήση τεχνητών νευρωνικών δικτύων με πολλά επίπεδα (βαθιά νευρωνικά δίκτυα). Το βασικό χαρακτηριστικό της Βαθιάς Μάθησης είναι η ικανότητα αυτόματης εξαγωγής χαρακτηριστικών απευθείας από τα ακατέργαστα δεδομένα, χωρίς να απαιτείται εκτενής ανθρώπινη παρέμβαση για το σχεδιασμό των χαρακτηριστικών. Για παράδειγμα, σε προβλήματα αναγνώρισης εικόνων, ένα βαθύ νευρωνικό δίκτυο μπορεί να μάθει μόνο του απλές δομές όπως ακμές, στη συνέχεια πιο σύνθετα σχήματα και τελικά ολόκληρα αντικείμενα. Η Βαθιά Μάθηση έχει οδηγήσει σε εντυπωσιακές προόδους σε τομείς όπως η υπολογιστική όραση, η αναγνώριση φωνής, η μετάφραση φυσικής γλώσσας και τα αυτόνομα συστήματα.

Η συσχέτιση ανάμεσα σε Τεχνητή Νοημοσύνη, Μηχανική Μάθηση και Βαθιά Μάθηση παρουσιάζεται σχηματικά στην Εικόνα 14. Στο παρόν κεφάλαιο θα περιγραφούν αλγόριθμοι που εμπίπτουν στη Μηχανική Μάθηση.



Εικόνα 14 – Συσχέτιση Τεχνητής Νοημοσύνης, Μηχανικής Μάθησης και Βαθιάς Μάθησης.

5.2. Python και μηχανική μάθηση

Η Python έχει καθιερωθεί ως μία από τις δημοφιλέστερες γλώσσες προγραμματισμού για τη Μηχανική Μάθηση, χάρη στην απλότητα της σύνταξής της, την αναγνωσιμότητα του κώδικα και το πλήθος σχετικών βιβλιοθηκών που διαθέτει. Παρέχει εργαλεία για όλα τα στάδια κατασκευής μιας εφαρμογής μηχανικής μάθησης, από τη συλλογή και προεπεξεργασία δεδομένων έως την εκπαίδευση, αξιολόγηση και χρήση μοντέλων.

5.2.1. Βιβλιοθήκες της Python για ανάλυση δεδομένων και εφαρμογή αλγορίθμων μηχανικής μάθησης

Η Python διαθέτει ένα εκτεταμένο και ώριμο οικοσύστημα βιβλιοθηκών που την καθιστά ιδιαίτερα κατάλληλη για ανάλυση δεδομένων και εφαρμογή αλγορίθμων μηχανικής μάθησης. Το Jupyter και το IPython παρέχουν διαδραστικά περιβάλλοντα που διευκολύνουν την πειραματική ανάλυση και την τεκμηρίωση της εργασίας, ενώ το NumPy αποτελεί τη βάση για αποδοτικούς αριθμητικούς υπολογισμούς. Πάνω σε αυτό το υπόβαθρο, βιβλιοθήκες όπως το Pandas και το Polars επιτρέπουν το μετασχηματισμό, τον καθαρισμό και την ανάλυση δεδομένων, με έμφαση είτε στη χρηστικότητα είτε στην υψηλή απόδοση σε μεγάλα σύνολα δεδομένων. Για την οπτικοποίηση, εργαλεία όπως τα Matplotlib, Seaborn, Plotly και Altair καλύπτουν ένα ευρύ φάσμα αναγκών, από απλά στατικά γραφήματα έως διαδραστικά dashboards και διερευνητική ανάλυση δεδομένων. Στον τομέα της μηχανικής μάθησης, η scikit-learn προσφέρει ένα συνεκτικό και εύχρηστο πλαίσιο για την υλοποίηση

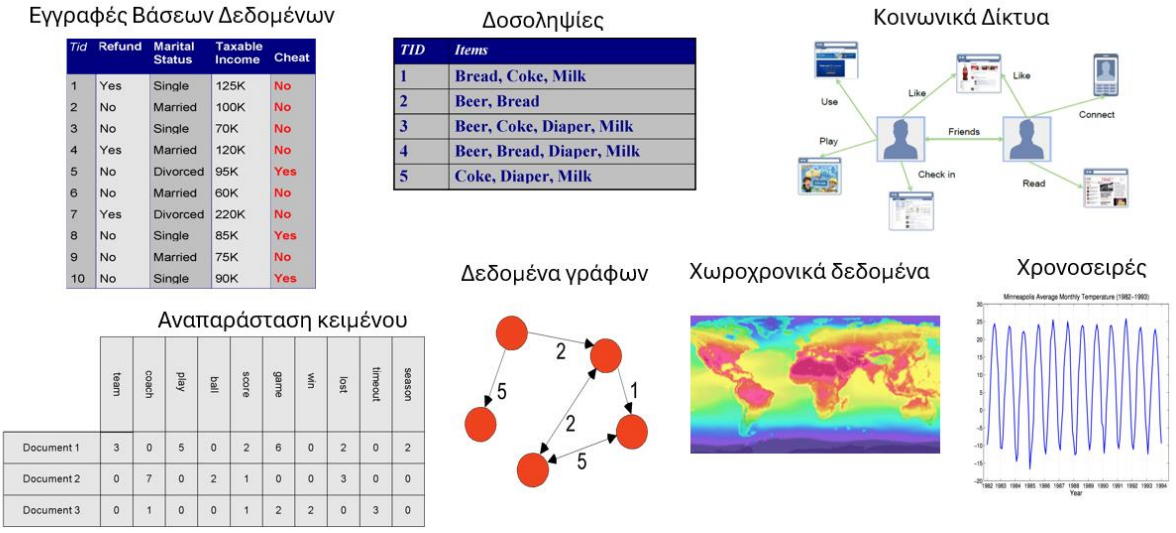
κλασικών αλγορίθμων κατηγοριοποίησης, παλινδρόμησης και συσταδοποίησης, καθώς και εργαλεία αξιολόγησης μοντέλων, ενώ βιβλιοθήκες όπως τα statsmodels εστιάζουν στη στατιστική ανάλυση και τη μοντελοποίηση. Τέλος, για πιο απαιτητικές εφαρμογές, βιβλιοθήκες υψηλής απόδοσης όπως τα XGBoost, LightGBM και CatBoost, καθώς και βιβλιοθήκες βαθιάς μάθησης όπως η PyTorch και η TensorFlow, επεκτείνουν τις δυνατότητες της Python σε εφαρμογές βαθιάς μάθησης.

5.3. Τα δεδομένα, οι τύποι και οι μορφές τους

Τα δεδομένα (data) αποτελούν τον θεμελιώδη πόρο κάθε ανάλυσης, μοντέλου ή συστήματος μηχανικής μάθησης. Τα δεδομένα είναι συλλογές από παρατηρήσεις ή αντικείμενα, όπου κάθε αντικείμενο περιγράφεται από ένα σύνολο χαρακτηριστικών (features). Κάθε χαρακτηριστικό αντιστοιχεί σε μια ιδιότητα που μπορεί να λαμβάνει τιμές με διάφορους τύπους όπως οι ονομαστικές (nominal) τιμές, οι διατεταγμένες (ordinal) τιμές και οι αριθμητικές (numerical) τιμές. Οι ονομαστικές τιμές, όπως «φύλο» ή «χώρα», απλώς διακρίνουν κατηγορίες χωρίς να υπονοείται κάποια σειρά ανάμεσα στις τιμές. Οι διατεταγμένες τιμές όπως «βαθμός ικανοποίησης», ορίζουν κάποια εννοιολογική σειρά στις τιμές (π.χ., «χαμηλός», «μέτριος», «υψηλός»). Οι αριθμητικές τιμές απλά αντιπροσωπεύουν ποσότητες που έχει νόημα να συγκριθούν μεταξύ τους καθώς και να συμμετέχουν σε υπολογισμούς.

Τα δεδομένα μπορούν να εμφανιστούν σε πολλές μορφές ανάλογα με τη δομή του προβλήματος και τη φύση του πεδίου εφαρμογής. Μια βολική μορφή, για εφαρμογές μηχανικής μάθησης, είναι τα δομημένα δεδομένα όπως είναι οι εγγραφές βάσεων δεδομένων σε πίνακες, όπου κάθε γραμμή αντιστοιχεί σε ένα αντικείμενο (π.χ. έναν πελάτη ή μια συναλλαγή) και κάθε στήλη σε ένα χαρακτηριστικό (π.χ. ηλικία, εισόδημα). Υπάρχουν όμως και άλλες κατηγορίες που ξεφεύγουν από τον απλό πίνακα. Οι δοσοληψίες (transactional data) περιγράφουν σύνολα αντικειμένων που συμβαίνουν μαζί, όπως λίστες προϊόντων σε αποδείξεις λιανικής, και χρησιμοποιούνται σε τεχνικές ανάλυσης καλαθιού αγορών (market basket analysis). Αντίστοιχα, τα κείμενα αποτελούν αδόμητα δεδομένα που πρέπει να μετατραπούν σε κατάλληλη αναπαράσταση (π.χ. πίνακες με TF-IDF ή embeddings) πριν εφαρμοστούν αλγόριθμοι μάθησης. Άλλες μορφές δεδομένων είναι τα δεδομένα γραφημάτων (graphs), όπου οι κόμβοι και οι ακμές αναπαριστούν σχέσεις μεταξύ οντοτήτων (π.χ. κοινωνικά δίκτυα), και χωροχρονικά δεδομένα, τα οποία ενσωματώνουν πληροφορία τόσο για το που, όσο και για το πότε συμβαίνει κάτι, όπως για παράδειγμα δεδομένα θερμοκρασίας στον παγκόσμιο χάρτη. Τέλος, μια ακόμα ειδική μορφή δεδομένων, που εμφανίζεται σε οικονομικά, μετεωρολογικά ή βιοϊατρικά δεδομένα είναι οι χρονοσειρές, όπου κάθε παρατήρηση αντιστοιχεί σε μια στιγμή στο χρόνο.

Κάθε μορφή δεδομένων απαιτεί κατάλληλη αναπαράσταση και προεπεξεργασία πριν εφαρμοστούν οι αλγόριθμοι μηχανικής μάθησης. Στα δομημένα δεδομένα, αυτή μπορεί να περιλαμβάνει την κωδικοποίηση κατηγορικών μεταβλητών και την κανονικοποίηση αριθμητικών. Στα κείμενα και τα γραφήματα, χρειάζονται μετασχηματισμοί όπως εξαγωγή χαρακτηριστικών. Στις χρονοσειρές, μπορεί να εφαρμοστούν τεχνικές όπως ανάλυση τάσεων και εποχικότητας πριν παραχθούν προβλέψεις. Παραδείγματα των μορφών δεδομένων που αναφέρθηκαν φαίνονται στην Εικόνα 15.



Εικόνα 15 – Παραδείγματα μορφών δεδομένων.

5.3.1.1 Αποθετήρια δεδομένων

Τα αποθετήρια δεδομένων (data repositories) αποτελούν βασική υποδομή για την ανάλυση δεδομένων και τη μηχανική μάθηση, καθώς παρέχουν ελεύθερα ή ανοικτά σύνολα δεδομένων που χρησιμοποιούνται για πειραματισμό, εκπαίδευση αλγορίθμων, σύγκριση μεθόδων και αναπαραγωγή ερευνητικών αποτελεσμάτων. Τα αποθετήρια περιλαμβάνουν δεδομένα διαφορετικών μορφών και επιπέδων δυσκολίας, από μικρά εκπαιδευτικά σύνολα έως μεγάλα πραγματικά δεδομένα, και συχνά συνοδεύονται από τεκμηρίωση, μεταδεδομένα και παραδείγματα χρήσης.

Ενδεικτικά αποθετήρια δεδομένων με κατάλληλα δεδομένα για πειραματισμό με αλγορίθμους μηχανικής μάθησης είναι τα ακόλουθα:

- Awesome Public Datasets (GitHub): https://github.com/awesomedata/awesome-public-datasets
- Calmcode datasets: https://calmcode.io/datasets
- Google Dataset Search: https://datasetsearch.research.google.com/
- Kaggle Datasets: https://www.kaggle.com/datasets
- Keras datasets: https://keras.io/api/datasets/

- OpenML: https://www.openml.org/search?type=data
- Scikit-learn Real-world datasets: https://scikit-learn.org/stable/datasets/real_world.html
- Scikit-learn Toy datasets: https://scikit-learn.org/stable/datasets/toy_dataset.html
- Seaborn datasets: https://www.kaggle.com/datasets/abdoomoh/all-seaborn-built-in-datasets
- UCI Machine Learning Repository: https://archive.ics.uci.edu/
- Κυβερνητικά δεδομένα Ε.Ε.: https://data.europa.eu/en
- Κυβερνητικά δεδομένα Η.Π.Α.: https://data.gov/

5.3.1.2 Σύνολα δεδομένων που θα χρησιμοποιηθούν στα παραδείγματα του κεφαλαίου

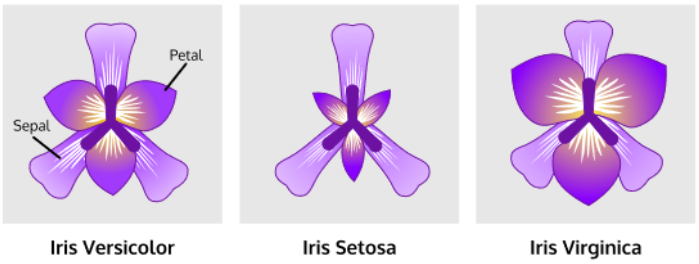
Στο πλαίσιο της εισαγωγής στη μηχανική μάθηση, είναι χρήσιμη η αξιοποίηση καθιερωμένων και καλά τεκμηριωμένων συνόλων δεδομένων. Τα σύνολα αυτά χρησιμοποιούνται ευρέως τόσο για εκπαίδευση όσο και στη διεθνή βιβλιογραφία, διευκολύνοντας την κατανόηση βασικών εννοιών όπως η κατηγοριοποίηση, η παλινδρόμηση, η προεπεξεργασία δεδομένων και η αξιολόγηση μοντέλων. Επιπλέον, επιτρέπουν τη σύγκριση διαφορετικών αλγορίθμων και την αναπαραγωγή πειραματικών αποτελεσμάτων, καθώς τα χαρακτηριστικά και η δομή τους είναι γνωστά και μελετημένα. Στα ακόλουθα σύνολα δεδομένων θα βασιστούν ορισμένα από τα παραδείγματα εφαρμογής αλγορίθμων μηχανικής μάθησης που θα παρουσιαστούν στη συνέχεια.

- **Iris data set**
  - https://en.wikipedia.org/wiki/Iris_flower_data_set
  - https://scikit-learn.org/stable/datasets/toy_dataset.html#iris-plants-dataset
  - https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
- **The Boston Housing Dataset**
  - https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html
  - https://lib.stat.cmu.edu/datasets/boston
  - https://www.kaggle.com/code/prasadperera/the-boston-housing-dataset
  - https://github.com/shridhar1504/Boston-House-Price-Prediction-Datascience-Project
- **Titanic Dataset**
  - https://seaborn.pydata.org/generated/seaborn.load_dataset.html
  - https://github.com/mwaskom/seaborn-data
  - https://www.kaggle.com/competitions/titanic
  - https://colab.research.google.com/drive/1EshABaip88itNX4vABJ1FHDmgRJSbanw

Το **Iris dataset** περιέχει περιλαμβάνει μετρήσεις από 150 φυτά ίριδας, κατανεμημένα σε τρία διαφορετικά είδη με 50 δείγματα για κάθε είδος. Για κάθε φυτό καταγράφονται τέσσερα αριθμητικά

χαρακτηριστικά (μήκος και πλάτος σέπαλου, μήκος και πλάτος πετάλου) καθώς και το όνομα του είδους στο οποίο ανήκει. Η απλή δομή και η σαφής διάκριση μεταξύ των κλάσεων καθιστούν το Iris dataset ιδιαίτερα κατάλληλο για εισαγωγικά παραδείγματα κατηγοριοποίησης, οπτικοποίησης δεδομένων και αξιολόγησης βασικών αλγορίθμων μηχανικής μάθησης.

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
...
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
...
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
...
```



Εικόνα 16 – Το σύνολο δεδομένων Iris.

Το **Boston Housing Dataset** είναι ένα κλασικό σύνολο δεδομένων παλινδρόμησης από στοιχεία της Υπηρεσίας Απογραφής των Η.Π.Α. για την περιοχή της Βοστώνης και περιλαμβάνει 506 εγγραφές με 13 αριθμητικά χαρακτηριστικά που αποτυπώνουν κοινωνικοοικονομικούς, περιβαλλοντικούς και πολεοδομικούς παράγοντες. Η μεταβλητή-στόχος είναι η διάμεση αξία κατοικιών σε χιλιάδες δολάρια. Παρά τη μακρόχρονη χρήση του στην εκπαίδευση και την έρευνα, το σύνολο δεδομένων έχει αποτελέσει αντικείμενο προβληματισμού, καθώς περιλαμβάνει μεταβλητές με κοινωνικά ευαίσθητο περιεχόμενο και ενσωματώνει ιστορικές ανισότητες που μπορεί να οδηγήσουν σε συμπεράσματα με μεροληψία. Για τον λόγο αυτό, η χρήση του σήμερα συνοδεύεται από την ανάγκη κριτικής προσέγγισης και ηθικής αξιολόγησης, αναδεικνύοντας τη σημασία της υπεύθυνης επιλογής και ερμηνείας δεδομένων στη μηχανική μάθηση.

Dataset Naming

The name for this dataset is simply **boston**. It has two prototasks: **nox**, in which the nitrous oxide level is to be predicted; and **price**, in which the median value of a home is to be predicted

Miscellaneous Details

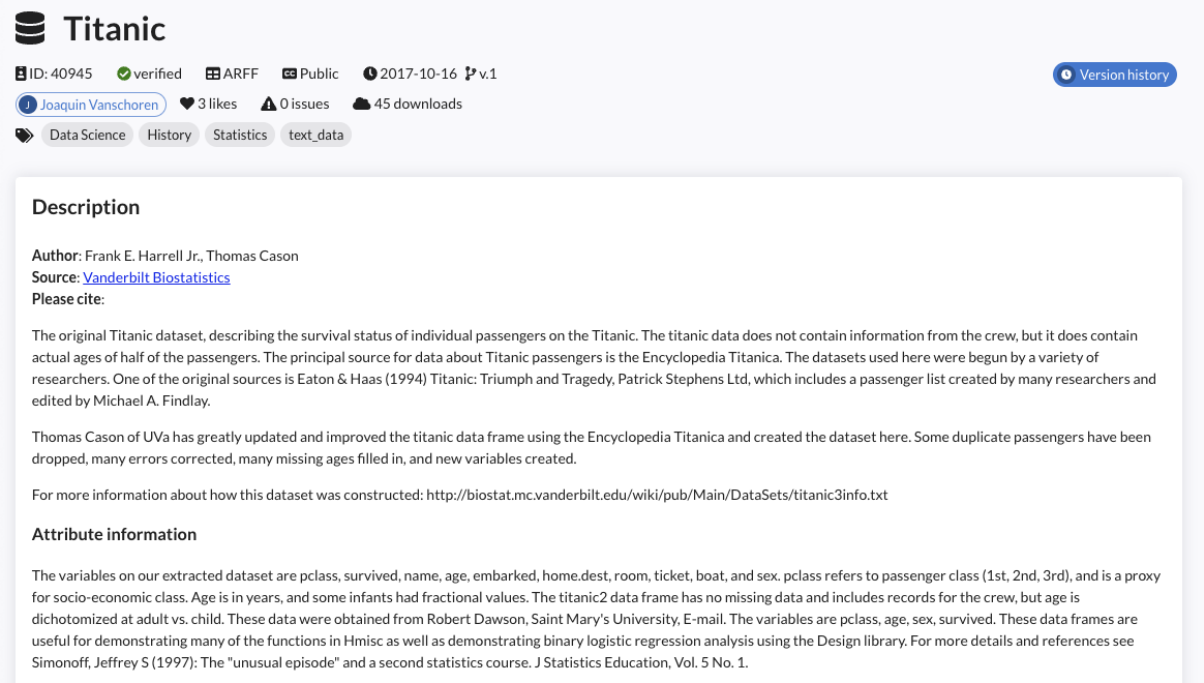
- **Origin**
  The origin of the boston housing data is **Natural**.
- **Usage**
  This dataset may be used for **Assessment**.
- **Number of Cases**
  The dataset contains a total of **506** cases.
- **Order**
  The order of the cases is **mysterious**.
- **Variables**
  There are **14** attributes in each case of the dataset. They are:
  1. CRIM - per capita crime rate by town
  2. ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
  3. INDUS - proportion of non-retail business acres per town.
  4. CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
  5. NOX - nitric oxides concentration (parts per 10 million)
  6. RM - average number of rooms per dwelling
  7. AGE - proportion of owner-occupied units built prior to 1940
  8. DIS - weighted distances to five Boston employment centres
  9. RAD - index of accessibility to radial highways
  10. TAX - full-value property-tax rate per $10,000
  11. PTRATIO - pupil-teacher ratio by town
  12. B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
  13. LSTAT - % lower status of the population
  14. MEDV - Median value of owner-occupied homes in $1000's

- **Note**
  Variable #14 seems to be censored at 50.00 (corresponding to a median price of $50,000); Censoring is suggested by the fact that the highest median price of exactly $50,000 is reported in 16 cases, while 15 cases have prices between $40,000 and $50,000, with prices rounded to the nearest hundred. Harrison and Rubinfeld do not mention any censoring.

Last Updated 10 October 1996
Comments and questions to: delve@cs.toronto.edu

Εικόνα 17 – Το σύνολο δεδομένων Boston housing.

Το **Titanic dataset** καταγράφει στοιχεία επιβατών του υπερωκεάνιου Τιτανικός. Κάθε εγγραφή αντιστοιχεί σε έναν επιβάτη και περιλαμβάνει χαρακτηριστικά όπως το φύλο, η ηλικία, η κοινωνική τάξη, το εισιτήριο και το λιμάνι επιβίβασης, με μεταβλητή-στόχο την επιβίωση ή μη μετά το ναυάγιο. Το σύνολο δεδομένων χρησιμοποιείται εκτενώς για την εισαγωγή σε προβλήματα δυαδικής ταξινόμησης, τη διαχείριση ελλιπών τιμών, την κωδικοποίηση κατηγορικών μεταβλητών και την αξιολόγηση αλγορίθμων πρόβλεψης.



Εικόνα 18 – Το σύνολο δεδομένων Titanic, με περιγραφή από το https://www.openml.org/search?type=data&sort=runs&id=40945&status=active

5.3.1.3 Εξαρτημένες μεταβλητές και ερμηνευτικές μεταβλητές

Στην μηχανική μάθηση κεντρικό ρόλο παίζουν οι εξαρτημένες μεταβλητές και οι ερμηνευτικές μεταβλητές, καθώς ορίζουν τι ακριβώς επιχειρεί να προβλέψει ή να εξηγήσει ένα μοντέλο. Η εξαρτημένη μεταβλητή είναι η ποσότητα ενδιαφέροντος, δηλαδή το αποτέλεσμα που μεταβάλλεται ως συνάρτηση άλλων μεταβλητών και αποτελεί τον στόχο της ανάλυσης. Στη βιβλιογραφία συναντάται επίσης ως μεταβλητή-στόχος, μεταβλητή απόκρισης ή ετικέτα, ενώ στα αγγλικά αναφέρεται ως dependent variable, target variable, response variable ή απλώς label.

Οι ερμηνευτικές μεταβλητές είναι οι μεταβλητές που χρησιμοποιούνται για να εξηγήσουν ή να προβλέψουν τη συμπεριφορά της εξαρτημένης μεταβλητής. Περιγράφουν τα χαρακτηριστικά των παρατηρήσεων και αποτελούν τις εισόδους του μοντέλου. Στα ελληνικά αναφέρονται επίσης ως ανεξάρτητες μεταβλητές, επεξηγηματικές μεταβλητές ή χαρακτηριστικά, ενώ στα αγγλικά ως independent variables, explanatory variables, predictors, features ή input variables. Παρότι συχνά αποκαλούνται «ανεξάρτητες», στην πράξη μπορεί να παρουσιάζουν μεταξύ τους συσχετίσεις και ο όρος δηλώνει κυρίως ότι δεν αποτελούν το αποτέλεσμα που μοντελοποιείται.

Ένα χαρακτηριστικό παράδειγμα είναι η πρόβλεψη της τιμής ενός σπιτιού. Σε αυτή την περίπτωση, η τιμή του σπιτιού αποτελεί την εξαρτημένη μεταβλητή, καθώς είναι το μέγεθος που επιδιώκεται να προβλεφθεί. Αντίθετα, μεταβλητές όπως το εμβαδόν του σπιτιού, ο αριθμός δωματίων, η ηλικία του

κτιρίου και η απόσταση από το κέντρο της πόλης λειτουργούν ως ερμηνευτικές μεταβλητές, επειδή χρησιμοποιούνται για να εξηγήσουν ή να προβλέψουν τη μεταβολή της τιμής.

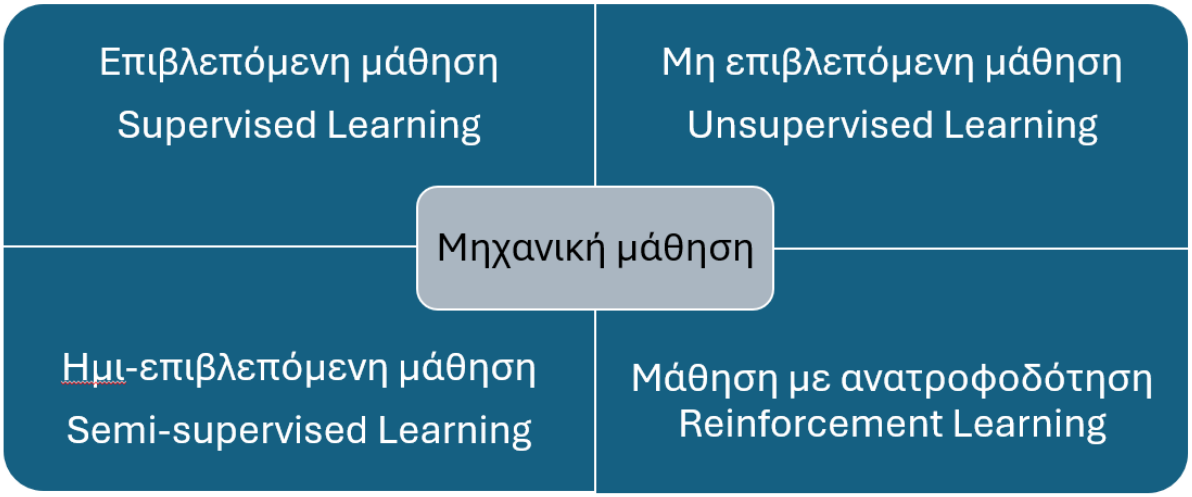
5.4. Κατηγορίες αλγορίθμων μηχανικής μάθησης

Οι αλγόριθμοι μηχανικής μάθησης ταξινομούνται σε 4 βασικές κατηγορίες (Εικόνα 19) με κριτήριο τον τρόπο με τον οποίο αξιοποιούν τα διαθέσιμα δεδομένα και το μηχανισμό μάθησης που ακολουθούν. Η πιο διαδεδομένη κατηγορία είναι η επιβλεπόμενη μάθηση (supervised learning), όπου το μοντέλο εκπαιδεύεται σε σύνολα δεδομένων που περιλαμβάνουν τόσο τα χαρακτηριστικά εισόδου όσο και τη σωστή έξοδο (ετικέτα). Στην κατηγορία αυτή ανήκουν προβλήματα όπως η πρόβλεψη κατηγορίας ή συνεχούς τιμής.

Η μη επιβλεπόμενη μάθηση (unsupervised learning) εφαρμόζεται όταν τα δεδομένα δεν συνοδεύονται από ετικέτες και ο στόχος είναι η ανακάλυψη κρυφών δομών ή προτύπων. Τυπικά παραδείγματα αποτελούν οι αλγόριθμοι που προσπαθούν να εντοπίσουν ομάδες παρόμοιων αντικειμένων ή να συνοψίσουν την πληροφορία δεδομένων με λιγότερες μεταβλητές. Η κατηγορία αυτή είναι ιδιαίτερα χρήσιμη στη διερευνητική ανάλυση δεδομένων.

Η ημι-επιβλεπόμενη μάθηση (semi-supervised learning) βρίσκεται μεταξύ των δύο προηγούμενων προσεγγίσεων και αξιοποιεί σύνολα δεδομένων στα οποία μόνο ένα μικρό μέρος των παρατηρήσεων έχει επισημάνσεις σωστής εξόδου. Χρησιμοποιούνται όταν η πλήρης επισήμανση είναι δαπανηρή ή δύσκολη.

Τέλος, η μάθηση με ανατροφοδότηση (reinforcement learning) βασίζεται στη διαδραστική μάθηση ενός πράκτορα (agent) σε ένα περιβάλλον, όπου οι αποφάσεις του αξιολογούνται μέσω ανταμοιβών ή ποινών. Στόχος είναι η εκμάθηση μιας στρατηγικής που μεγιστοποιεί τη συνολική ανταμοιβή με την πάροδο του χρόνου. Η προσέγγιση αυτή χρησιμοποιείται σε προβλήματα λήψης αποφάσεων, όπως η ρομποτική, τα παιχνίδια και τα αυτόνομα συστήματα.



Εικόνα 19 – Βασικές κατηγορίες αλγορίθμων μηχανικής μάθησης.

Στη συνέχεια του κεφαλαίου θα εξεταστούν μόνο αλγόριθμοι επιβλεπόμενης μάθησης και αλγόριθμοι μη-επιβλεπόμενης μάθησης.

5.4.1. Περισσότερα για επιβλεπόμενη μάθηση και μη-επιβλεπόμενη μάθηση

Στην επιβλεπόμενη μάθηση, το μοντέλο μαθαίνει από παραδείγματα στα οποία η σωστή απάντηση είναι ήδη γνωστή. Τα δεδομένα εκπαίδευσης εμφανίζονται συνήθως σε ζεύγη (X, y), όπου το X αντιστοιχεί στις ερμηνευτικές μεταβλητές και το y στην εξαρτημένη μεταβλητή. Σε ορισμένα προβλήματα μπορεί να υπάρχουν περισσότερες από μία εξαρτημένες μεταβλητές.

Τα προβλήματα επιβλεπόμενης μάθησης διακρίνονται κυρίως στην παλινδρόμηση (regression) και στην κατηγοριοποίηση (classification). Στην παλινδρόμηση, ο στόχος είναι η πρόβλεψη αριθμητικών, συνεχών τιμών, όπως για παράδειγμα η εκτίμηση της τιμής ενός ακινήτου με βάση χαρακτηριστικά όπως το εμβαδόν ή η τοποθεσία. Τυπικοί αλγόριθμοι παλινδρόμησης είναι η γραμμική και η πολυωνυμική παλινδρόμηση, καθώς και τα νευρωνικά δίκτυα. Αντίθετα, στην κατηγοριοποίηση ο στόχος είναι η ανάθεση διακριτών ετικετών στις παρατηρήσεις, όπως η ταξινόμηση ενός email ως κανονικό ή ανεπιθύμητο (spam). Στην κατηγορία αυτή ανήκουν αλγόριθμοι όπως τα δέντρα αποφάσεων, οι k-πλησιέστεροι γείτονες (k-NN), η λογιστική παλινδρόμηση, ο Naive Bayes, τα Support Vector Machines (SVM), τα νευρωνικά δίκτυα κ.α.

Στη μη επιβλεπόμενη μάθηση (unsupervised learning), τα δεδομένα εκπαίδευσης δεν περιλαμβάνουν εξαρτημένη μεταβλητή. Ο αλγόριθμος δεν γνωρίζει εκ των προτέρων «σωστές απαντήσεις», αλλά προσπαθεί να εντοπίσει δομές, πρότυπα ή σχέσεις που υπάρχουν στα δεδομένα. Μία βασική υποκατηγορία μη επιβλεπόμενης μάθησης είναι η συσταδοποίηση (clustering), όπου στόχος είναι η ομαδοποίηση παρατηρήσεων με βάση την ομοιότητά τους. Ένα χαρακτηριστικό παράδειγμα είναι η ομαδοποίηση πελατών μιας επιχείρησης σύμφωνα με τη συμπεριφορά ή τα χαρακτηριστικά τους.

Δημοφιλείς αλγόριθμοι συσταδοποίησης είναι οι K-Means, η ιεραρχική συσταδοποίηση και ο DBSCAN.

Μια δεύτερη σημαντική υποκατηγορία της μη επιβλεπόμενης μάθησης είναι η μείωση διαστάσεων (dimension reduction). Σε αυτή την περίπτωση, στόχος είναι η απεικόνιση δεδομένων από έναν χώρο υψηλής διάστασης σε έναν χώρο μικρότερης διάστασης, διατηρώντας όσο το δυνατόν περισσότερη από τη σημαντική πληροφορία. Η μείωση διαστάσεων χρησιμοποιείται συχνά για οπτικοποίηση δεδομένων με πολλά χαρακτηριστικά ή για απλοποίηση προβλημάτων πριν την εφαρμογή άλλων αλγορίθμων. Κλασικές τεχνικές είναι η Ανάλυση Κύριων Συνιστωσών (PCA) και η μέθοδος t-SNE.

5.5. Παλινδρόμηση

Η παλινδρόμηση (regression) χρησιμοποιείται για τη μελέτη και μοντελοποίηση της σχέσης μεταξύ μιας εξαρτημένης μεταβλητής και μίας ή περισσότερων ερμηνευτικών μεταβλητών. Στόχος της είναι η κατανόηση του τρόπου με τον οποίο οι μεταβολές στις ερμηνευτικές μεταβλητές επηρεάζουν τη μεταβλητή-στόχο. Η παλινδρόμηση αφορά συνεχή μεγέθη, όπως ποσότητες ή μετρήσεις. Στο πλαίσιο της ανάλυσης δεδομένων με την Python, οι μέθοδοι παλινδρόμησης παρέχουν ένα ισχυρό και ευέλικτο εργαλείο τόσο για περιγραφική ερμηνεία όσο και για κατασκευή προβλεπτικών μοντέλων, αποτελώντας συχνά το πρώτο βήμα στη διερεύνηση ποσοτικών σχέσεων μέσα σε ένα σύνολο δεδομένων.

5.5.1. Γραμμική παλινδρόμηση

Η γραμμική παλινδρόμηση (linear regression) υποθέτει ότι η εξαρτημένη μεταβλητή μπορεί να εκφραστεί ως γραμμικός συνδυασμός ενός συνόλου ερμηνευτικών μεταβλητών. Για p ερμηνευτικές μεταβλητές, αυτό μπορεί να περιγραφεί μαθηματικά ως εξής:

$$y = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_p x_p + \varepsilon$$

όπου b_0 είναι ο σταθερός όρος, $b_1, \ldots, b_p$ είναι οι συντελεστές παλινδρόμησης και ε είναι το σφάλμα.

Κατά τη διαδικασία προσαρμογής του μοντέλου (fitting), επιδιώκουμε να προσδιορίσουμε τις τιμές των συντελεστών έτσι ώστε η γραμμική συνάρτηση να προσεγγίζει όσο το δυνατόν καλύτερα τις παρατηρήσεις. Η συνηθέστερη μέθοδος για να επιτευχθεί αυτό είναι η μέθοδος των ελαχίστων τετραγώνων (OLS = Ordinary Least Squares). Βασική της ιδέα είναι ο προσδιορισμός της ευθείας (στην απλή παλινδρόμηση) ή του υπερεπιπέδου (στην πολλαπλή παλινδρόμηση) που προσαρμόζεται στα δεδομένα με τέτοιο τρόπο ώστε να ελαχιστοποιείται το συνολικό σφάλμα πρόβλεψης. Το σφάλμα αυτό μετριέται ως η διαφορά μεταξύ των πραγματικών τιμών y_i και των τιμών που προβλέπονται $\hat{y}_i$

και συγκεκριμένα ως το άθροισμα των τετραγώνων αυτών των διαφορών. Τυπικά, η OLS ελαχιστοποιεί το άθροισμα των τετραγώνων των σφαλμάτων (SSE = Sum of Squared Errors):

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Οι παράμετροι $b_0, \dots, b_p$ επιλέγονται έτσι ώστε το SSE να λαμβάνει την ελάχιστη δυνατή τιμή. Για την ορθότητα της OLS, η θεωρία προβλέπει ότι θα πρέπει να ισχύουν ορισμένες προϋποθέσεις όπως γραμμικότητα, ανεξαρτησία παρατηρήσεων, σταθερή διακύμανση (ομοσκεδαστικότητα) σφαλμάτων και τέλος τα σφάλματα να ακολουθούν την κανονική κατανομή. Για τον υπολογισμό των τιμών των συντελεστών b τελικά λύνεται ένα σύστημα εξισώσεων.

Στην Python, η υλοποίηση της γραμμικής παλινδρόμησης είναι ιδιαίτερα απλή χάρη στη βιβλιοθήκη scikit-learn, η οποία παρέχει την κλάση LinearRegression και χρησιμοποιεί τις μεθόδους fit() και predict(). Η διαδικασία περιλαμβάνει τον ορισμό του μοντέλου, την προσαρμογή του στα δεδομένα εκπαίδευσης και στη συνέχεια την παραγωγή προβλέψεων για νέα δεδομένα. Εναλλακτικά, για μια πιο στατιστικά προσανατολισμένη ανάλυση, όπου ενδιαφέρει όχι μόνο η πρόβλεψη αλλά και η στατιστική ερμηνεία των συντελεστών (p-values, διαστήματα εμπιστοσύνης, έλεγχοι υποθέσεων), μπορεί να χρησιμοποιηθεί η βιβλιοθήκη statsmodels.

5.5.2. Απλή γραμμική παλινδρόμηση

Στην ειδική περίπτωση όπου το μοντέλο περιλαμβάνει μία μόνο ερμηνευτική μεταβλητή και μία εξαρτημένη μεταβλητή, μιλάμε για απλή γραμμική παλινδρόμηση (simple linear regression). Το μοντέλο περιγράφεται από την εξίσωση:

$$y = b_0 + b_1 x_1 + \varepsilon$$

όπου b_0 είναι ο σταθερός όρος (intercept), b_1 ο συντελεστής κλίσης (slope) και ε το σφάλμα. Ο συντελεστής b_1εκφράζει τη μεταβολή της εξαρτημένης μεταβλητής y για κάθε μονάδα μεταβολής της ερμηνευτικής μεταβλητής x, γεγονός που καθιστά το μοντέλο ιδιαίτερα εύκολο στην ερμηνεία.

Η οπτικοποίηση της απλής γραμμικής παλινδρόμησης είναι άμεση και διαισθητική, καθώς εμπλέκονται μόνο δύο μεταβλητές και το μοντέλο μπορεί να αναπαρασταθεί σε δισδιάστατο καρτεσιανό σύστημα συντεταγμένων. Τα δεδομένα απεικονίζονται συνήθως ως διάγραμμα διασποράς (scatter plot), ενώ η εκτιμημένη ευθεία παλινδρόμησης προστίθεται στο ίδιο γράφημα, επιτρέποντας την οπτική αξιολόγηση της προσαρμογής. Στο πλαίσιο της ανάλυσης δεδομένων με την Python, εργαλεία όπως το matplotlib ή το seaborn διευκολύνουν ιδιαίτερα αυτή τη διαδικασία,

επιτρέποντας την ταυτόχρονη παρουσίαση των δεδομένων και της ευθείας παλινδρόμησης με ελάχιστο κώδικα.

Το ακόλουθο παράδειγμα δείχνει τη διαδικασία εφαρμογής της απλής γραμμικής παλινδρόμησης σε συνθετικά δεδομένα, με χρήση της βιβλιοθήκης scikit-learn. Ο κώδικας παρουσιάζεται σε τμήματα κώδικα, που αντιστοιχούν στα κελιά ενός jupyter σημειωματαρίου μαζί με το αποτέλεσμα της εκτέλεσης κάθε τμήματος.

Αρχικά, στον κώδικα Κ. 5.5.1 δημιουργείται ένα τεχνητό σύνολο δεδομένων. Ορίζεται ως seed μια συγκεκριμένη τιμή (np.random.seed(42)) ώστε τα αποτελέσματα να μπορούν να αναπαραχθούν. Στη συνέχεια, κατασκευάζεται ένα διάνυσμα τιμών x στο διάστημα [0, 10] και παράγονται οι αντίστοιχες τιμές y σύμφωνα με τη γραμμική σχέση:

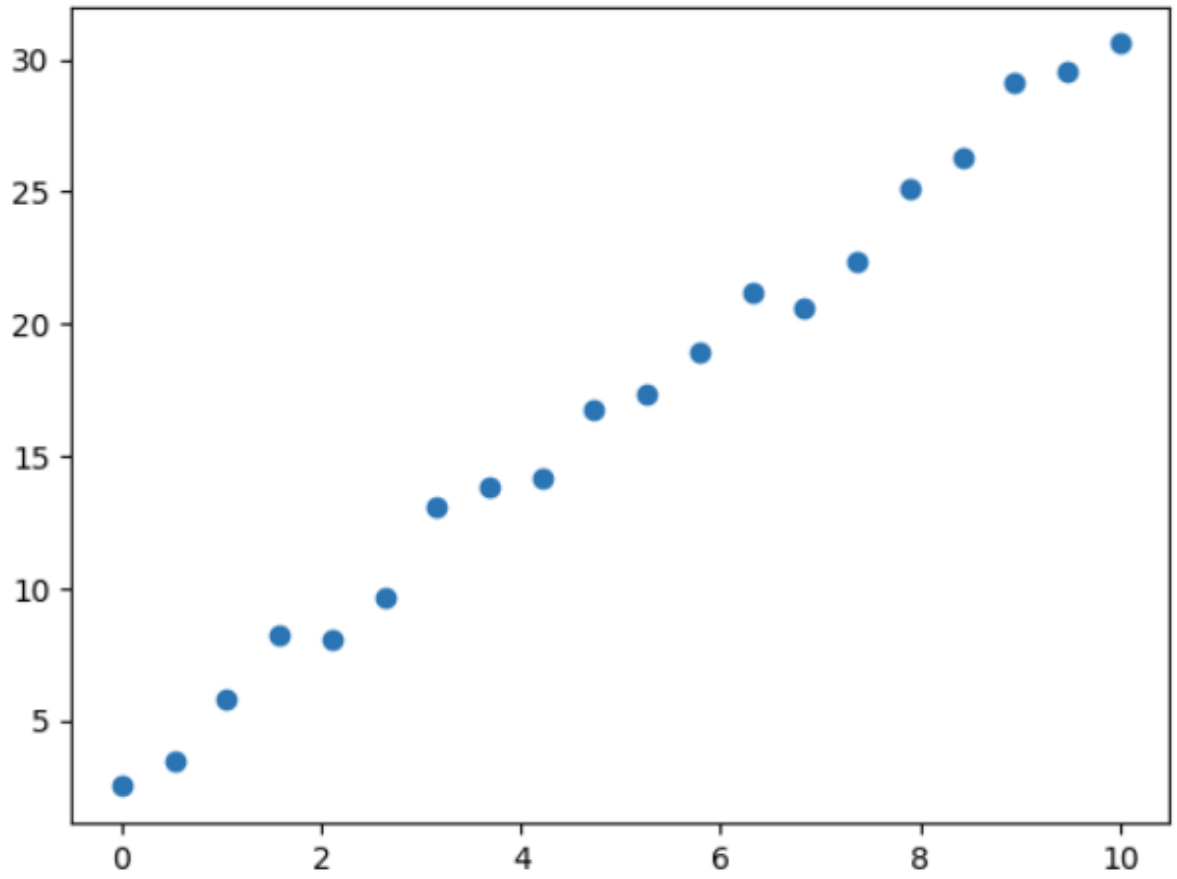
$$y = 3x + 2 + \varepsilon$$

όπου ε είναι ένας τεχνητός θόρυβος που δημιουργείται με την np.random.randn(). Με αυτό τον τρόπο προσομοιώνεται ένα σενάριο όπου η πραγματική σχέση είναι γραμμική, αλλά οι παρατηρήσεις περιέχουν τυχαίες αποκλίσεις.

```
import numpy as np
import matplotlib.pyplot as plt
import sklearn

# Δημιουργία συνθετικών δεδομένων
np.random.seed(42)
n = 20
x = np.linspace(0, 10, n)
y = 3 * x  + 2 + np.random.randn(n)
plt.scatter(x, y)
```

Κ. 5.5.1 – Δημιουργία συνθετικών δεδομένων με προσθήκη θορύβου.

Το διάγραμμα διασποράς (scatter plot) που εμφανίζεται στην Εικόνα 20 επιβεβαιώνει οπτικά την σχεδόν γραμμική τάση των δεδομένων.



Εικόνα 20 – Σημεία της ευθείας γραμμής y = 3x +2, στα οποία έχει προστεθεί θόρυβος.

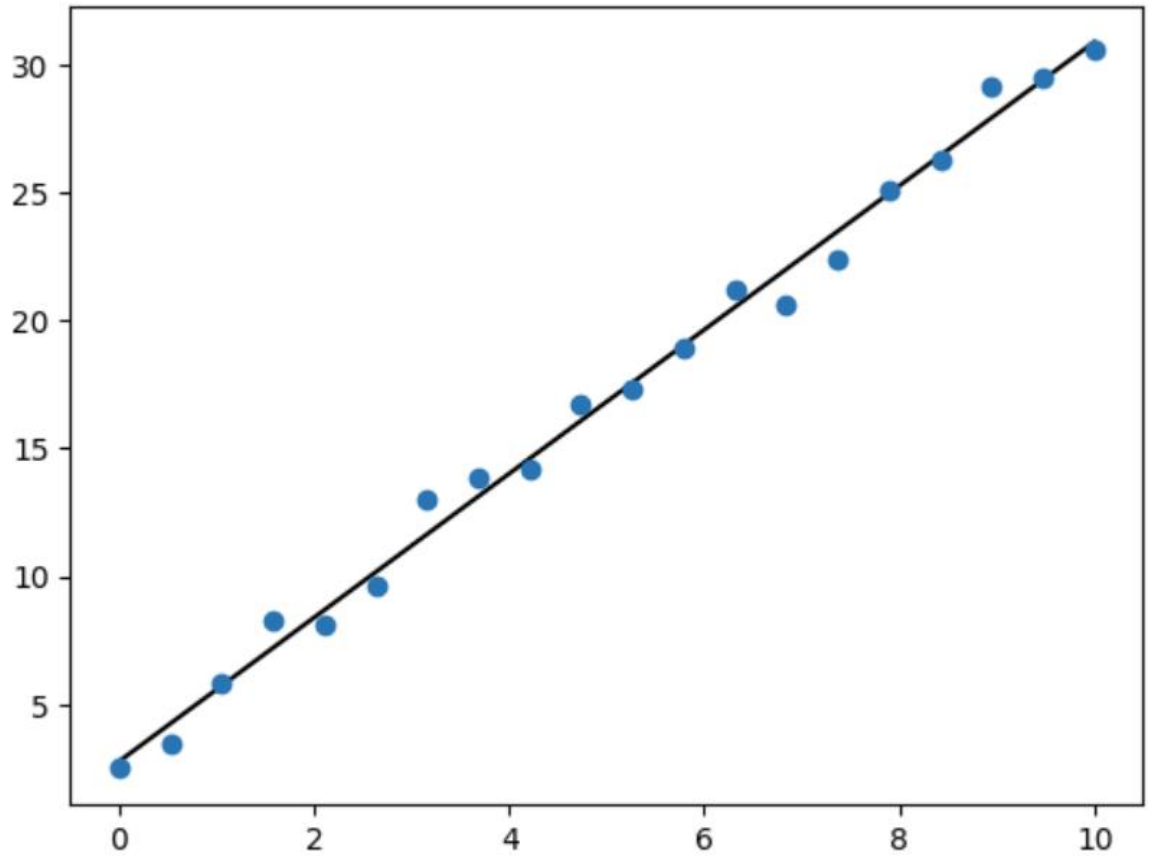
Ιδιαίτερη προσοχή πρέπει να δοθεί στη μορφή των δεδομένων εισόδου που θα χρησιμοποιηθούν καθώς το scikit-learn απαιτεί η ανεξάρτητη μεταβλητή να έχει μορφή πίνακα δύο διαστάσεων (μορφή $n \times p$), ακόμη και όταν υπάρχει μία μόνο ερμηνευτική μεταβλητή. Για το λόγο αυτό στον κώδικα Κ. 5.5.2 γίνεται μετασχηματισμός των δεδομένων της μεταβλητής x από διάνυσμα n στοιχείων σε πίνακα-στήλη, n x 1. Αυτό συμβαίνει με τη χρήση του x[:, np.newaxis], στη θέση του x. Το x[:, np.newaxis] σημαίνει λήψη όλων των στοιχείων του διανύσματος x και προσθήκη μιας νέας δεύτερης διάστασης. Αξίζει να σημειωθεί ότι το ίδιο αποτέλεσμα θα είχε επιτευχθεί με χρήση της εντολής np.vstack(x).T καθώς και με την x.reshape(1, -1). Στον κώδικα Κ. 5.5.2 γίνεται εισαγωγή της κλάσης LinearRegression από το υποπακέτο sklearn.linear_model και δημιουργείται ένα αντικείμενο μοντέλου. Η μέθοδος fit() καλείται με ορίσματα το x[:, np.newaxis] και το διάνυσμα y, υλοποιώντας στην πράξη τη μέθοδο OLS. Με τον τρόπο αυτό υπολογίζονται οι εκτιμήσεις των παραμέτρων b_0 και b_1, οι οποίες ελαχιστοποιούν το άθροισμα τετραγώνων των σφαλμάτων. Στη συνέχεια, χρησιμοποιείται η μέθοδος predict() για τον υπολογισμό των προβλεπόμενων τιμών $\hat{y}$, ώστε να σχεδιαστεί η γραμμή παλινδρόμησης.

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
```

```
model.fit(x[:, np.newaxis], y)
xfit = x
yfit = model.predict(x[:, np.newaxis])
plt.plot(xfit, yfit, color="black")
plt.plot(x, y, "o")
plt.show()
```

Κ. 5.5.2 – Προσαρμογή και πρόβλεψη.

Το αποτέλεσμα παρουσιάζεται στην Εικόνα 21.



Εικόνα 21 – Γραμμή παλινδρόμησης.

Τέλος, με τον κώδικα Κ. 5.5.3 εκτυπώνονται οι τιμές των παραμέτρων που έχουν υπολογιστεί.

```
# Εκτύπωση συντελεστών της γραμμής παλινδρόμησης
print("Παράμετροι:", model.coef_, model.intercept_)
slope = model.coef_[0]
intercept = model.intercept_
print(f"y={slope:.2f}x + {intercept:.2f}")
```

Κ. 5.5.3 – Εκτύπωση συντελεστών παλινδρόμησης.

Τα αποτελέσματα που εκτυπώνονται είναι:

```
Παράμετροι: [2.81082696] 2.7745666335933983
y=2.81x + 2.77
```

Οι τιμές 2.81 και 2.77 είναι κοντά στις τιμές 3 και 2 αντίστοιχα, στο y = 3x+2, που χρησιμοποιήθηκαν κατά τη δημιουργία των αρχικών δεδομένων, χωρίς την προσθήκη θορύβου, και δείχνουν τη ικανότητα της γραμμικής παλινδόμησης να ανιχνεύει την υπάρχουσα τάση στα δεδομένα εισόδου.

5.5.3. Γραμμική παλινδρόμηση με πολλαπλές ερμηνευτικές μεταβλητές

Όταν το μοντέλο περιλαμβάνει περισσότερες από μία ερμηνευτικές μεταβλητές, τότε μιλάμε για πολλαπλή γραμμική παλινδρόμηση (*multiple linear regression*). Στην περίπτωση αυτή, η εξαρτημένη μεταβλητή y εκφράζεται ως γραμμικός συνδυασμός πολλών χαρακτηριστικών $x_1, x_2, \dots, x_p$. Το μοντέλο γράφεται στη γενική του μορφή ως:

$$y = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_p x_p + \varepsilon,$$

όπου b_0 είναι ο σταθερός όρος, $b_1, \dots, b_p$οι συντελεστές παλινδρόμησης και ε το σφάλμα. Κάθε συντελεστής εκφράζει τη μεταβολή της εξαρτημένης μεταβλητής όταν η αντίστοιχη ερμηνευτική μεταβλητή αυξάνεται κατά μία μονάδα, διατηρώντας τις υπόλοιπες σταθερές. Η εκτίμηση των παραμέτρων πραγματοποιείται με τον ίδιο ακριβώς τρόπο όπως και στην απλή γραμμική παλινδρόμηση, δηλαδή μέσω της μεθόδου Ordinary Least Squares, η οποία ελαχιστοποιεί το άθροισμα των τετραγώνων των σφαλμάτων. Η βασική διαφορά έγκειται στη δομή των δεδομένων: οι ερμηνευτικές μεταβλητές δεν αποτελούν πλέον ένα απλό διάνυσμα αλλά έναν πίνακα διαστάσεων $m \times p$, όπου m είναι το πλήθος των παρατηρήσεων (δειγμάτων) και p το πλήθος των χαρακτηριστικών.

Η υλοποίηση της πολλαπλής γραμμικής παλινδρόμησης με τη βιβλιοθήκη scikit-learn είναι πανομοιότυπη με εκείνη της απλής παλινδρόμησης. Η κλάση LinearRegression δέχεται ως είσοδο έναν δισδιάστατο πίνακα χαρακτηριστικών και ένα διάνυσμα στόχο, ενώ η χρήση των μεθόδων fit() και predict() παραμένουν οι ίδιες. Η ενιαία αυτή διεπαφή καθιστά ιδιαίτερα εύκολη τη μετάβαση από μοντέλα με ένα χαρακτηριστικό σε μοντέλα με πολλαπλές μεταβλητές.

Ως ένα απλό παράδειγμα γραμμικής παλινδρόμησης με 2 ερμηνευτικές μεταβλητές θα λυθεί ένα πρόβλημα εκτίμησης κόστους που αφορά δύο προϊόντα Α και Β για τα οποία είναι γνωστές οι ακόλουθες αξίες διαφόρων συνδυασμών ποσοτήτων τους:

- 0 μονάδες προϊόντος Α και 0 μονάδες προϊόντος Β, έχουν αξία 0 ευρώ.
- 1 μονάδα προϊόντος Α και 1 μονάδα προϊόντος Β, έχουν αξία 500 ευρώ.
- 2 μονάδες προϊόντος Α και 3 μονάδες προϊόντος Β, έχουν αξία 1150 ευρώ.
- 3 μονάδες προϊόντος Α και 2 μονάδες προϊόντος Β έχουν αξία 1090 ευρώ.
- 5 μονάδες προϊόντος Α και 6 μονάδες προϊόντος Β έχουν αξία 2250 ευρώ.

Έστω ότι το ζητούμενο είναι μια εκτίμηση για την αξία 4 μονάδων του προϊόντος Α και 5 μονάδων του προϊόντος Β.

Ο κώδικας Κ. 5.5.4 δίνει μια λύση στο πρόβλημα με τη χρήση γραμμικής παλινδρόμησης. Τα δεδομένα οργανώνονται σε έναν πίνακα ερμηνευτικών μεταβλητών X όπου κάθε γραμμή περιέχει ένα ζεύγος ποσοτήτων, και σε ένα διάνυσμα στόχο y που περιλαμβάνει τις αντίστοιχες τιμές κόστους. Με τη χρήση της predict([[4, 5]]) υπολογίζεται η εκτιμώμενη αξία για 4 μονάδες του προϊόντος Α και 5 μονάδες του προϊόντος Β. Η τιμή που προκύπτει (1891.05 ευρώ) αποτελεί την πρόβλεψη του γραμμικού μοντέλου, δηλαδή την καλύτερη γραμμική εκτίμηση του κόστους με βάση τα δοθέντα παραδείγματα.

```
from sklearn.linear_model import LinearRegression

X = [[0, 0], [1, 1], [2, 3], [3, 2], [5, 6]]
y = [0, 500, 1150, 1090, 2250]

model = LinearRegression()
model.fit(X, y)
yfit = model.predict([[4, 5]])
print(f"Προβλεπόμενη αξία {yfit[0]:.2f}") # Προβλεπόμενη αξία 1891.05
```

Κ. 5.5.4 – Εντοπισμός προβλεπόμενης αξίας συνδυασμού προϊόντων με γραμμική παλινδρόμηση.

5.5.4. Πολυωνυμική παλινδρόμηση

Σε πολλές πραγματικές εφαρμογές, η σχέση μεταξύ των ερμηνευτικών μεταβλητών και της εξαρτημένης μεταβλητής δεν μπορεί να περιγραφεί ικανοποιητικά από ένα απλό γραμμικό μοντέλο της μορφής $y = ax + b$. Όταν τα δεδομένα εμφανίζουν καμπυλότητα ή πιο σύνθετη δομή, ένα γραμμικό υπόδειγμα ενδέχεται να παρουσιάζει συστηματικά σφάλματα προσαρμογής. Σε τέτοιες περιπτώσεις, μπορεί να χρησιμοποιηθεί η πολυωνυμική παλινδρόμηση (*polynomial regression*), η οποία επιτρέπει την εισαγωγή μη γραμμικών όρων ως προς τις μεταβλητές εισόδου.

Η βασική ιδέα είναι ότι, παρότι το μοντέλο παραμένει γραμμικό ως προς τις παραμέτρους, περιλαμβάνει δυνάμεις των ερμηνευτικών μεταβλητών. Για παράδειγμα, ένα πολυώνυμο δευτέρου βαθμού γράφεται ως:

$$y = ax^2 + bx + c,$$

ενώ ένα πολυώνυμο τρίτου βαθμού ως:

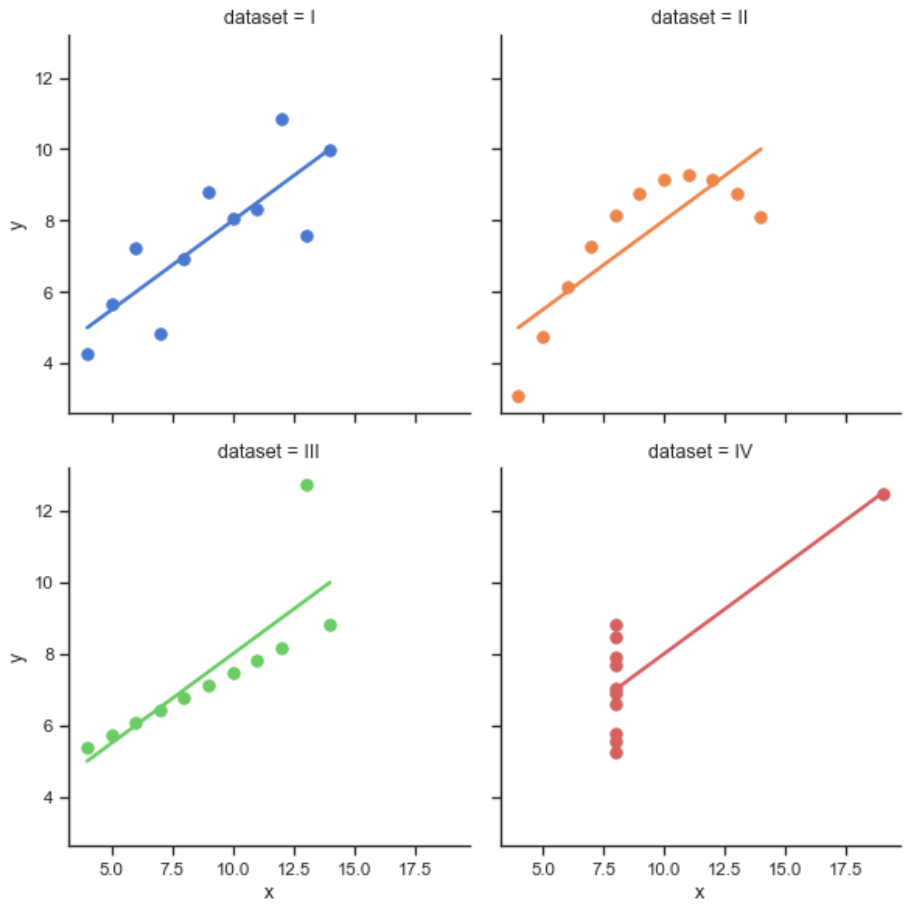
$$y = ax^3 + bx^2 + cx + d.$$

Με αυτόν τον τρόπο, το μοντέλο αποκτά μεγαλύτερη ευελιξία και μπορεί να προσαρμοστεί σε καμπύλες τάσεις των δεδομένων. Σημαντικό είναι ότι η εκτίμηση των παραμέτρων εξακολουθεί να

γίνεται με τη μέθοδο των Ελαχίστων Τετραγώνων, καθώς το πρόβλημα παραμένει γραμμικό ως προς τους συντελεστές (παραμέτρους).

Στην Python, η υλοποίηση της πολυωνυμικής παλινδρόμησης γίνεται συνήθως σε δύο στάδια. Αρχικά, χρησιμοποιείται ο μετασχηματισμός χαρακτηριστικών μέσω της κλάσης PolynomialFeatures από το υποπακέτο sklearn.preprocessing, ο οποίος δημιουργεί αυτόματα τους πολυωνυμικούς όρους (π.χ. x^2, x^3, ή και γινόμενα μεταβλητών σε περίπτωση πολλαπλών χαρακτηριστικών). Στη συνέχεια, εφαρμόζεται ένα κλασικό μοντέλο γραμμικής παλινδρόμησης. Η διαδικασία αυτή επιτρέπει τη μοντελοποίηση μη γραμμικών σχέσεων, διατηρώντας ταυτόχρονα την υπολογιστική απλότητα και τη θεωρητική βάση της γραμμικής παλινδρόμησης.

Ως ένα παράδειγμα των δυνατοτήτων της πολυωνυμικής παλινδρόμησης θα παρουσιαστεί η προαρμογή που μπορεί να επιτευχθεί στα δεδομένα του dataset II από το Anscombe quartet. Μια απεικόνιση των δεδομένων των Anscombe datasets (I, II, III, IV) φαίνεται στην Εικόνα 22, δείχνοντας ότι ενώ τα δεδομένα έχουν διαφορετικές μορφές, η απλή γραμμική παλινδρόμηση, θα επιστρέψει την ίδια γραμμή παλινδρόμησης και στις τέσσερις περιπτώσεις.



Εικόνα 22 – Διαφορετικές μορφές δεδομένων, αλλά ίδια γραμμή παλινδρόμησης.

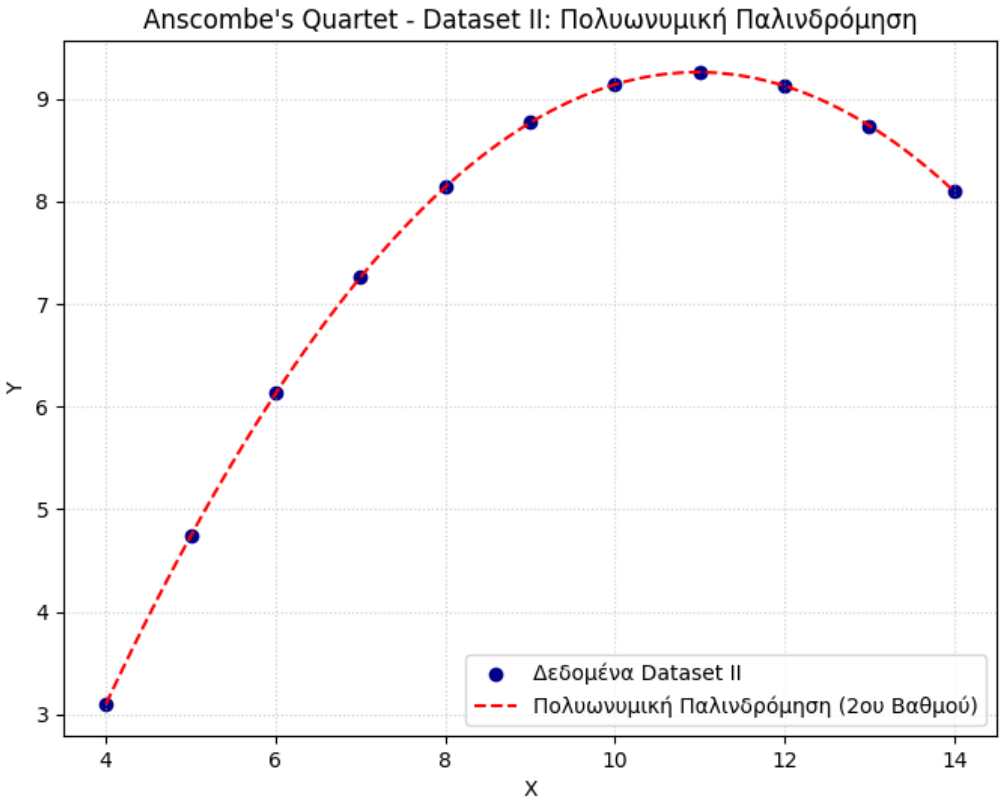
Ο κώδικας Κ. 5.5.5 δείχνει την εφαρμογή της πολυωνυμικής παλινδρόμησης στο dataset II.

```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# 1. Φόρτωση και επιλογή του Dataset II
df = sns.load_dataset("anscombe")
df_ii = df[df["dataset"] == "II"]
X = df_ii[["x"]]  # Ερμηνευτική μεταβλητή
y = df_ii["y"]  # Εξαρτημένη μεταβλητή

# 2. Μετασχηματισμός χαρακτηριστικών (δημιουργία του x^2)
# Καθορίζουμε το βαθμό (degree) 2 για να συμπεριλάβουμε όρους x και x^2
poly_features = PolynomialFeatures(degree=2, include_bias=False)
# Το fit_transform δημιουργεί ένα νέο πίνακα με τις στήλες [x, x^2]
X_poly = poly_features.fit_transform(X)

# 3. Εφαρμογή γραμμικής παλινδρόμησης στο μετασχηματισμένο πίνακα
# Πολυωνυμική παλινδρόμηση: Y = a + b1*x + b2*x^2
model = LinearRegression()
model.fit(X_poly, y)

# 4. Οπτικοποίηση
# Δημιουργία σημείων X_plot για ομαλή γραμμή
X_plot = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
```

```
# Μετασχηματισμός των σημείων X_plot με τους ίδιους πολυωνυμικούς όρους
X_plot_poly = poly_features.transform(X_plot)
y_poly_pred = model.predict(X_plot_poly)

plt.figure(figsize=(8, 6))
plt.scatter(df_ii["x"], df_ii["y"], label="Δεδομένα Dataset II",
color="darkblue")
plt.plot(
    X_plot,
    y_poly_pred,
    label="Πολυωνυμική Παλινδρόμηση (2ου Βαθμού)",
    color="red",
    linestyle="--",
)
plt.title("Anscombe's Quartet - Dataset II: Πολυωνυμική Παλινδρόμηση")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.grid(True, linestyle=":", alpha=0.6)
plt.show()
```

Κ. 5.5.5 – Προσαρμογή πολυωνυμικής συνάρτησης δεύτερου βαθμού στα δεδομένα Anscombe Dataset II.

Το αποτέλεσμα της εκτέλεσης του παραπάνω κώδικα είναι η Εικόνα 23.



Εικόνα 23 – Πολυωνυμική παλινδρόμηση των δεδομένων Anscombe Dataset II.

5.5.5. Ένα μεγαλύτερο παράδειγμα γραμμικής παλινδρόμησης

Στο σημείο αυτό θα παρουσιαστεί η εφαρμογή της γραμμικής παλινδρόμησης σε ένα μεγαλύτερο παράδειγμα. Θα χρησιμοποιηθούν τα δεδομένα του Boston House Pricing που μπορούν να μεταφορτωθούν από το https://lib.stat.cmu.edu/datasets/boston και θα επιχειρηθεί η εκτίμηση της αξίας σπιτιών με βάση χαρακτηριστικά τους που περιέχονται στο σύνολο δεδομένων. Ειδικότερα, τα βήματα που θα ακολουθηθούν θα είναι τα ακόλουθα:

1. Φόρτωση δεδομένων και μετασχηματισμός σε dataframe με 13 στήλες για τις ερμηνευτικές μεταβλητές και 1 στήλη για την εξαρτημένη μεταβλητή (τιμή σπιτιού).
2. Διαχωρισμός δεδομένων (506 δειγμάτων) σε 80% train και 20% test.
3. Δημιουργία και εκπαίδευση του μοντέλου γραμμικής παλινδρόμησης στο σύνολο train και εκτύπωση των συντελεστών παλινδρόμησης που προκύπτουν.
4. Εφαρμογή του μοντέλου στα δείγματα που έχουν διατηρηθεί στο test.
5. Αποτίμηση απόδοσης με τις μετρικές MSE (Mean Square Error) και R^2 (R-Squared).
6. Εκτίμηση τιμής για ένα "νέο" σπίτι. Συγγραφή συνάρτησης predict_price() που δέχεται ως ορίσματα το μοντέλο γραμμικής παλινδρόμησης και ένα **kwargs με όσες ερμηνευτικές μεταβλητές έχουν γνωστές τιμές.
7. Αποτίμηση αποτελεσμάτων, ιδέες βελτίωσης.

Το παράδειγμα παρουσιάζεται σταδιακά με τμήματα κώδικα που αντιστοιχούν στα κελιά ενός Jupyter σημειωματαρίου. Στο πρώτο τμήμα κώδικα, τον κώδικα Κ. 5.5.6, πραγματοποιείται το βήμα 1. Καθώς τα δεδομένα δεν είναι απευθείας σε μορφή που να μπορεί να χρησιμοποιηθεί, θα πρέπει να προηγηθεί μια διαδικασία τοποθέτησης των δεδομένων σε ένα dataframe.

```
# 1. Φόρτωση δεδομένων
import numpy as np
import pandas as pd

url = "https://lib.stat.cmu.edu/datasets/boston"
raw = pd.read_csv(url, sep="\s+", skiprows=22, header=None)
# Εντοπισμός ερμηνευτικών μεταβλητών και εξαρτημένης μεταβλητής
data = np.hstack(
    [raw.values[::2, :], raw.values[1::2, :2]]
)  # 13 ερμηνευτικές μεταβλητές
target = raw.values[1::2, 2]  # Διάμεση αξία σπιτιού

# Ονόματα ερμηνευτικών μεταβλητών
feature_names = [
    "CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE",
    "DIS", "RAD", "TAX", "PTRATIO", "B", "LSTAT",
]

# Συνδυασμός σε ένα DataFrame για ευκολία
df = pd.DataFrame(data, columns=feature_names)
df["MEDV"] = target
```

Κ. 5.5.6 – Βήμα 1: Φόρτωση και μετασχηματισμός δεδομένων.

Το dataframe που δημιουργείται έχει την μορφή της Εικόνα 24.

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

Εικόνα 24 – Οι πρώτες 5 εγγραφές του dataframe μετά το μετασχηματισμό των δεδομένων.

Στο βήμα 2 τα δεδομένα χωρίζονται σε δεδομένα εκπαίδευσης και σε δεδομένα ελέγχου, ώστε το μοντέλο να εκπαιδεύεται σε ένα υποσύνολο δεδομένων και να αξιολογείται σε ένα διαφορετικό υποσύνολο, εξασφαλίζοντας τη γενίκευσή του και την αποφυγή της υπερπροσαρμογής (overfitting). Ο κώδικας Κ. 5.5.7 χρησιμοποιεί τη συνάρτηση train_test_split() για να το επιτύχει αυτό.

```
# 2. Διαχωρισμός δεδομένων σε σύνολα εκπαίδευσης και ελέγχου
from sklearn.model_selection import train_test_split

X = df[feature_names].values
y = df["MEDV"].values
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

Κ. 5.5.7 – Βήμα 2: Διαχωρισμός σε δεδομένα εκπαίδευσης και δεδομένα ελέγχου.

Στο βήμα 3 δημιουργείται το μοντέλο γραμμικής παλινδρόμησης και γίνεται η εκπαίδευσή του χρησιμοποιώντας μόνο τα δεδομένα train. Ο κώδικας Κ. 5.5.8 πραγματοποιεί την προσαρμογή και εμφανίζει τους συντελεστές για τις ερμηνευτικές μεταβλητές.

```
# 3. Δημιουργία μοντέλου γραμμικής παλινδρόμησης, fit() στα δεδομένα train
model = LinearRegression()
model.fit(X_train, y_train)
# Εμφάνιση συντελεστών του μοντέλου
coeff_table = pd.DataFrame({
    "feature": feature_names,
    "Coefficient": model.coef_
})
print("\nΣυντελεστές μοντέλου:\n", coeff_table)
```

Κ. 5.5.8 – Δημιουργία μοντέλου παλινδρόμησης.

Οι συντελεστές του μοντέλου γραμμικής παλινδρόμησης που υπολογίζονται εμφανίζονται στη συνέχεια:

```
Συντελεστές μοντέλου:
     feature  Coefficient
0       CRIM    -0.113056
1         ZN     0.030110
2      INDUS     0.040381
3       CHAS     2.784438
4        NOX   -17.202633
5         RM     4.438835
6        AGE    -0.006296
```

```
7        DIS    -1.447865
8        RAD     0.262430
9        TAX    -0.010647
10   PTRATIO    -0.915456
11         B     0.012351
12     LSTAT    -0.508571
```

Στο βήμα 4 γίνονται προβλέψεις για τα δεδομένα του συνόλου test. Αυτό συμβαίνει καλώντας απλά τη μέθοδο predict(), όπως φαίνεται στον κώδικα Κ. 5.5.9.

```
# 4. Δημιουργία προβλέψεων για το σύνολο ελέγχου
y_pred = model.predict(X_test)
```

Κ. 5.5.9 – Λήψη προβλέψεων για όλα τα δεδομένα του συνόλου ελέγχου.

Στο βήμα 5 υπολογίζονται οι μετρικές απόδοσης MSE και R^2. Το MSE (Mean Squared Error) είναι το μέσο τετραγωνικό σφάλμα, ενώ συχνά χρησιμοποιείται και το RMSE που είναι η τετραγωνική ρίζα του MSE. Το R^2 που συχνά αναφέρεται ως R squared είναι ο λεγόμενος συντελεστής προσδιορισμού (coefficient of determination) και εκφράζει πόση από τη διακύμανση της εξαρτημένης μεταβλητής μπορεί να εξηγηθεί από τις ερμηνευτικές μεταβλητές. Τιμή του R^2 ίση με 1 σημαίνει τέλεια παρεμβολή, ενώ τιμή ίση με 0 σημαίνει ότι η παρεμβολή είναι το ίδιο καλή με το να έχει χρησιμοποιηθεί ο μέσος όρος των δεδομένων. Ο κώδικας Κ. 5.5.10 δείχνει ότι οι μετρικές MSE και R^2 μπορούν να υπολογιστούν με απλές κλήσεις συναρτήσεων του sklearn.

```
# 5. Αποτίμηση της απόδοσης του μοντέλου
from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", round(mse, 2))  # Mean Squared Error: 24.29
print("R^2 Score:", round(r2, 2))  # R^2 Score: 0.67
```

Κ. 5.5.10 – Μετρικές απόδοσης.

Θα πρέπει να σημειωθεί ότι υπάρχουν και άλλες μετρικές που χρησιμοποιούνται για την εκτίμηση ποιότητας μοντέλων γραμμικής παλινδρόμησης. Ορισμένες από αυτές είναι οι: F-statistic, BIC (Bayesian Information Criterion) και AIC (Akaike Information Criterion).

Στο βήμα 6 επιχειρείται η πρόβλεψη της τιμής μιας νέας εγγραφής δεδομένων που αφορά ένα «νέο» σπίτι για το οποίο είναι γνωστές οι τιμές των 13 ερμηνευτικών μεταβλητών που το περιγράφουν. Ο κώδικας Κ. 5.5.11 ορίζει τη συνάρτηση predict_price() που δέχεται ως ορίσματα το μοντέλο παλινδρόμησης και τις τιμές των ερμηνευτικών μεταβλητών για ένα νέο στιγμιότυπο δεδομένων.

```
# 6. Πρόβλεψη τιμών
def predict_price(model, **features):
    x = np.array([[features.get(name, 0) for name in feature_names]])
    price_thousands = model.predict(x)[0]
    return round(price_thousands * 1000, 2)
```

```
pred = predict_price(
    model,
    CRIM=0.05,
    ZN=18.0,
    INDUS=2.31,
    CHAS=0,
    NOX=0.54,
    RM=6.8,
    AGE=60.0,
    DIS=4.0,
    RAD=1,
    TAX=296,
    PTRATIO=15.3,
    B=390.5,
    LSTAT=5.0,
)

print(f"Πρόβλεψη τιμής: ${pred}")  # Πρόβλεψη τιμής: $30986.62
```

Κ. 5.5.11 – Πρόβλεψη τιμής νέου σπιτιού.

Στο βήμα 7 γίνεται μια προσπάθεια αποτίμησης της ποιότητας των αποτελεσμάτων και πως το μοντέλο θα μπορούσε να βελτιωθεί. Η τιμή του R^2 = 0.67, που έχει υπολογιστεί, δείχνει μια μάλλον μέτρια δυνατότητα πρόβλεψης για το μοντέλο γραμμικής παλινδρόμησης που αναπτύχθηκε. Μια τυπική λίστα ενεργειών που μπορεί να εφαρμοστεί προκειμένου να βελτιωθεί το μοντέλο είναι η ακόλουθη:

- Μελέτη δεδομένων.
- Αφαίρεση ή μετασχηματισμός ακραίων τιμών.
- Κλιμάκωση μεταβλητών (scaling).
- Εφαρμογή κανονικοποίησης (regularization) Lasso ή Ridge.
- Καλύτερη αξιολόγηση με Cross-Validation.
- Δοκιμή ισχυρότερων αλγορίθμων λαμβάνοντας υπόψη ότι τα μη γραμμικά μοντέλα μπορούν να περιγράψουν πολύπλοκες σχέσεις.

Μελέτη δεδομένων: Μελετώντας τα δεδομένα εντοπίζονται ορισμένα προβλήματα στα δεδομένα του Boston House Pricing dataset. Αρχικά, ένα σημαντικό πρόβλημα που σχετίζεται με την ηθική χρήση των αλγορίθμων μηχανικής μάθησης είναι ότι στα δεδομένα περιέχονται ως ερμηνευτικές μεταβλητές που σχετίζονται με την αξία των σπιτιών δεδομένα που έχουν να κάνουν με το ποσοστό των μαύρων κατοίκων στην περιοχή. Αυτό θέτει ένα σημαντικό θέμα διακρίσεων σε βάρος τμήματος πληθυσμού και είναι στη βάση τουπροβληματικό. Από την άλλη μεριά, η εξαρτώμενη μεταβλητή MEDV μετράται σε χιλιάδες δολλάρια ΗΠΑ αλλά τιμές πάνω από 50.000 φαίνεται να έχουν περιοριστεί τεχνητά στις 50.000. Το αποτέλεσμα είναι η μεταβλητή MEDV να είναι μια censored (λογοκριμένη) μεταβλητή και αυτό να οδηγεί σε υποεκτίμηση τιμών για τα υψηλής αξίας σπίτια. Ένας τρόπος αντιμετώπισης αυτής της κατάστασης είναι να εφαρμοστεί κάποιο άλλο μοντέλο που να είναι

ανθεκτικό σε λογοκριμένα δεδομένα όπως για παράδειγμα είναι το μοντέλο tobit regression που υπάρχει στο statsmodels. Ένας άλλος απλός τρόπος είναι να απορριφθούν όλα τα δείγματα με τιμή MEDV ίση με 50.000, αλλά αυτό μειώνει το μέγεθος των δεδομένων που μπορούν να χρησιμοποιηθούν για εκπαίδευση και για έλεγχο.

Αφαίρεση ή μετασχηματισμός ακραίων τιμών: Η παρουσία ακραίων τιμών (outliers) μπορεί να επηρεάσει δυσανάλογα την εκτίμηση των παραμέτρων, ιδιαίτερα σε μοντέλα όπως η γραμμική παλινδρόμηση που βασίζονται στην ελαχιστοποίηση του αθροίσματος τετραγώνων σφαλμάτων. Η διερεύνηση μέσω διαγραμμάτων διασποράς, boxplots ή μέτρων όπως το IQR και το z-score επιτρέπει τον εντοπισμό τέτοιων παρατηρήσεων. Ανάλογα με τη φύση τους, οι ακραίες τιμές μπορούν να αφαιρεθούν (εφόσον πρόκειται για σφάλματα καταγραφής), να μετασχηματιστούν (π.χ. λογαριθμικός μετασχηματισμός), ή να περιοριστούν μέσω άλλων τεχνικών. Η επιλογή στρατηγικής πρέπει να τεκμηριώνεται, καθώς η αυθαίρετη διαγραφή δεδομένων μπορεί να αλλοιώσει τη στατιστική δομή του δείγματος.

Κλιμάκωση μεταβλητών: Η κλιμάκωση των ερμηνευτικών μεταβλητών είναι κρίσιμη όταν τα χαρακτηριστικά μετρώνται σε διαφορετικές κλίμακες ή μονάδες, διότι μεταβλητές με μεγάλες αριθμητικές τιμές μπορεί να κυριαρχήσουν στη διαδικασία εκπαίδευσης. Τεχνικές όπως η τυποποίηση (standardization, μετασχηματισμός σε μέση τιμή 0 και διασπορά 1) ή η κανονικοποίηση σε συγκεκριμένο εύρος τιμών (π.χ. [0,1]) εξασφαλίζουν συγκρίσιμη συμβολή των χαρακτηριστικών.

Εφαρμογή κανονικοποίησης Lasso ή Ridge: Η κανονικοποίηση εισάγει έναν όρο ποινής στη συνάρτηση κόστους με στόχο τον περιορισμό του μεγέθους των συντελεστών και τη μείωση της υπερπροσαρμογής. Η Ridge (L2) ποινή συρρικνώνει ομαλά τους συντελεστές, ενώ η Lasso (L1) μπορεί να μηδενίσει ορισμένους από αυτούς, επιτελώντας ταυτόχρονα και επιλογή ερμηνευτικών μεταβλητών. Σε προβλήματα με μεγάλο αριθμό μεταβλητών, οι τεχνικές αυτές βελτιώνουν την απόδοση του μοντέλου.

Καλύτερη αξιολόγηση με Cross-Validation: Η αξιολόγηση με μία μόνο διάσπαση σε σύνολο εκπαίδευσης και ελέγχου μπορεί να οδηγήσει σε ασταθή εκτίμηση της απόδοσης. Η τεχνική της διασταυρούμενης επικύρωσης (k-fold cross-validation) επαναλαμβάνει την εκπαίδευση και αξιολόγηση σε πολλαπλές διαφορετικές διασπάσεις των δεδομένων, παρέχοντας μια πιο αξιόπιστη εκτίμηση της αναμενόμενης απόδοσης σε νέα δεδομένα.

Δοκιμή ισχυρότερων αλγορίθμων: Εάν ένα γραμμικό μοντέλο αποδεικνύεται ανεπαρκές για την αποτύπωση της σχέσης μεταξύ ερμηνευτικών μεταβλητών και εξαρτημένης μεταβλητής, μπορεί να

εξεταστεί η χρήση μη γραμμικών αλγορίθμων, όπως δέντρα αποφάσεων, τυχαία δάση (random forests) ή μέθοδοι ενίσχυσης (boosting). Τα μοντέλα αυτά έχουν τη δυνατότητα να συλλαμβάνουν πολύπλοκες αλληλεπιδράσεις και μη γραμμικότητες, με το κόστος αυξημένης υπολογιστικής πολυπλοκότητας και ενδεχομένως μειωμένης ερμηνευσιμότητας. Ενδεικτικά, στα δέντρα αποφάσεων περιλαμβάνονται αλγόριθμοι όπως οι CART και C4.5, στα τυχαία δάση η βασική ιδέα της τυχαιοποιημένης δειγματοληψίας και της συνάθροισης (bagging), ενώ στις μεθόδους ενίσχυσης περιλαμβάνονται αλγόριθμοι όπως AdaBoost, Gradient Boosting Machines (GBM), καθώς και σύγχρονες υλοποιήσεις όπως XGBoost, LightGBM και CatBoost. Επιπλέον, μπορούν να εξεταστούν και άλλες κατηγορίες μη γραμμικών μοντέλων, όπως οι μηχανές διανυσμάτων υποστήριξης με μη γραμμικούς πυρήνες (kernel SVM) και τα τεχνητά νευρωνικά δίκτυα (neural networks), τα οποία είναι ιδιαίτερα ισχυρά στην προσέγγιση σύνθετων συναρτήσεων.

5.6. Κατηγοριοποίηση

Η κατηγοριοποίηση ή αλλιώς ταξινόμηση αποτελεί θεμελιώδες πρόβλημα της επιβλεπόμενης μάθησης, στο οποίο στόχος είναι η ανάθεση μιας παρατήρησης σε μία από ένα σύνολο προκαθορισμένων κατηγοριών. Δεδομένου ενός συνόλου χαρακτηριστικών (features) που περιγράφουν κάθε δείγμα και ενός ιστορικού συνόλου δεδομένων με γνωστές ετικέτες (labels), κατασκευάζεται ένα μοντέλο το οποίο μαθαίνει τη σχέση μεταξύ χαρακτηριστικών και κατηγορίας. Το εκπαιδευμένο μοντέλο μπορεί στη συνέχεια να χρησιμοποιηθεί για την πρόβλεψη της κατηγορίας νέων, άγνωστων δεδομένων. Η ταξινόμηση εφαρμόζεται σε πληθώρα πεδίων, όπως η αναγνώριση προτύπων, η ανίχνευση ανεπιθύμητης αλληλογραφίας (spam detection), οι ιατρικές διαγνώσεις, οι πιστοληπτικές αξιολογήσεις κ.α.

Οι αλγόριθμοι που πραγματοποιούν κατηγοριοποίηση ονομάζονται κατηγοριοποιητές (classifiers) ή ταξινομητές και η βιβλιοθήκη scikit-learn παρέχει πληθώρα classifiers. Ενδεικτικά αναφέρεται εδώ ότι διατίθενται γραμμικά μοντέλα όπως η LogisticRegression, μέθοδοι βασισμένες σε αποστάσεις όπως ο KNeighborsClassifier, μοντέλα μεγιστοποίησης περιθωρίου όπως ο SVC (Support Vector Classifier), καθώς και δέντρα απόφασης (DecisionTreeClassifier) και μέθοδοι συνόλων όπως οι RandomForestClassifier, ExtraTreesClassifier και GradientBoostingClassifier. Όλοι αυτοί οι ταξινομητές ακολουθούν κοινή προγραμματιστική διεπαφή παρέχοντας τις μεθόδους fit(), predict(), και predict_proba() όπου υποστηρίζεται, επιτρέποντας την εύκολη σύγκριση διαφορετικών τεχνικών και την επιλογή του καταλληλότερου μοντέλου για το κάθε πρόβλημα.

5.6.1. Κατηγοριοποίηση Naïve Bayes

Ο αλγόριθμος Naive Bayes αποτελεί έναν από τους απλούστερους και ταχύτερους αλγορίθμους κατηγοριοποίησης. Η αποτελεσματικότητά του οφείλεται σε δύο βασικά στοιχεία: στην αξιοποίηση του θεωρήματος του Bayes και στην απλοποιητική υπόθεση ότι οι ερμηνευτικές μεταβλητές είναι υπό συνθήκη ανεξάρτητες μεταξύ τους, δεδομένης της ετικέτας της κλάσης. Παρότι η υπόθεση αυτή σπάνια ισχύει αυστηρά στην πράξη, ο αλγόριθμος συχνά επιτυγχάνει εξαιρετικά αποτελέσματα, ιδιαίτερα σε προβλήματα υψηλής διάστασης, όπως η ανάλυση κειμένου.

Η βασική ιδέα ξεκινά από την παρατήρηση ότι, από τα δεδομένα εκπαίδευσης, μπορούμε να εκτιμήσουμε για κάθε πιθανή ετικέτα L την πιθανότητα εμφάνισης συγκεκριμένων χαρακτηριστικών, δηλαδή την ποσότητα $P(\text{features} \mid L)$, καθώς και την εκ των προτέρων πιθανότητα της κλάσης $P(L)$. Ωστόσο, στο πρόβλημα πρόβλεψης αυτό που επιθυμούμε είναι η αντίστροφη πιθανότητα, δηλαδή η πιθανότητα η ετικέτα να είναι L, δεδομένων των χαρακτηριστικών ενός νέου δείγματος:

$$P(L \mid \text{features})$$

Το θεώρημα του Bayes παρέχει ακριβώς αυτόν τον μετασχηματισμό:

$$P(L \mid \text{features}) = \frac{P(\text{features} \mid L)\, P(L)}{P(\text{features})}$$

Στην πράξη, για σκοπούς κατηγοριοποίησης, ο παρονομαστής $P(\text{features})$ είναι κοινός για όλες τις κατηγορίες και συνεπώς δεν επηρεάζει τη σύγκριση. Έτσι, για δύο κατηγορίες L_1 και L_2, η απόφαση βασίζεται στη σύγκριση των ποσοτήτων:

$$P(\text{ features} \mid L_1)P(L_1) \text{ και } P(\text{features} \mid L_2)P(L_2)$$

Το νέο δείγμα ταξινομείται στην κατηγορία που μεγιστοποιεί την παραπάνω ποσότητα (κανόνας μέγιστης εκ των υστέρων πιθανότητας – Maximum A Posteriori, MAP).

Η «naive» υπόθεση ανεξαρτησίας απλοποιεί σημαντικά τον υπολογισμό της πιθανότητας $P(\text{features} \mid L)$, καθώς επιτρέπει να εκφραστεί ως γινόμενο επιμέρους πιθανοτήτων κάθε χαρακτηριστικού:

$$P(x_1, x_2, \ldots, x_p \mid L) = \prod_{j=1}^{p} P(\, x_j \mid L).$$

Αυτό μειώνει δραστικά την υπολογιστική πολυπλοκότητα και καθιστά τον αλγόριθμο ιδιαίτερα αποδοτικό ακόμη και σε μεγάλα σύνολα δεδομένων.

Η βιβλιοθήκη scikit-learn παρέχει διαφορετικές παραλλαγές του Naïve Bayes (Gaussian, Multinomial, Bernoulli) ανάλογα με τη φύση των δεδομένων. Παρά την απλότητά του, ο Naive Bayes αποτελεί ισχυρό εργαλείο κατηγοριοποίησης και συχνά λειτουργεί ως αποτελεσματικό baseline μοντέλο σε προβλήματα μηχανικής μάθησης.

5.6.1.1 Παράδειγμα εφαρμογής της κατηγοριοποίησης Naive Bayes σε συνθετικά δεδομένα

Στο παράδειγμα που παρουσιάζεται στην παρούσα παράγραφο δημιουργούνται συνθετικά δεδομένα προκειμένου στη συνέχεια να κατηγοριοποιηθούν. Ο κώδικας του παραδείγματος αντιστοιχεί σε κελιά ενός Jupyter σημειωματαρίου τα οποία θα παρουσιαστούν στη συνέχεια για τα ακόλουθα βήματα:

1. Δημιουργία 1200 τυχαίων δειγμάτων με δύο χαρακτηριστικά (features) για κάθε δείγμα. Κάθε δείγμα θα έχει μια ετικέτα που θα προσδιορίζει την κατηγορία στην οποία ανήκει. Οι πιθανές κατηγορίες θα είναι τρεις (0, 1 και 2).
2. Διαχωρισμός των δεδομένων σε δεδομένα εκπαίδευσης (70%) και σε δεδομένα ελέγχου (30%).
3. Δημιουργία ενός κατηγοριοποιητή GaussianNB και προσαρμογή του στα δεδομένα εκπαίδευσης.
4. Υπολοισμός της ορθότητας εντοπισμού της κατηγορίας στην οποία ανήκει καθένα από τα δείγματα του συνόλου ελέγχου.

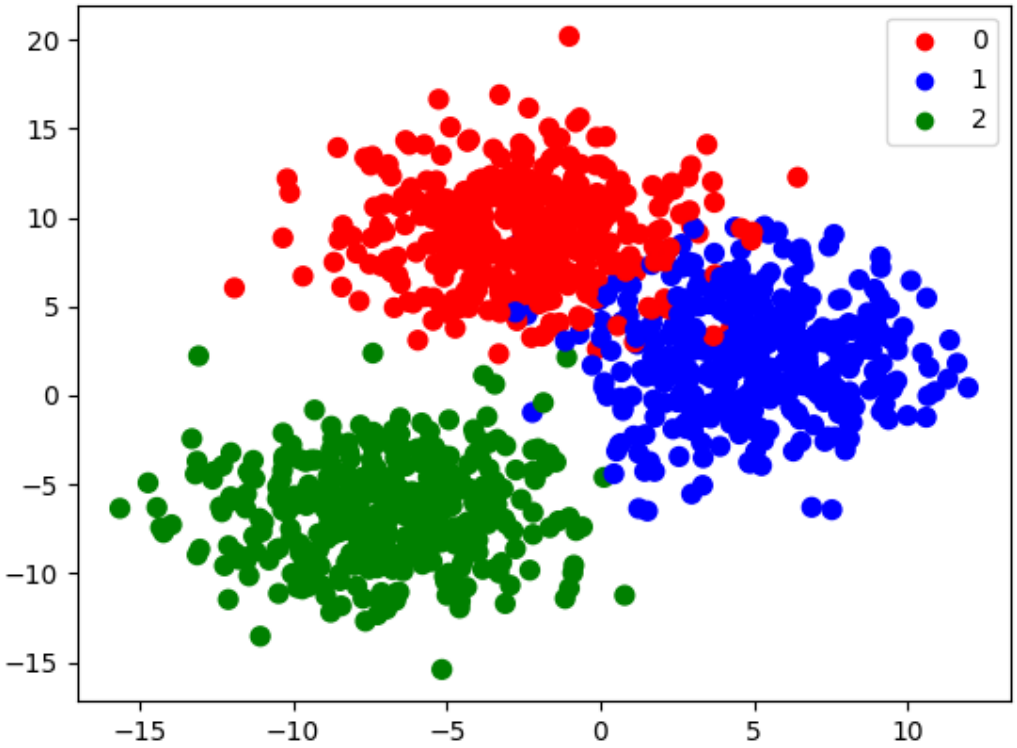
Το βήμα 1 υλοποιείται με τον κώδικα Κ. 5.6.1. Σε αυτό το βήμα δημιουργείται ένα συνθετικό σύνολο δεδομένων με τη συνάρτηση make_blobs, το οποίο περιλαμβάνει 1200 δείγματα, δύο αριθμητικά χαρακτηριστικά ανά δείγμα και τρεις διακριτές κατηγορίες. Η παράμετρος cluster_std καθορίζει τη διασπορά των συστάδων, ενώ η παράμετρος random_state εξασφαλίζει αναπαραγωγιμότητα. Στη συνέχεια, τα δεδομένα απεικονίζονται σε δισδιάστατο διάγραμμα διασποράς, όπου κάθε κατηγορία εμφανίζεται με διαφορετικό χρώμα.

```
# 1. Δημιουργία συνθετικών δεδομένων: 2 χαρακτηριστικά, 3 ετικέτες
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

X, y = make_blobs(
    n_samples=1200, n_features=2, centers=3, random_state=42,
cluster_std=2.9
)
colors = np.array(["red", "blue", "green"])
```

```
plt.scatter(X[:, 0], X[:, 1], c=colors[y], s=50)
for label, c in enumerate(colors):
    plt.scatter([], [], c=c, label=str(label))
plt.legend()
plt.show()
```

Κ. 5.6.1 – Δημιουργία με make_blobs() και οπτικοποίηση συνθετικών δεδομένων.

Η εκτέλεση του παραπάνω κώδικα θα εμφανίσει το γράφημα της Εικόνα 25.



Εικόνα 25 – Οπτικοποίηση συνθετικών δεδομένων και των κατηγοριών τους.

Το βήμα 2 υλοποιείται με τον κώδικα Κ. 5.6.2. Στο βήμα αυτό τα δεδομένα χωρίζονται σε σύνολο εκπαίδευσης και σύνολο ελέγχου με χρήση της συνάρτησης train_test_split. Το 70% των δεδομένων χρησιμοποιείται για την εκπαίδευση του μοντέλου και το υπόλοιπο 30% για την αξιολόγησή του. Η παράμετρος random_state διασφαλίζει ότι ο διαχωρισμός θα είναι ο ίδιος σε κάθε εκτέλεση.

```
# 2. Διαχωρισμός δεδομένων: 70% training, 30% testing
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

Κ. 5.6.2 – Διαχωρισμός δεδομένων σε σύνολο εκπαίδευσης και ελέγχου (70%–30%).

Το βήμα 3 υλοποιείται με τον κώδικα Κ. 5.6.3. Στο βήμα αυτό δημιουργείται ένα αντικείμενο κατηγοριοποιητή τύπου GaussianNB, το οποίο υλοποιεί τον αλγόριθμο Naïve Bayes. Με την κλήση της μεθόδου fit(), το μοντέλο υπολογίζει τις παραμέτρους του μοντέλου από τα δεδομένα εκπαίδευσης.

```
# 3. Δημιουργία ενός κατηγοριοποιητή Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
```

```
model = GaussianNB()
model.fit(X_train, y_train)  # εκπαίδευση στα δεδομένα train
```
Κ. 5.6.3 – Εκπαίδευση κατηγοριοποιητή Gaussian Naive Bayes.

Το βήμα 4 υλοποιείται με τον κώδικα Κ. 5.6.4. Στο βήμα αυτό το εκπαιδευμένο μοντέλο χρησιμοποιείται για την πρόβλεψη των κατηγοριών των δειγμάτων του συνόλου ελέγχου μέσω της μεθόδου predict(). Στη συνέχεια, η συνάρτηση accuracy_score() υπολογίζει το ποσοστό των σωστών προβλέψεων επί του συνόλου των δειγμάτων test, παρέχοντας ένα βασικό μέτρο αξιολόγησης της απόδοσης του κατηγοριοποιητή.

```
# 4. Υπολογισμός ορθότητας (accuracy) προβλέψεων στα δεδομένα test
from sklearn.metrics import accuracy_score

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}") # Accuracy: 0.9722
```
Κ. 5.6.4 – Αξιολόγηση απόδοσης με υπολογισμό accuracy στο σύνολο ελέγχου.

5.6.2. Μετρικές για προβλήματα κατηγοριοποίησης

Στα προβλήματα κατηγοριοποίησης, η αξιολόγηση ενός μοντέλου δεν περιορίζεται απλώς στο ποσοστό σωστών προβλέψεων, αλλά απαιτεί τη χρήση κατάλληλων μετρικών που αποτυπώνουν με μεγαλύτερη ακρίβεια τη συμπεριφορά του μοντέλου. Η απλούστερη μετρική είναι η accuracy (ορθότητα), δηλαδή το ποσοστό των σωστά ταξινομημένων δειγμάτων επί του συνόλου. Ωστόσο, η ορθότητα μπορεί να είναι παραπλανητική, όπως σε περιπτώσεις μη-ισορροπημένων δεδομένων (imbalanced datasets), όπου μία κατηγορία υπερισχύει σημαντικά της άλλης.

Βασικό ρόλο για την εκτίμηση των αποτελεσμάτων ενός μοντέλου έχει ο πίνακας σύγχυσης (confusion matrix), ο οποίος καταγράφει αναλυτικά τις προβλέψεις του μοντέλου. Ειδικά για την περίπτωση δυαδικής κατηγοριοποίησης (κατηγοριοποίησης με 2 κατηγορίες-κλάσεις) διακρίνονται τέσσερις βασικές ποσότητες:

- **True Positives (TP)**: θετικά δείγματα που ταξινομήθηκαν σωστά ως θετικά,
- **True Negatives (TN)**: αρνητικά δείγματα που ταξινομήθηκαν σωστά ως αρνητικά,
- **False Positives (FP)**: αρνητικά δείγματα που ταξινομήθηκαν εσφαλμένα ως θετικά (σφάλμα τύπου I),
- **False Negatives (FN)**: θετικά δείγματα που ταξινομήθηκαν εσφαλμένα ως αρνητικά (σφάλμα τύπου II).

Για παράδειγμα, σε ένα σύστημα ανίχνευσης ανεπιθύμητης αλληλογραφίας (spam detection), ένα κανονικό μήνυμα που επισημαίνεται ως spam αποτελεί False Positive, ενώ ένα spam μήνυμα που

χαρακτηρίζεται ως κανονικό αποτελεί False Negative. Η διάκριση αυτή είναι κρίσιμη, διότι τα δύο είδη σφαλμάτων ενδέχεται να έχουν διαφορετική σημασία ανάλογα με την περίπτωση προβλήματος.

Με βάση τις ποσότητες TP, TN, FP και FN ορίζονται σημαντικές μετρικές όπως:

- **Precision:** $\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$, που μετρά πόσο αξιόπιστες είναι οι θετικές προβλέψεις.
- **Recall (ή Sensitivity):** $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$, που μετρά την ικανότητα του μοντέλου να εντοπίζει τα πραγματικά θετικά δείγματα.
- **F-score**: $F_\beta = (1+\beta^2)\cdot \frac{Precision \cdot Recall}{\beta^2 \cdot Precision + Recall}$ όπου το β καθορίζει τη σχετική βαρύτητα μεταξύ Precision και Recall, με $\beta > 1$ να δίνει μεγαλύτερη έμφαση στο Recall ενώ με $\beta < 1$ να δίνει μεγαλύτερη έμφαση στο Precision.
- **F1-score**: ειδική περίπτωση του F-score με ίση βαρύτητα σε precision και recall (αρμονικός μέσος precision και recall, $\beta = 1$).

Η επιλογή της κατάλληλης μετρικής εξαρτάται από το πρόβλημα. Σε εφαρμογές όπου ένα False Positive έχει υψηλό κόστος (π.χ. απόρριψη έγκυρου email), το precision είναι ιδιαίτερα σημαντικό. Αντίθετα, για παράδειγμα σε ιατρικές διαγνώσεις, όπου η αποτυχία εντοπισμού μιας νόσου είναι κρίσιμη, προτεραιότητα δίνεται στο recall.

Επιπλέον, όταν το μοντέλο επιστρέφει πιθανότητες και όχι μόνο τελικές ετικέτες, χρησιμοποιούνται καμπύλες **ROC (Receiver Operating Characteristic)** και η μετρική **AUC (Area Under the Curve)**. Η ROC καμπύλη απεικονίζει τη σχέση μεταξύ του ποσοστού αληθών θετικών (True Positive Rate) και του ποσοστού ψευδών θετικών (False Positive Rate) για διαφορετικά κατώφλια απόφασης. Όσο μεγαλύτερη είναι η επιφάνεια κάτω από την καμπύλη (AUC), τόσο καλύτερη θεωρείται η διακριτική ικανότητα του μοντέλου. Τιμή AUC ίση με 1 υποδηλώνει τέλειο κατηγοριοποιητή, ενώ τιμή 0.5 αντιστοιχεί σε τυχαία επιλογή.

Τέλος, σε πιθανοτικά μοντέλα χρησιμοποιούνται και μετρικές όπως η **cross-entropy (διασταυρούμενη εντροπία)**, η οποία αξιολογεί την απόκλιση μεταξύ των πραγματικών ετικετών και των προβλεπόμενων πιθανοτήτων. Η μετρική αυτή είναι ιδιαίτερα σημαντική σε μοντέλα όπως η λογιστική παλινδρόμηση και τα νευρωνικά δίκτυα, όπου η ποιότητα της πιθανότητας είναι εξίσου σημαντική με τη σωστή τελική ταξινόμηση.

5.6.3. Κατηγοριοποίηση του Iris dataset με το GaussianNB

Ως ένα ακόμα παράδειγμα εφαρμογής του κατηγοριοποιητή Naive Bayes θα παρουσιαστεί η εφαρμογή του στο Iris dataset που παρουσιάστηκε στην ενότητα 5.3.1.2. Στο παράδειγμα αυτό θα ακολουθηθούν τα εξής βήματα:

1. Φόρτωση του Iris dataset από το scikit-learn και επισκόπηση των δειγμάτων που περιέχει.
2. Διαχωρισμός των δειγμάτων σε training set και testing set χρησιμοποιώντας τη μέθοδο train_test_split() έτσι ώστε το training set να είναι 80% των συνολικών δεδομένων με παράλληλη χρήση του random_state=0 για επαναληψιμότητα.
3. Εκπαίδευση του GaussianNB στο training set και πρόβλεψη ετικετών για το testing set.
4. Υπολογισμός και εκτύπωση διαφόρων μετρικών του μοντέλου:
    - confusion matrix και σχεδίασή του με το seaborn
    - accuracy
    - precision
    - recall
    - F1-score

Το παράδειγμα παρουσιάζεται σταδιακά με τμήματα κώδικα που αντιστοιχούν στα κελιά ενός Jupyter σημειωματαρίου. Στο πρώτο βήμα, στον κώδικα Κ. 5.6.5, πραγματοποιείται η φόρτωση του dataset Iris από τη βιβλιοθήκη scikit-learn. Η συνάρτηση load_iris(as_frame=True) επιστρέφει τα δεδομένα σε μορφή pandas DataFrame, γεγονός που διευκολύνει την επισκόπηση και τον χειρισμό τους. Το αντικείμενο iris περιλαμβάνει τόσο τα χαρακτηριστικά (μήκος και πλάτος σέπαλου και πέταλου) όσο και την κατηγορία κάθε δείγματος. Με την εντολή iris.frame δημιουργείται ένα ενιαίο DataFrame που περιέχει όλες τις μεταβλητές, ενώ η head() επιτρέπει την προβολή των πρώτων γραμμών, ώστε να γίνει κατανοητή η δομή των δεδομένων και η μορφή των κατηγοριών-στόχων.

```
# 1. Φόρτωση του dataset
from sklearn.datasets import load_iris

iris = load_iris(as_frame=True)
df = iris.frame
print(df.head())
```

Κ. 5.6.5 – Φόρτωση του Iris dataset.

Το dataframe που φορτώνεται έχει την ακόλουθη μορφή:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Στο βήμα 2, στον κώδικα Κ. 5.6.6, πραγματοποιείται ο διαχωρισμός των δεδομένων σε σύνολο εκπαίδευσης (training set) και σύνολο ελέγχου (testing set). Τα χαρακτηριστικά αποθηκεύονται στη

μεταβλητή X, ενώ οι ετικέτες-στόχοι στη μεταβλητή y. Η συνάρτηση train_test_split() χρησιμοποιείται για τον τυχαίο διαχωρισμό των δεδομένων, ορίζοντας test_size=0.2, ώστε το 20% των δειγμάτων να χρησιμοποιηθεί για αξιολόγηση και το υπόλοιπο 80% για εκπαίδευση. Η παράμετρος random_state=0 διασφαλίζει ότι ο διαχωρισμός θα είναι αναπαραγώγιμος, επιτρέποντας την επανάληψη των ίδιων αποτελεσμάτων σε μελλοντικές εκτελέσεις του κώδικα.

```
# 2. Διαχωρισμός σε training και testing (80%-20%)
from sklearn.model_selection import train_test_split

X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)
```

Κ. 5.6.6 – Διαχωρισμός Iris dataset σε training και testing σύνολα.

Στο βήμα 3, στον κώδικα Κ. 5.6.7, εκπαιδεύεται ο κατηγοριοποιητής Gaussian Naive Bayes στο σύνολο εκπαίδευσης. Δημιουργείται ένα αντικείμενο GaussianNB, το οποίο υλοποιεί την εκδοχή του Naive Bayes που υποθέτει ότι τα χαρακτηριστικά ακολουθούν κανονική κατανομή μέσα σε κάθε κατηγορία. Με την κλήση της μεθόδου fit(X_train, y_train) το μοντέλο υπολογίζει τις παραμέτρους των κατανομών (μέσους και διακυμάνσεις) για κάθε χαρακτηριστικό και για κάθε κλάση. Στη συνέχεια, με τη μέθοδο predict(X_test) παράγονται οι προβλέψεις για τα δείγματα του testing set, δηλαδή οι εκτιμώμενες κατηγορίες των άγνωστων δειγμάτων.

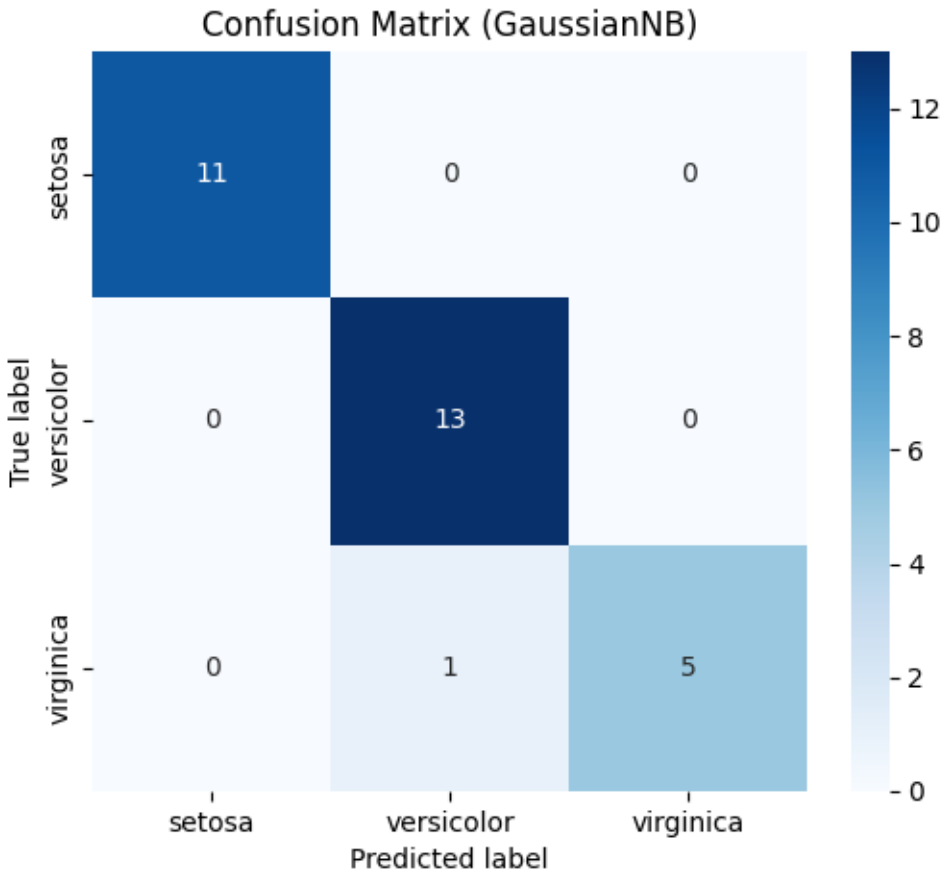
```
# 3α. Εκπαίδευση του Gaussian Naive Bayes στο training set
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()
model.fit(X_train, y_train)

# 3β.Προβλέψεις στο testing set
y_pred = model.predict(X_test)
```

Κ. 5.6.7 – Εκπαίδευση του Naive Bayes και λήψη προβλέψεων.

Στο βήμα 4, στον κώδικα Κ. 5.6.8, αξιολογείται η απόδοση του μοντέλου με τη χρήση διαφόρων μετρικών κατηγοριοποίησης. Αρχικά υπολογίζεται ο πίνακας σύγχυσης, ο οποίος αποτυπώνει αναλυτικά τις σωστές και λανθασμένες προβλέψεις για κάθε κατηγορία. Η οπτικοποίησή του με heatmap μέσω της βιβλιοθήκης seaborn, όπως φαίνεται στην Εικόνα 26, επιτρέπει την εύκολη κατανόηση της κατανομής των σφαλμάτων. Στη συνέχεια υπολογίζονται οι μετρικές accuracy, precision, recall και F1-score με χρήση της επιλογής average="macro", η οποία υπολογίζει τον μέσο όρο των μετρικών για κάθε κατηγορία χωρίς στάθμιση ως προς το πλήθος των δειγμάτων. Τα αποτελέσματα δείχνουν υψηλή ορθότητα (accuracy = 0.97), γεγονός που υποδηλώνει ότι το μοντέλο ταξινομεί σωστά τη μεγάλη πλειονότητα των δειγμάτων του Iris dataset. Οι υψηλές τιμές precision και recall επιβεβαιώνουν ότι ο Gaussian Naive Bayes αποδίδει ικανοποιητικά και στις τρεις κατηγορίες του προβλήματος.

```python
# 4. Υπολογισμός και εκτύπωση μετρικών
from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
)

cm = confusion_matrix(y_test, y_pred)

# Σχεδίαση Heatmap με το seaborn
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(6, 5))
sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=iris.target_names,
    yticklabels=iris.target_names,
)
plt.xlabel("Predicted label")
plt.ylabel("True label")
plt.title("Confusion Matrix (GaussianNB)")
plt.show()

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, average="macro")
rec = recall_score(y_test, y_pred, average="macro")
f1 = f1_score(y_test, y_pred, average="macro")
print(f"\nAccuracy:  {acc:.2f}")  # Accuracy:  0.97
print(f"Precision: {prec:.2f}")  # Precision: 0.98
print(f"Recall:    {rec:.2f}")  # Recall:    0.94
print(f"F1-score:  {f1:.2f}")  # F1-score:  0.96
```

Κ. 5.6.8 – Μετρικές απόδοσης του μοντέλου.



Εικόνα 26 – Πίνακας σύγχυσης σχεδιασμένος με τη βιβλιοθήκη seaborn.

5.6.4. Άλλοι αλγόριθμοι κατηγοριοποίησης

Όπως ήδη αναφέρθηκε υπάρχουν πολλοί αλγόριθμοι κατηγοριοποίησης. Στη συνέχεια θα παρουσιαστούν εν συντομία τέσσερις τέτοιοι αλγόριθμοι: τα δένδρα αποφάσεων, η λογιστική παλινδρόμηση, ο αλγόριθμος των k-πλησιέστερων γειτόνων και οι μηχανές υποστήριξης διανυσμάτων.

5.6.4.1 Δέντρα αποφάσεων

Τα δέντρα αποφάσεων (*Decision Trees*) αποτελούν έναν ιδιαίτερα δημοφιλή αλγόριθμο επιβλεπόμενης μάθησης, ο οποίος χρησιμοποιείται τόσο για προβλήματα κατηγοριοποίησης όσο και για προβλήματα παλινδρόμησης. Η βασική του ιδέα είναι η διαδοχική διάσπαση του χώρου των δεδομένων σε υποσύνολα, με βάση τιμές χαρακτηριστικών, ώστε να παραχθεί μια ιεραρχική δομή λήψης αποφάσεων. Κάθε απόφαση λαμβάνεται μέσω ενός ερωτήματος της μορφής «αν–τότε», το οποίο οδηγεί σε διαφορετικό κλάδο του δέντρου. Η δομή του μοντέλου είναι δενδρική με τους εσωτερικούς κόμβους να αντιστοιχούν σε ερωτήσεις ή διαχωρισμούς πάνω σε κάποιο χαρακτηριστικό (π.χ. «ηλικία < 30;»), τα κλαδιά αντιπροσωπεύουν τα πιθανά αποτελέσματα αυτών

των ερωτήσεων, και τα φύλλα (leaves) αντιστοιχούν στις τελικές προβλέψεις, δηλαδή στις κατηγορίες ή στις αριθμητικές τιμές που αποδίδει το μοντέλο. Η κατασκευή του δέντρου γίνεται με επαναληπτικό τρόπο, επιλέγοντας κάθε φορά τον διαχωρισμό που οδηγεί στη μεγαλύτερη «καθαρότητα» των παραγόμενων κόμβων. Η έννοια της καθαρότητας σχετίζεται με το πόσο ομοιογενή είναι τα δείγματα σε έναν κόμβο. Για προβλήματα κατηγοριοποίησης χρησιμοποιούνται συνήθως μέτρα όπως ο δείκτης Gini ή η εντροπία (entropy). Ο στόχος του αλγορίθμου είναι να επιλέγει διαχωρισμούς που ελαχιστοποιούν την μη-καθαρότητα και οδηγούν σε κόμβους όπου κυριαρχεί μία κατηγορία.

Ένα από τα βασικά πλεονεκτήματα των δέντρων απόφασης είναι η ερμηνευσιμότητά τους. Το μοντέλο μπορεί να απεικονιστεί γραφικά και οι αποφάσεις του να αναλυθούν εύκολα, γεγονός που το καθιστά ιδιαίτερα χρήσιμο σε εφαρμογές όπου απαιτείται διαφάνεια. Επιπλέον, μπορεί να χειριστεί τόσο αριθμητικά όσο και κατηγορικά δεδομένα χωρίς ιδιαίτερη προεπεξεργασία. Ωστόσο, τα δέντρα απόφασης εμφανίζουν έντονη τάση για υπερπροσαρμογή (overfitting), ειδικά όταν επιτρέπεται να αναπτυχθούν χωρίς περιορισμούς (π.χ. χωρίς όριο στο βάθος του δέντρου). Για τον λόγο αυτό εφαρμόζονται τεχνικές περιορισμού της πολυπλοκότητας, όπως το κλάδεμα (pruning) ή ο καθορισμός παραμέτρων όπως το max_depth και το min_samples_split.

Η βιβλιοθήκη scikit-learn προσφέρει ολοκληρωμένη υποστήριξη για μοντέλα βασισμένα σε δέντρα απόφασης, καλύπτοντας τόσο βασικές όσο και προχωρημένες τεχνικές. Το απλό δέντρο απόφασης υλοποιείται μέσω της κλάσης DecisionTreeClassifier του υποπακέτου sklearn.tree, η οποία επιτρέπει την κατασκευή ιεραρχικών μοντέλων διαχωρισμού με κριτήρια όπως ο δείκτης Gini ή η εντροπία. Παρέχονται παράμετροι ελέγχου της πολυπλοκότητας, όπως max_depth, min_samples_split και min_samples_leaf. Πέρα από το μεμονωμένο δέντρο, η βιβλιοθήκη υποστηρίζει και μεθόδους συνόλων (ensemble methods) που βασίζονται σε πολλαπλά δέντρα. Η κλάση RandomForestClassifier του υποπακέτου sklearn.ensemble υλοποιεί τη μέθοδο Random Forest, συνδυάζοντας πολλά δέντρα μέσω τεχνικής bagging και τυχαίας επιλογής χαρακτηριστικών, με στόχο τη βελτίωση της γενίκευσης. Η ExtraTreesClassifier (Extremely Randomized Trees) εισάγει ακόμη μεγαλύτερη τυχαιότητα στον τρόπο επιλογής των διαχωρισμών, επιτυγχάνοντας συχνά ταχύτερη εκπαίδευση και ανταγωνιστική απόδοση. Επιπλέον, διατίθενται τεχνικές boosting, όπως οι GradientBoostingClassifier και HistGradientBoostingClassifier, όπου τα δέντρα κατασκευάζονται διαδοχικά και κάθε νέο δέντρο διορθώνει τα σφάλματα των προηγούμενων. Όλα τα παραπάνω μοντέλα ακολουθούν την ενιαία διεπαφή της βιβλιοθήκης scikit-learn, με τις μεθόδους fit(), predict() και predict_proba(), γεγονός που διευκολύνει τη σύγκριση διαφορετικών προσεγγίσεων και την ενσωμάτωσή τους σε ροές εργασίας (pipelines).

5.6.4.2 Λογιστική παλινδρόμηση

Η λογιστική παλινδρόμηση (logistic regression) αποτελεί μία από τις βασικότερες μεθόδους επιβλεπόμενης μάθησης για προβλήματα κατηγοριοποίησης, ιδιαίτερα όταν η μεταβλητή-στόχος είναι δυαδική (0/1). Σε αντίθεση με τη γραμμική παλινδρόμηση, η οποία προβλέπει συνεχείς αριθμητικές τιμές, η λογιστική παλινδρόμηση στοχεύει στην εκτίμηση της πιθανότητας ένα παρατηρούμενο δείγμα να ανήκει σε μια συγκεκριμένη κατηγορία.

Η μέθοδος βασίζεται στη λογιστική συνάρτηση (logistic function ή sigmoid), η οποία μετασχηματίζει έναν γραμμικό συνδυασμό των ερμηνευτικών μεταβλητών σε τιμή στο διάστημα (0,1), επιτρέποντας έτσι την ερμηνεία της ως πιθανότητα. Η συνάρτηση έχει τη μορφή:

$$P(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n)}}$$

όπου ο όρος $\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$ είναι ένας γραμμικός συνδυασμός των χαρακτηριστικών.

Το μοντέλο αποδίδει ως αποτέλεσμα πιθανότητες και όχι απλώς ετικέτες κατηγοριών. Για να ληφθεί τελική απόφαση ταξινόμησης, χρησιμοποιείται συνήθως ένα κατώφλι (threshold), συχνά ίσο με 0.5. Για παράδειγμα, αν η εκτιμώμενη πιθανότητα είναι μεγαλύτερη από 0.5, τότε το δείγμα ταξινομείται στην κατηγορία 1, διαφορετικά στην κατηγορία 0. Αυτή η ιδιότητα καθιστά τη λογιστική παλινδρόμηση ιδιαίτερα χρήσιμη σε εφαρμογές όπως η πρόβλεψη αποδοχής/απόρριψης, επιτυχίας/αποτυχίας ή απαντήσεων ναι/όχι.

Η λογιστική παλινδρόμηση υλοποιείται εύκολα μέσω της βιβλιοθήκης scikit-learn και ειδικότερα μέσω της κλάσης LogisticRegression του υποπακέτου sklearn.linear_model. Η διεπαφή της είναι παρόμοια με τα υπόλοιπα μοντέλα της βιβλιοθήκης, επιτρέποντας εκπαίδευση με τη μέθοδο fit(), πρόβλεψη κατηγορίας με predict() και πρόβλεψη πιθανοτήτων με predict_proba().

5.6.4.3 Κ-πλησιέστεροι γείτονες

Ο αλγόριθμος των Κ-πλησιέστερων γειτόνων (k-Nearest Neighbors – KNN) αποτελεί μία από τις πιο απλές και διαισθητικές μεθόδους επιβλεπόμενης μάθησης για προβλήματα κατηγοριοποίησης και παλινδρόμησης. Η βασική του ιδέα είναι ότι ένα νέο σημείο δεδομένων μπορεί να κατηγοριοποιηθεί εξετάζοντας τα «γειτονικά» του σημεία στον χώρο των χαρακτηριστικών. Συγκεκριμένα, για μια νέα παρατήρηση, ο αλγόριθμος υπολογίζει τις αποστάσεις της από όλα τα διαθέσιμα σημεία του συνόλου εκπαίδευσης και εντοπίζει τα k πλησιέστερα. Η τελική κατηγορία αποδίδεται συνήθως με πλειοψηφική ψήφο μεταξύ των γειτόνων αυτών.

Η μέθοδος δεν προϋποθέτει ρητή εκπαίδευση με την έννοια της εκτίμησης παραμέτρων. Για τον λόγο αυτό συχνά χαρακτηρίζεται ως «lazy learning» αλγόριθμος, καθώς η κύρια υπολογιστική επιβάρυνση μεταφέρεται στο στάδιο της πρόβλεψης. Αυτό καθιστά τον KNN ιδιαίτερα απλό στην υλοποίηση και αποτελεσματικό σε μικρά ή μεσαίου μεγέθους σύνολα δεδομένων. Ωστόσο, σε μεγάλα σύνολα δεδομένων η ανάγκη υπολογισμού αποστάσεων από όλα τα σημεία καθιστά τον αλγόριθμο υπολογιστικά απαιτητικό.

Ένα ακόμη κρίσιμο ζήτημα είναι η κλίμακα των χαρακτηριστικών. Επειδή ο KNN βασίζεται σε μετρικές αποστάσεων (όπως η Ευκλείδεια απόσταση), μεταβλητές με μεγαλύτερο εύρος τιμών μπορεί να επηρεάσουν δυσανάλογα το αποτέλεσμα. Για τον λόγο αυτό, πριν από την εφαρμογή του αλγορίθμου, είναι συνήθως απαραίτητη η κανονικοποίηση των δεδομένων.

Ο KNN υλοποιείται μέσω της βιβλιοθήκης scikit-learn και ειδικότερα της κλάσης KNeighborsClassifier του υποπακέτου sklearn.neighbors. Η επιλογή της παραμέτρου k, καθώς και της μετρικής απόστασης, επηρεάζει σημαντικά την απόδοση του μοντέλου και αποτελεί αντικείμενο πειραματισμού και αξιολόγησης στο πλαίσιο της ανάλυσης δεδομένων.

5.6.4.4 Support Vector Machines

Οι Μηχανές Διανυσμάτων Υποστήριξης (Support Vector Machines - SVM) αποτελούν μια ισχυρή μέθοδο επιβλεπόμενης μάθησης για προβλήματα κατηγοριοποίησης και παλινδρόμησης. Η βασική τους ιδέα είναι ο προσδιορισμός ενός υπερεπιπέδου (hyperplane) το οποίο διαχωρίζει τις κατηγορίες με τον καλύτερο δυνατό τρόπο. Σε αντίθεση με απλούστερες μεθόδους που επιδιώκουν απλώς έναν διαχωρισμό, οι SVM επιλέγουν το υπερεπίπεδο που μεγιστοποιεί το περιθώριο (margin), δηλαδή την ελάχιστη απόσταση μεταξύ του διαχωριστικού επιπέδου και των πλησιέστερων σημείων κάθε κατηγορίας. Τα σημεία αυτά ονομάζονται διανύσματα υποστήριξης (support vectors) και αυτά καθορίζουν τη θέση του διαχωριστή.

Σε γραμμικά διαχωρίσιμα δεδομένα, το υπερεπίπεδο είναι γραμμικό. Ωστόσο, όταν τα δεδομένα δεν μπορούν να διαχωριστούν γραμμικά στον αρχικό χώρο χαρακτηριστικών, οι SVM αξιοποιούν τη λεγόμενη τεχνική του kernel trick. Μέσω κατάλληλων συναρτήσεων πυρήνα (kernel functions), όπως ο πολυωνυμικός ή ο RBF (Gaussian) πυρήνας, τα δεδομένα προβάλλονται σε χώρο υψηλότερης διάστασης όπου ο γραμμικός διαχωρισμός καθίσταται εφικτός. Με τον τρόπο αυτό, το μοντέλο μπορεί να αποδώσει μη γραμμικά σύνορα απόφασης στον αρχικό χώρο.

Οι SVM παρουσιάζουν ιδιαίτερα καλή απόδοση σε προβλήματα υψηλής διάστασης, ακόμη και όταν ο αριθμός χαρακτηριστικών είναι μεγάλος σε σχέση με τον αριθμό των παρατηρήσεων. Ωστόσο, το

υπολογιστικό κόστος εκπαίδευσης αυξάνεται σημαντικά σε μεγάλα σύνολα δεδομένων, ιδίως όταν χρησιμοποιούνται μη γραμμικοί πυρήνες, γεγονός που περιορίζει τη χρήση τους σε περιβάλλοντα πολύ μεγάλου όγκου δεδομένων.

Η υλοποίηση των SVM παρέχεται από τη βιβλιοθήκη scikit-learn και ειδικότερα από την κλάση SVC του υποπακέτου sklearn.svm. Το μοντέλο επιτρέπει την επιλογή τύπου πυρήνα, καθώς και ρύθμιση υπερπαραμέτρων.

5.6.4.5 Εφαρμογή πολλαπλών αλγορίθμων κατηγοριοποίησης στο Titanic dataset

Ο κώδικας Κ. 5.6.9 υλοποιεί μια συγκριτική μελέτη πέντε διαφορετικών κατηγοριοποιητών στο πρόβλημα πρόβλεψης επιβίωσης επιβατών του Titanic. Αρχικά, το σύνολο δεδομένων φορτώνεται από τη βιβλιοθήκη seaborn με `sns.load_dataset("titanic")` και απομακρύνονται οι εγγραφές με ελλιπείς τιμές σε βασικά χαρακτηριστικά (`age`, `sex`, `fare`, `class`, `survived`). Στη συνέχεια πραγματοποιείται απλή κωδικοποίηση κατηγορικών μεταβλητών: το φύλο μετατρέπεται σε δυαδική αριθμητική μορφή (0 για άνδρες, 1 για γυναίκες) και η κατηγορία εισιτηρίου (First, Second, Third) στις αριθμητικές τιμές 1, 2 και 3. Ως χαρακτηριστικά επιλέγονται οι μεταβλητές `sex`, `age`, `fare` και `class`, ενώ η μεταβλητή-στόχος είναι η `survived`.

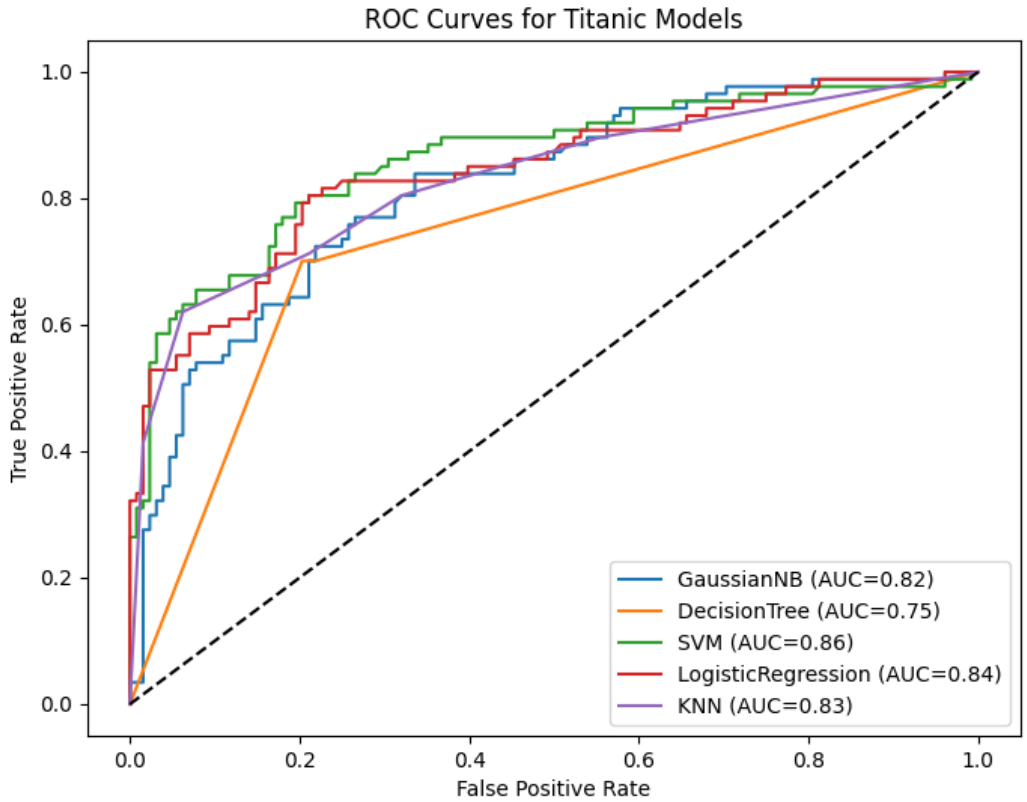
Τα δεδομένα χωρίζονται σε σύνολο εκπαίδευσης και ελέγχου με αναλογία 70%-30%, χρησιμοποιώντας train_test_split. Η παράμετρος stratify=y διασφαλίζει ότι η αναλογία επιζώντων και μη επιζώντων παραμένει ίδια και στα δύο σύνολα, γεγονός ιδιαίτερα σημαντικό σε προβλήματα δυαδικής κατηγοριοποίησης. Ακολουθεί κλιμάκωση των χαρακτηριστικών με StandardScaler, ώστε κάθε μεταβλητή να έχει μηδενικό μέσο και μοναδιαία διακύμανση. Η κλιμάκωση είναι κρίσιμη για αλγορίθμους που βασίζονται σε αποστάσεις ή σε εσωτερικά γινόμενα, όπως οι SVM, KNN και Logistic Regression. Το scaler εφαρμόζεται μόνο στα δεδομένα εκπαίδευσης (fit) και στη συνέχεια χρησιμοποιείται για τον μετασχηματισμό και των δεδομένων ελέγχου (transform).

Στη συνέχεια ορίζονται πέντε μοντέλα: Gaussian Naive Bayes, Decision Tree, Support Vector Machine, Logistic Regression και K-Nearest Neighbors. Για κάθε μοντέλο εκτελείται εκπαίδευση στο training set και πρόβλεψη στο test set. Παράλληλα υπολογίζονται βασικές μετρικές απόδοσης (accuracy, precision, recall, F1-score), οι οποίες αποθηκεύονται σε πίνακα για συγκριτική παρουσίαση. Επιπλέον, για κάθε μοντέλο υπολογίζεται η καμπύλη ROC και το αντίστοιχο εμβαδόν κάτω από την καμπύλη (AUC). Τέλος, οι καμπύλες όλων των μοντέλων σχεδιάζονται στο ίδιο γράφημα, επιτρέποντας άμεση οπτική σύγκριση (Εικόνα 27), ενώ οι αριθμητικές μετρικές παρουσιάζονται σε μορφή DataFrame (Εικόνα 28).

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    roc_curve,
    auc,
)
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

# 1. Φόρτωση του Titanic dataset από το seaborn
titanic = sns.load_dataset("titanic").dropna(
    subset=["age", "sex", "fare", "class", "survived"]
)

# 2. Προεπεξεργασία χαρακτηριστικών φύλο (sex) και κατηγορία εισιτηρίου
(class)
titanic["sex"] = titanic["sex"].map({"male": 0, "female": 1})
titanic["class"] = titanic["class"].map({"First": 1, "Second": 2, "Third":
3})

X = titanic[["sex", "age", "fare", "class"]]
y = titanic["survived"]

# 2. Διαχωρισμός σε train/test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# 3. Κλιμάκωση χαρακτηριστικών (σημαντικό για SVM, KNN, Logistic
Regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 4. Ορισμός μοντέλων
models = {
    "GaussianNB": GaussianNB(),
    "DecisionTree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "LogisticRegression": LogisticRegression(max_iter=1000,
random_state=42),
    "KNN": KNeighborsClassifier(n_neighbors=5),
}

# 5. Εκπαίδευση και πρόβλεψη για κάθε μοντέλο, υπολογισμός μετρικών
results = []
plt.figure(figsize=(8, 6))
for name, model in models.items():
    # Εκπαίδευση
    model.fit(X_train_scaled, y_train)
```

```
    # Πρόβλεψη
    y_pred = model.predict(X_test_scaled)
    y_prob = (
        model.predict_proba(X_test_scaled)[:, 1]
        if hasattr(model, "predict_proba")
        else model.decision_function(X_test_scaled)
    )

    # Υπολογισμός μετρικών
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    results.append([name, acc, prec, rec, f1])

    # Σχεδίαση καμπύλης ROC
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f"{name} (AUC={roc_auc:.2f})")

# ROC καμπύλες
plt.plot([0, 1], [0, 1], "k--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves for Titanic Models")
plt.legend()
plt.show()

# Εμφάνιση μετρικών
results_df = pd.DataFrame(
    results, columns=["Model", "Accuracy", "Precision", "Recall", "F1-
Score"]
)
print(results_df.round(3))
```

Κ. 5.6.9 – Εφαρμογή και σύγκριση αποτελεσμάτων 5 αλγορίθμων κατηγοριοποίησης στο dataset Titanic.



Εικόνα 27 – ROC καμπύλες για 5 αλγόριθμους κατηγοριοποίησης για το dataset Titanic.

	Model	Accuracy	Precision	Recall	F1-Score
0	GaussianNB	0.744	0.663	0.747	0.703
1	DecisionTree	0.758	0.701	0.701	0.701
2	SVM	0.791	0.792	0.655	0.717
3	LogisticRegression	0.767	0.713	0.713	0.713
4	KNN	0.758	0.697	0.713	0.705

Εικόνα 28 – Μετρικές 5 αλγορίθμων κατηγοριοποίησης για το dataset Titanic.

Συνεπώς, όπως φαίνεται από τις Εικόνα 27 και Εικόνα 28 ο κατηγοριοποιητής που έχει την καλύτερη επίδοση είναι ο SVM. Ωστόσο, υπολείπεται έναντι των άλλων κατηγοριοποιητών στη μετρική Recall που σημαίνει ότι έχει αυξημένο αριθμό False Negatives, δηλαδή ο κατηγοριοποιητής συμπεριφέρεται «συντηρητικά» στο να προβλέψει ότι κάποιος επέζησε.

5.7. Συσταδοποίηση

Η συσταδοποίηση (*clustering*) αποτελεί βασική τεχνική μη-επιβλεπόμενης μάθησης (*unsupervised learning*), στην οποία δεν υπάρχουν προκαθορισμένες ετικέτες για τα δεδομένα. Σε αντίθεση με την ταξινόμηση, όπου το μοντέλο εκπαιδεύεται με γνωστές κατηγορίες-στόχους, στη συσταδοποίηση ο αλγόριθμος καλείται να ανακαλύψει μόνος του τη δομή που υπάρχει στα δεδομένα και να

ομαδοποιήσει τα δείγματα σε συστάδες (clusters) με βάση την ομοιότητά τους. Η βασική ιδέα είναι ότι κάθε σημείο δεδομένων πρέπει τελικά να ανατεθεί σε μία ομάδα, έτσι ώστε τα σημεία που ανήκουν στην ίδια συστάδα να είναι «κοντά» μεταξύ τους, ενώ τα σημεία που ανήκουν σε διαφορετικές συστάδες να είναι, κατά το δυνατόν, «μακριά». Για να επιτευχθεί αυτό, ορίζεται κάποια μορφή μέτρου απόστασης ή ομοιότητας μεταξύ των παρατηρήσεων, όπως για παράδειγμα η Ευκλείδεια απόσταση σε έναν πολυδιάστατο χώρο χαρακτηριστικών. Η έννοια της «εγγύτητας» εξαρτάται από τη φύση των δεδομένων και την επιλογή της μετρικής.

Στο πλαίσιο της ανάλυσης δεδομένων με την Python, η συσταδοποίηση χρησιμοποιείται συχνά για διερευνητική ανάλυση (exploratory data analysis), για την αναγνώριση προτύπων, τμηματοποίηση πελατών και για ανίχνευση ανωμαλιών. Δημοφιλείς αλγόριθμοι, όπως ο k-means, η ιεραρχική συσταδοποίηση και ο DBSCAN, υλοποιούνται στο `scikit-learn`, επιτρέποντας την εφαρμογή τους με σχετικά απλό και συνεπή τρόπο.

5.7.1. Ο αλγόριθμος k-means

Ο αλγόριθμος **k-means** αποτελεί έναν από τους απλούστερους και πλέον διαδεδομένους αλγορίθμους συσταδοποίησης. Η βασική αρχή του είναι ότι κάθε συστάδα αναπαρίσταται από το κέντρο της (centroid), το οποίο ορίζεται ως ο μέσος όρος όλων των σημείων που ανήκουν σε αυτήν. Επιπλέον, κάθε σημείο δεδομένων ανήκει στη συστάδα της οποίας το κέντρο βρίσκεται πλησιέστερα σε αυτό σε σχέση με τα κέντρα των άλλων συστάδων. Με τον τρόπο αυτό, το πρόβλημα συσταδοποίησης μετατρέπεται σε πρόβλημα ελαχιστοποίησης του αθροίσματος των τετραγώνων των αποστάσεων κάθε σημείου από το αντίστοιχο κέντρο της συστάδας του.

Ο αλγορίθμος ξεκινά με την επιλογή του επιθυμητού αριθμού συστάδων k. Στη συνέχεια, επιλέγονται αρχικά k σημεία από το σύνολο των δεδομένων ως αρχικά κέντρα, συνήθως με τυχαίο τρόπο. Ακολουθούν δύο επαναλαμβανόμενα βήματα: α) κάθε σημείο ανατίθεται στη συστάδα του πλησιέστερου κέντρου και β) για κάθε συστάδα υπολογίζεται νέο κέντρο ως ο μέσος όρος των σημείων που της έχουν ανατεθεί. Τα δύο αυτά βήματα επαναλαμβάνονται έως ότου οι αναθέσεις δεν αλλάζουν πλέον ή η μεταβολή των κέντρων γίνει αμελητέα. Η διαδικασία αυτή έχει αποδειχθεί ότι συγκλίνει. Ωστόσο, το τελικό αποτέλεσμα εξαρτάται από την αρχική επιλογή των κέντρων, γεγονός που καθιστά χρήσιμες τεχνικές όπως η πολλαπλή αρχικοποίηση ή η μέθοδος k-means++.

5.7.1.1 Παράδειγμα εφαρμογής του k-means σε συνθετικά δεδομένα

Στο παράδειγμα αυτό παρουσιάζεται η εφαρμογή του αλγορίθμου k-means σε συνθετικά δεδομένα, με στόχο την κατανόηση της λειτουργίας του. Τα βήματα που θα ακολουθηθούν είναι τα ακόλουθα:

1. Δημιουργία και απεικόνιση συνθετικών δεδομένων.

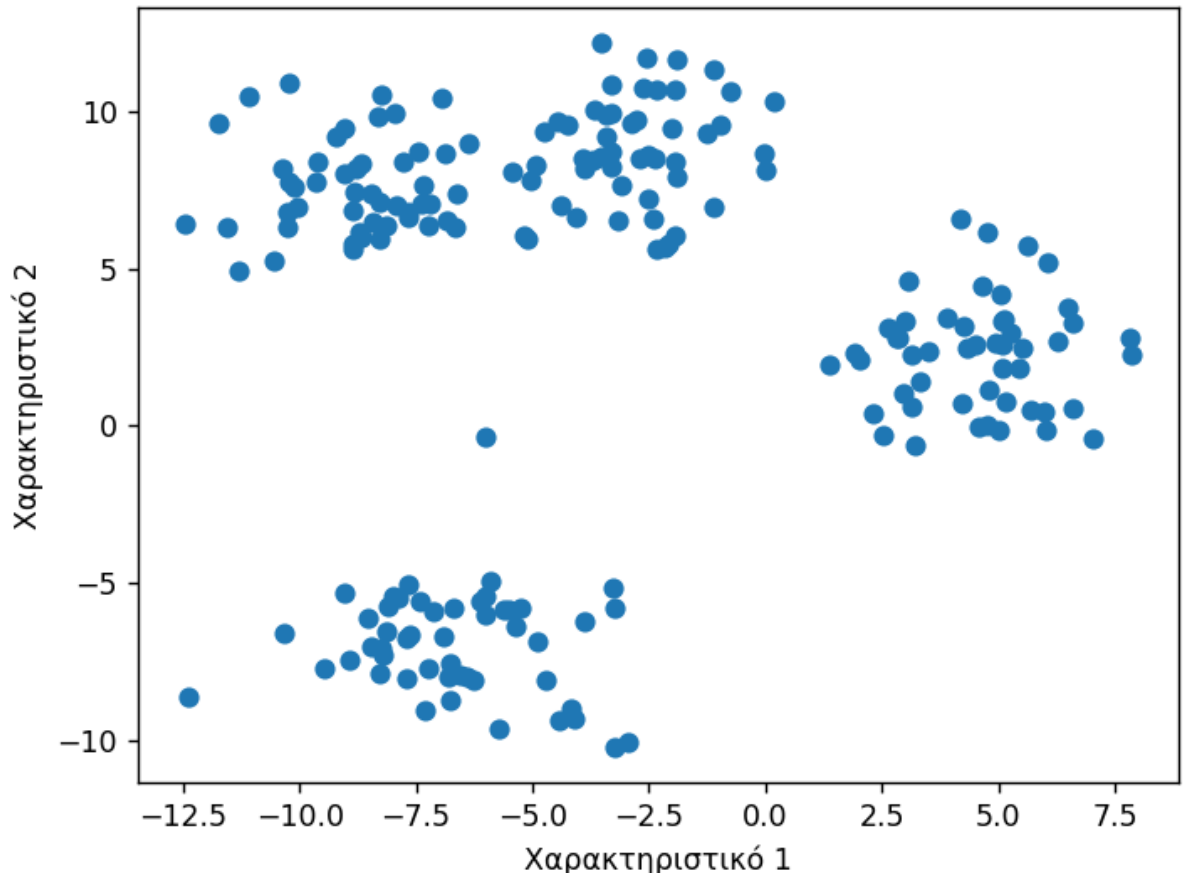
2. Εφαρμογή k-means για εντοπισμό συστάδων και γραφική απεικόνισή τους.
3. Υπολογισμός της μετρικής silhouette_score για την εκτίμηση της ποιότητας της συσταδοποίησης που έχει επιτευχθεί.

Το παράδειγμα παρουσιάζεται σταδιακά με τμήματα κώδικα που αντιστοιχούν στα κελιά ενός Jupyter σημειωματαρίου. Στον κώδικα Κ. 5.7.1 δημιουργείται ένα τεχνητό σύνολο δεδομένων με τη συνάρτηση make_blobs από το υποπακέτο sklearn.datasets. Οι παράμετροι centers=4, n_samples=200, cluster_std=1.7 και random_state=42 καθορίζουν αντίστοιχα τον αριθμό των πραγματικών συστάδων, το πλήθος των δειγμάτων, τη διασπορά γύρω από κάθε κέντρο και τη δυνατότητα αναπαραγωγής των ίδιων δεδομένων. Με τον τρόπο αυτό παράγονται δεδομένα που συγκεντρώνονται γύρω από τέσσερα διακριτά κέντρα, χωρίς όμως να είναι εκ των προτέρων γνωστά στον αλγόριθμο.

```
# 1. Δημιουργία και απεικόνιση συνθετικών δεδομένων
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

X, y = make_blobs(centers=4, n_samples=200, random_state=42,
cluster_std=1.7)
plt.scatter(X[:, 0], X[:, 1])
plt.xlabel("Χαρακτηριστικό 1")
plt.ylabel("Χαρακτηριστικό 2")
plt.show()
```

Κ. 5.7.1 – Δημιουργία συνθετικών δεδομένων με την make_blobs().

Η απεικόνιση των σημείων που δημιουργούνται παρουσιάζεται στο γράφημα της Εικόνα 29.

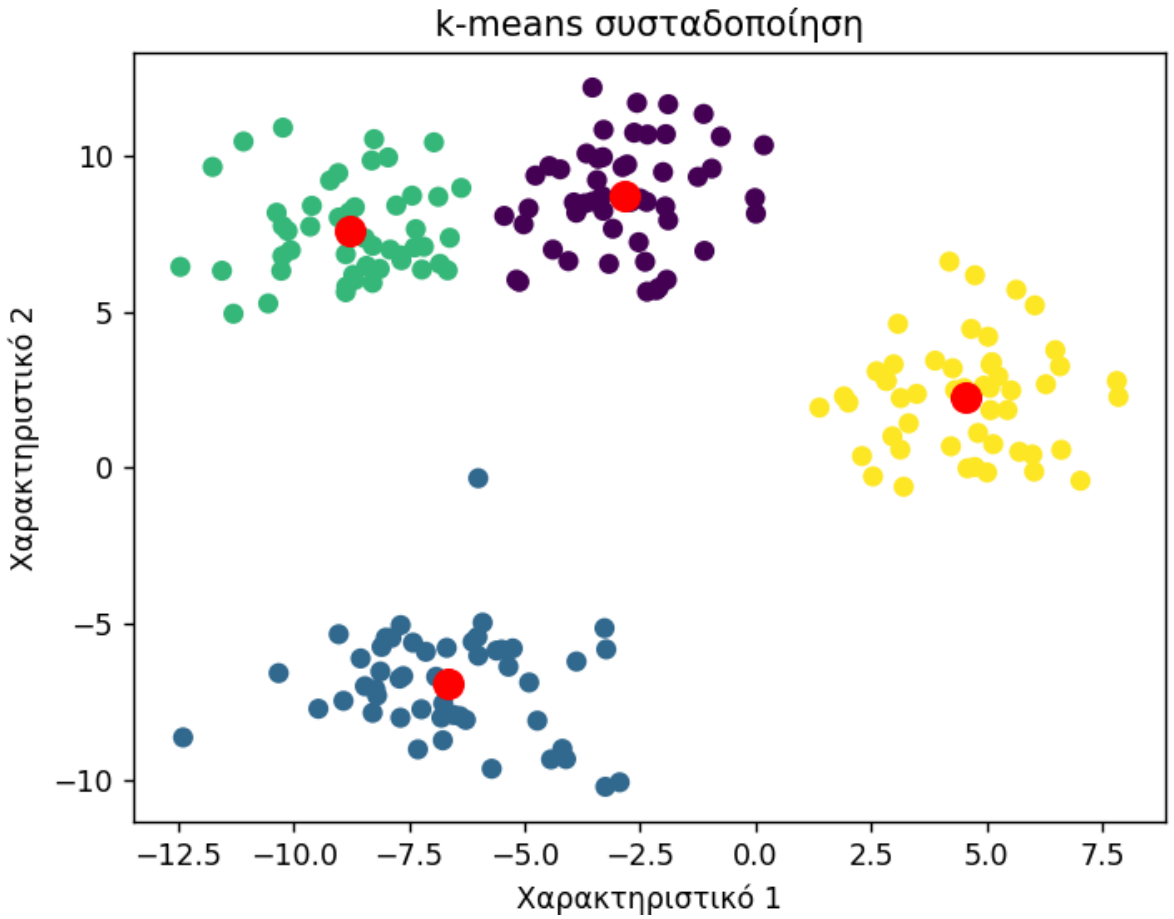


Εικόνα 29 – Απεικόνιση των 2D συνθετικών δεδομένων.

Το επόμενο βήμα είναι η εφαρμογή του αλγορίθμου k-means για τον εντοπισμό των συστάδων. Αυτό συμβαίνει στον κώδικα Κ. 5.7.2 όπου εφαρμόζεται ο αλγόριθμος k-means επιλέγοντας ως υπερπαράμετρο τον αριθμό συστάδων $k = 4$. Ο αλγόριθμος υπολογίζει τα κέντρα και αναθέτει κάθε σημείο στη συστάδα του πλησιέστερου κέντρου, σύμφωνα με την Ευκλείδεια απόσταση. Ακολούθως, τα ίδια δεδομένα απεικονίζονται εκ νέου, στην Εικόνα 30, αυτή τη φορά με διαφορετικό χρώμα για κάθε συστάδα, ώστε να είναι ορατό το αποτέλεσμα της συσταδοποίησης.

```
# 2. Εφαρμογή k-means για εντοπισμό και εμφάνιση συστάδων
from sklearn.cluster import KMeans

model = KMeans(4)
model.fit(X)
plt.scatter(X[:, 0], X[:, 1], c=model.labels_)
plt.scatter(
    model.cluster_centers_[:, 0], model.cluster_centers_[:, 1], s=100,
color="red"
)
plt.title("k-means συσταδοποίηση")
plt.xlabel("Χαρακτηριστικό 1")
plt.ylabel("Χαρακτηριστικό 2")
```

Κ. 5.7.2 – Εφαρμογή του k-means.



Εικόνα 30 – Οι συστάδες που εντοπίζει ο k-means και τα κέντρα τους.

Τέλος, στον κώδικα Κ. 5.7.3 υπολογίζεται η μετρική silhouette score, η οποία μετρά πόσο καλά κάθε σημείο ταιριάζει στη δική του συστάδα σε σχέση με τις γειτονικές. Υψηλές τιμές του δείκτη υποδηλώνουν καλή συνοχή εντός συστάδων και σαφή διαχωρισμό μεταξύ διαφορετικών συστάδων.

```
# 3. Υπολογισμός Silhouette Score
from sklearn.metrics import silhouette_score

score = silhouette_score(X, model.labels_)
print("Silhouette score:", score)  # Silhouette score: 0.6489658465525686
```

Κ. 5.7.3 – Υπολογισμός της μετρικής silhouette.

5.7.2. Ιεραρχική συσταδοποίηση

Η ιεραρχική συσταδοποίηση (*hierarchical clustering*) αποτελεί μια διαφορετική προσέγγιση στη μη-επιβλεπόμενη μάθηση, η οποία βασίζεται στη σταδιακή συγχώνευση ή διάσπαση συστάδων, δημιουργώντας μια δενδροειδή αναπαράσταση της δομής των δεδομένων. Στην πιο συνηθισμένη μορφή της, την λεγόμενη **agglomerative**[3] συσταδοποίηση, η διαδικασία ξεκινά τοποθετώντας κάθε σημείο δεδομένων στη δική του συστάδα. Έπειτα, σε κάθε επανάληψη, δύο συστάδες που

[3] Agglomerative σημαίνει “χτίζω προς τα πάνω συνδυάζοντας μικρά τμήματα”.

θεωρούνται πιο «κοντινές» με βάση κάποιο κριτήριο απόστασης συγχωνεύονται. Η διαδικασία συνεχίζεται μέχρι να απομείνει ο επιθυμητός αριθμός συστάδων.

Όπως και στον αλγόριθμο k-means, απαιτείται πρώτα ένα μέτρο απόστασης μεταξύ σημείων, που συνήθως είναι η Ευκλείδεια απόσταση. Ωστόσο, στην ιεραρχική συσταδοποίηση χρειάζεται να προσδιοριστεί και ο τρόπος με τον οποίο υπολογίζεται η απόσταση μεταξύ δύο συστάδων. Το κριτήριο αυτό, γνωστό ως **linkage,** καθορίζει τον τρόπο που συγκρίνονται ομάδες σημείων και οδηγεί σε διαφορετικές μορφές συσταδοποίησης. Τα πλέον συνθισμένα κριτήρια linkage είναι τα:

Single linkage: η απόσταση δύο συστάδων U και V ορίζεται ως η ελάχιστη απόσταση μεταξύ οποιουδήποτε ζεύγους σημείων των δύο συστάδων.

$$d(U,V) = \min_{u \in U,\, v \in V} d(u,v)$$

Complete linkage: η απόσταση ορίζεται ως η μέγιστη απόσταση μεταξύ σημείων των δύο συστάδων.

$$d(U,V) = \max_{u \in U,\, v \in V} d(u,v)$$

Average linkage: η απόσταση ορίζεται ως ο μέσος όρος όλων των αποστάσεων μεταξύ σημείων των δύο συστάδων.

$$d(U,V) = \frac{1}{|U|\ |V|} \sum_{u \in U,\, v \in V} d(u,v)$$

Ward linkage: επιλέγει τη συγχώνευση που ελαχιστοποιεί την αύξηση του αθροίσματος των τετραγωνικών σφαλμάτων στην κάθε συστάδα.

5.7.2.1 Παράδειγμα ιεραρχικής συσταδοποίησης σε συνθετικά δεδομένα

Ως παράδειγμα εφαρμογής της ιεραρχικής συσταδοποίησης θα παρουσιαστεί το ίδιο παράδειγμα συσταδοποίησης συνθετικών δεδομένων που αντιμετωπίστηκε στην παράγραφο 5.7.1.1 με τον k-means. Ο κώδικας Κ. 5.7.4 δημιουργεί αρχικά τα συνθετικά δεδομένα και στη συνέχεια εφαρμόζει ιεραρχική (agglomerative) συσταδοποίηση για τέσσερις διαφορετικές μεθόδους linkage (single, complete, average, ward), παράγοντας τέσσερα διαγράμματα διασποράς στα οποία τα σημεία χρωματίζονται ανάλογα με το cluster στο οποίο ανήκουν (Εικόνα 31). Παράλληλα υπολογίζεται ο δείκτης silhouette που ποσοτικοποιεί την ποιότητα του διαχωρισμού συστάδων που επιτυγχάνεται. Στη συνέχεια σχεδιάζονται τα αντίστοιχα δενδρογράμματα (dendrograms), τα οποία απεικονίζουν τη διαδοχική συγχώνευση των συστάδων και την απόσταση στην οποία αυτή πραγματοποιείται (Εικόνα

32). Ο κατακόρυφος άξονας εκφράζει την απόσταση συγχώνευσης, ενώ η οριζόντια διάταξη των φύλλων αντιστοιχεί στα αρχικά δείγματα. Η διακεκομμένη οριζόντια γραμμή που εμφανίζεται σε κάθε δενδρόγραμμα αναπαριστά το «ύψος κοπής» (cut height), δηλαδή την τιμή απόστασης στην οποία αν διακοπεί το δέντρο προκύπτουν ακριβώς τέσσερις συστάδες, συνδέοντας έτσι οπτικά τη δενδρική αναπαράσταση με τα τέσσερα clusters που εμφανίζονται στα αντίστοιχα διαγράμματα διασποράς. Για να συμβεί αυτό χρησιμοποιείται η βιβλιοθήκη scipy της Python που παρέχει συναρτήσεις για ιεραρχική συσταδοποίηση και κατασκευή δενδρογραμμάτων, όπως οι linkage και dendrogram από το υποπακέτο scipy.cluster.hierarchy, οι οποίες υπολογίζουν τη διαδοχική συγχώνευση των συστάδων και επιτρέπουν την οπτικοποίηση της δενδρικής δομής τους.

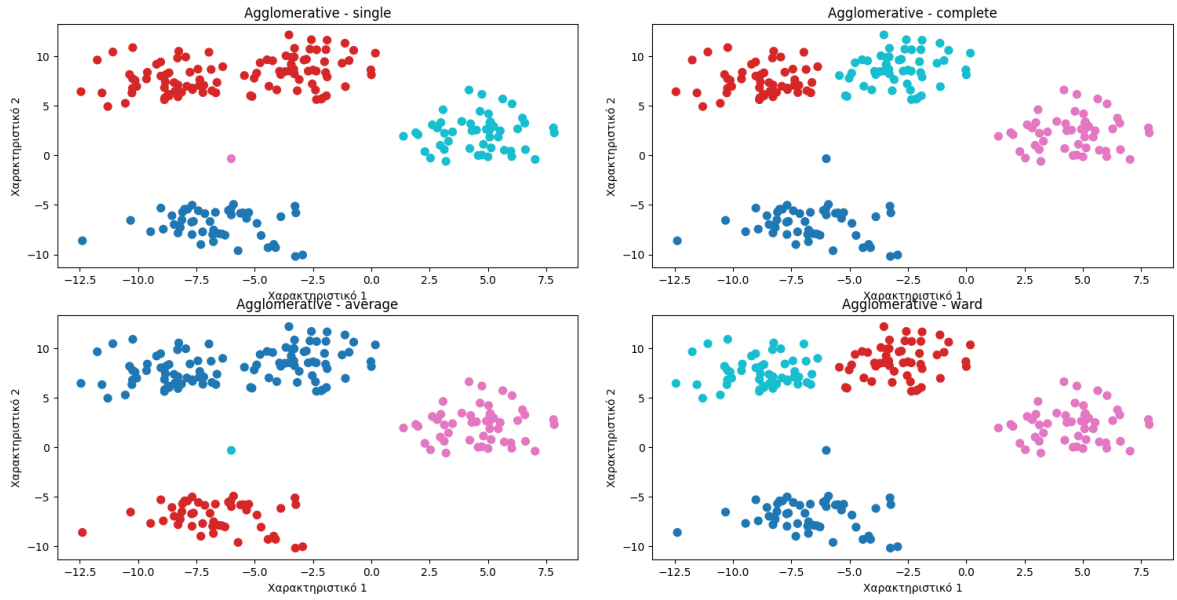
```
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import dendrogram, linkage

# 1. Δημιουργία συνθετικών δεδομένων
X, y = make_blobs(centers=4, n_samples=200, random_state=42,
cluster_std=1.7)


# 2. Ιεραρχική συσταδοποίηση με 4 διαφορετικά linkages
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
linkage_methods = ["single", "complete", "average", "ward"]
for ax, method in zip(axes.ravel(), linkage_methods):
    model = AgglomerativeClustering(n_clusters=4, linkage=method)
    labels = model.fit_predict(X)
    ax.scatter(X[:, 0], X[:, 1], c=labels, cmap="tab10", s=50)
    ax.set_title(f"Agglomerative - {method}")
    ax.set_xlabel("Χαρακτηριστικό 1")
    ax.set_ylabel("Χαρακτηριστικό 2")
    score = silhouette_score(X, labels)
    print(f"Linkage = {method:8s}  Silhouette score = {score:.3f}")
plt.tight_layout()
plt.show()

# 3. Σχεδίαση των 4 dendrograms
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
for ax, method in zip(axes.ravel(), linkage_methods):
    Z = linkage(X, method=method)
    n = X.shape[0]
    k = 4
    cut_height = Z[n - k, 2]
    dendrogram(Z, ax=ax, no_labels=True)
    ax.axhline(y=cut_height, linestyle="--")
    ax.set_title(f"Dendrogram - {method}")
    ax.set_ylabel("Απόσταση")
plt.tight_layout()
plt.show()
```

Κ. 5.7.4 – Ιεραρχική συσταδοποίηση για 4 διαφορετικά linkages (single, complete, average, ward).

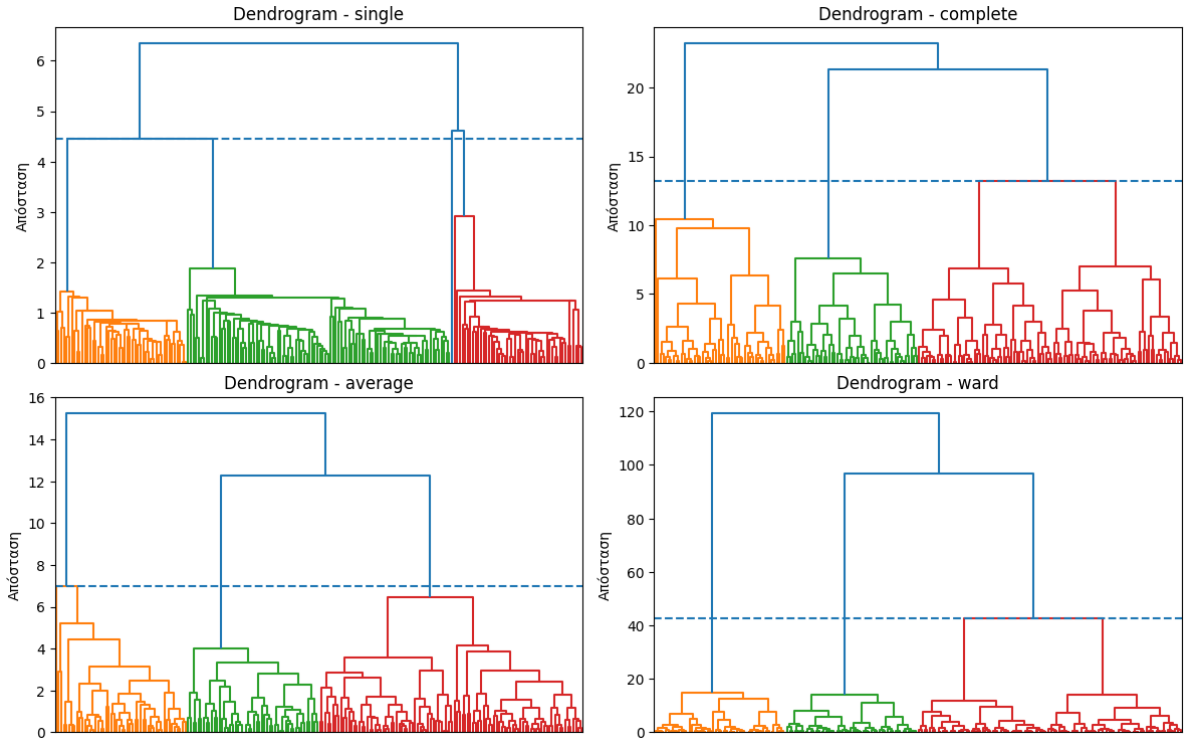
Η έξοδος που εμφανίζεται κατά την εκτέλεση του κώδικα είναι η ακόλουθη:

```
Linkage = single    Silhouette score = 0.555
```

```
Linkage = complete  Silhouette score = 0.649
Linkage = average   Silhouette score = 0.555
Linkage = ward      Silhouette score = 0.649
```



Εικόνα 31 – Αποτελέσματα συσταδοποίησης με ιεραρχική συσταδοποίηση για 4 διαφορετικά linkages.



Εικόνα 32 – Δενδρογράμματα ιεραρχικής συσταδοποίησης για 4 διαφορετικά linkages.

5.8. Ασκήσεις

5.8.1. Άσκηση 1

Στόχος της άσκησης είναι η εκπαίδευση και αξιολόγηση ενός μοντέλου γραμμικής παλινδρόμησης για την πρόβλεψη της μέσης τιμής κατοικίας (median house value) στο σύνολο δεδομένων California Housing. Επιπλέον, ζητείται η διερεύνηση της επίδρασης της κανονικοποίησης (feature scaling) στις επιδόσεις του μοντέλου.

Για τη φόρτωση των δεδομένων να χρησιμοποιηθεί ο ακόλουθος κώδικας:

```
from sklearn.datasets import fetch_california_housing
california_housing = fetch_california_housing(as_frame=True)
```

Να υλοποιηθούν τα παρακάτω βήματα:

1. Φόρτωση και αρχική διερεύνηση δεδομένων
    - Να εξαχθούν τα χαρακτηριστικά (X) και η μεταβλητή-στόχος (y).
    - Να παρουσιαστούν οι διαστάσεις του συνόλου δεδομένων.
    - Να εμφανιστούν οι πρώτες γραμμές του πίνακα.
2. Διαχωρισμός σε σύνολα εκπαίδευσης και ελέγχου
    - Να πραγματοποιηθεί διαχωρισμός σε σύνολο εκπαίδευσης και σύνολο ελέγχου με αναλογία 80% - 20%.
    - Να παρουσιαστούν οι διαστάσεις των επιμέρους συνόλων.
3. Εκπαίδευση μοντέλου
    - Να προσαρμοστεί μοντέλο γραμμικής παλινδρόμησης στα δεδομένα εκπαίδευσης.
4. Πρόβλεψη
    - Να υπολογιστούν οι προβλέψεις του μοντέλου για τα δεδομένα ελέγχου.
5. Αξιολόγηση
    - Να υπολογιστούν το Mean Squared Error (MSE) και ο συντελεστής προσδιορισμού R^2.
6. Κανονικοποίηση χαρακτηριστικών
    - Να εφαρμοστεί κανονικοποίηση με χρήση της κλάσης StandardScaler.
    - Στη συνέχεια να επαναληφθούν τα βήματα 3–5 με τα κανονικοποιημένα δεδομένα και να πραγματοποιηθεί σύγκριση των αποτελεσμάτων με εκείνα του αρχικού μοντέλου.

5.8.2. Άσκηση 2

Μελετήστε το παράδειγμα που παρουσιάζεται στο «Μια οπτική εισαγωγή στη μηχανική μάθηση» - https://r2d3.us/οπτική-εισαγωγή-στη-μηχανική-μάθηση-μέρος-1/. Στην άσκηση αυτή θα δημιουργηθεί ένα απλοποιημένο αντίστοιχο σενάριο που αφορά 2 υποθετικές πόλεις CITYA και CITYB. Θα δημιουργηθούν συνθετικά δεδομένα με features τα elevation, price, sqft, bedrooms και με label το city και θα διαχωριστούν σε σύνολα εκπαίδευσης και ελέγχου. Στη συνέχεια θα εκπαιδευτούν δύο κατηγοριοποιητές (classifiers), και θα εξεταστούν μετρικές απόδοσης που θα υπολογιστούν στο σύνολο ελέγχου.

Να χρησιμοποιηθεί ο ακόλουθος κώδικας (Κ. 5.8.1) για τη δημιουργία των συνθετικών δεδομένων:

```
import numpy as np
import pandas as pd

np.random.seed(7)
N = 500
# CITYA
citya_elev = np.random.normal(loc=80, scale=30, size=N)
citya_price = np.random.normal(loc=1100, scale=180, size=N)
citya_sqft = np.random.normal(loc=900, scale=120, size=N)
citya_beds = np.random.randint(1, 5, size=N)
# CITYB
cityb_elev = np.random.normal(loc=50, scale=25, size=N)
cityb_price = np.random.normal(loc=950, scale=150, size=N)
cityb_sqft = np.random.normal(loc=800, scale=80, size=N)
cityb_beds = np.random.randint(1, 4, size=N)

df = pd.DataFrame(
    {
        "elevation": np.concatenate([citya_elev, cityb_elev]),
        "price": np.concatenate([citya_price, cityb_price]),
        "sqft": np.concatenate([citya_sqft, cityb_sqft]),
        "bedrooms": np.concatenate([citya_beds, cityb_beds]),
        "city": ["CITYA"] * N + ["CITYB"] * N,
    }
)
```

Κ. 5.8.1 – Δημιουργία συνθετικών δεδομένων για την άσκηση 2.

Να υλοποιηθούν τα ακόλουθα βήματα:

1. Να σχεδιαστεί ένα διάγραμμα διασποράς "elevation vs. price" για τις πόλεις CITYA και CITYB.
2. Να πραγματοποιηθεί διαχωρισμός σε σύνολο εκπαίδευσης και σε σύνολο ελέγχου με αναλογία 75%-25% (random_state=7). Να εμφανιστούν οι διαστάσεις των συνόλων και το πλήθος ανά κλάση (CITYA ή CITYB) σε καθένα από τα δύο σύνολα.
3. Να εκπαιδευτεί ένα δένδρο αποφάσεων (DecisionTreeClassifier με max_depth=4, random_state=7). Για τα δεδομένα ελέγχου να υπολογιστεί το accuracy, το precision, το recall και το f-score και να σχεδιαστεί με το seaborn ο πίνακας σύγχυσης.

4. Να εκπαιδευτεί ένας κατηγοριοποιητής λογιστικής παλινδρόμησης (LogisticRegression). Να υπολογιστούν οι ίδιες μετρικές απόδοσης με αυτές που υπολογίστηκαν για το DecisionTreeClassifier στο προηγούμενο ερώτημα και να σχεδιαστεί ο πίνακας σύγχυσης. Ποιος κατηγοριοποιητής συμπεριφέρεται καλύτερα;

5.9. Ερωτήσεις αυτοαξιολόγησης

1. Ποια από τις παρακάτω περιπτώσεις αποτελεί παράδειγμα επιβλεπόμενης μάθησης;
    α) Ομαδοποίηση πελατών με βάση τη συμπεριφορά αγορών
    β) Μείωση διαστάσεων με PCA (Principal Component Analysis)
    γ) Πρόβλεψη τιμής κατοικίας με βάση χαρακτηριστικά της
    δ) Εύρεση δομής δεδομένων χωρίς ετικέτες

2. Ποιος είναι ο στόχος της γραμμικής παλινδρόμησης;
    α) Η πρόβλεψη κατηγορίας
    β) Η εκτίμηση συνεχούς μεταβλητής-στόχου
    γ) Η ομαδοποίηση δεδομένων
    δ) Η επιλογή χαρακτηριστικών

3. Ποια από τις παρακάτω μετρικές χρησιμοποιείται σε προβλήματα κατηγοριοποίησης;
    α) Mean Squared Error (MSE)
    β) R^2
    γ) Accuracy
    δ) Root Mean Squared Error (RMSE)

4. Ποιο από τα παρακάτω αποτελεί παράδειγμα μη-επιβλεπόμενης μάθησης;
    α) Logistic Regression
    β) Decision Tree Classifier
    γ) k-means
    δ) Linear Regression

5. Στον αλγόριθμο k-means, το k δηλώνει:
    α) Τον αριθμό χαρακτηριστικών
    β) Τον αριθμό επαναλήψεων
    γ) Τον αριθμό δειγμάτων
    δ) Τον αριθμό συστάδων

6. Ποια είναι η βασική μέθοδος εκπαίδευσης ενός μοντέλου στο scikit-learn;
    α) fit()
    β) train()
    γ) learn()
    δ) optimize()

7. Ποιο module του scikit-learn περιλαμβάνει την κλάση LinearRegression;
    α) sklearn.metrics
    β) sklearn.cluster
    γ) sklearn.preprocessing
    δ) sklearn.linear_model

8. Ποια συνάρτηση του scikit-learn χρησιμοποιείται για διαχωρισμό δεδομένων σε σύνολο εκπαίδευσης και ελέγχου;
    α) split_dataset()
    β) train_test_split()
    γ) cross_val_split()
    δ) data_partition()

9. Ποια κλάση του scikit-learn χρησιμοποιείται για κανονικοποίηση χαρακτηριστικών (standardization);

α) StandardScaler
β) Normalizer
γ) RobustTransformer
δ) MinMaxScaler

10. Ποιο module του scikit-learn περιλαμβάνει τη συνάρτηση silhouette_score;

α) sklearn.preprocessing
β) sklearn.model_selection
γ) sklearn.decomposition
δ) sklearn.metrics

11. Η ιεραρχική συσταδοποίηση μπορεί να παρουσιαστεί γραφικά μέσω:

α) ROC curve
β) Dendrogram
γ) Scatter matrix
δ) Boxplot

12. Στα προβλήματα κατηγοριοποίησης, ο πίνακας σύγχυσης (confusion matrix) περιλαμβάνει:

α) Μόνο σωστές προβλέψεις
β) Τετραγωνικά σφάλματα
γ) Πραγματικές και προβλεπόμενες κλάσεις
δ) Τιμές πιθανοτήτων

13. Η μετρική R^2 εκφράζει:

α) Το ποσοστό σωστών ταξινομήσεων
β) Τη μέση απόλυτη απόκλιση
γ) Τον αριθμό των χαρακτηριστικών
δ) Το ποσοστό διακύμανσης της εξαρτημένης μεταβλητής που εξηγείται από το μοντέλο

14. Ποια από τις παρακάτω ποσότητες ελαχιστοποιείται συνήθως στη γραμμική παλινδρόμηση;

α) Η ακρίβεια (accuracy)
β) Το άθροισμα τετραγώνων των σφαλμάτων
γ) Η εντροπία
δ) Η απόσταση Manhattan

15. Σε ένα πρόβλημα πολλαπλής γραμμικής παλινδρόμησης:

α) Υπάρχει μία μόνο ερμηνευτική μεταβλητή
β) Η μεταβλητή-στόχος είναι κατηγορική
γ) Υπάρχουν περισσότερες από μία ερμηνευτικές μεταβλητές
δ) Το μοντέλο είναι πάντα μη γραμμικό

16. Στο scikit-learn ποια μέθοδος καλείται συνήθως μετά την fit();

α) predict()
β) fit_next()
γ) learn()
δ) compile()

17. Ποια ιδιότητα ενός εκπαιδευμένου LinearRegression μοντέλου περιέχει τους συντελεστές παλινδρόμησης;

α) weights_
β) beta_
γ) coef_
δ) parameters_

18. Τι υπολογίζει ο παρακάτω κώδικας;

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

α) Τις πιθανότητες κλάσεων
β) Τις προβλέψεις συνεχούς μεταβλητής για το X_test
γ) Τις συστάδες των δεδομένων
δ) Το R^2 στο training set

19. Τι επιστρέφει η μεταβλητή score στον παρακάτω κώδικα;

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)
score = model.score(X_test, y_test)
```

α) Το R^2 στο test set
β) Το Mean Squared Error
γ) Την accuracy
δ) Τον αριθμό παραμέτρων

20. Ποιο είναι το λάθος στον ακόλουθο κώδικα;

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
y_pred = model.predict(X_test)
model.fit(X_train, y_train)
```

α) Η LinearRegression δεν υποστηρίζει τη μέθοδο predict()
β) Πρέπει να γίνει πρώτα fit() στο μοντέλο και μετά predict()
γ) Πρέπει να γίνει πρώτα predict() και μετά fit()
δ) Η fit() χρειάζεται μόνο X_train, όχι y_train

5.9.1. Απαντήσεις στις ερωτήσεις αυτοαξιολόγησης

1. γ) Πρόβλεψη τιμής κατοικίας με βάση χαρακτηριστικά της
2. β) Η εκτίμηση συνεχούς μεταβλητής-στόχου
3. γ) Accuracy
4. γ) k-means
5. δ) Τον αριθμό συστάδων
6. α) fit()
7. δ) sklearn.linear_model
8. β) train_test_split()
9. α) StandardScaler
10. δ) sklearn.metrics
11. β) Dendrogram
12. γ) Πραγματικές και προβλεπόμενες κλάσεις
13. δ) Το ποσοστό διακύμανσης της εξαρτημένης μεταβλητής που εξηγείται από το μοντέλο
14. β) Το άθροισμα τετραγώνων των σφαλμάτων
15. γ) Υπάρχουν περισσότερες από μία ερμηνευτικές μεταβλητές
16. α) predict()
17. γ) coef_
18. β) Τις προβλέψεις συνεχούς μεταβλητής για το X_test
19. α) Το R^2 στο test set
20. β) Πρέπει να γίνει πρώτα fit() στο μοντέλο και μετά predict()

ΒΙΒΛΙΟΓΡΑΦΙΑ

- **Bruce, P., Bruce, A., & Gedeck, P. (2020).** *Practical statistics for data scientists: 50+ essential concepts using R and Python* (2nd ed.). O'Reilly Media.
- **Geron, A. (2022).** *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow* (3rd ed.). O'Reilly Media.
- **Heydt, M. (2017).** *Learning pandas: High-performance data manipulation and analysis using Python* (2nd ed.). Packt Publishing.
- **Karau, H., & Warren, R. (2019).** *Ανάλυση δεδομένων με Python* (Κ. Γραμμένος, Μεταφρ.). Εκδόσεις Μ. Γκιούρδας.
- **McKinney, W. (2022).** *Python for data analysis: Data wrangling with pandas, NumPy, and Jupyter* (3rd ed.). O'Reilly Media.
- **Molenaar, J. (2021).** *Pandas in action*. Manning Publications.
- **Petrou, T. (2020).** *Pandas cookbook: Recipes for scientific computing, time series analysis and data visualization using Python* (2nd ed.). Packt Publishing.
- **Stephens, J. (2022).** *Pandas 1.x cookbook: Over 100 recipes for using pandas to answer questions from your data* (2nd ed.). Packt Publishing.
- **VanderPlas, J. (2021).** *Python Data Science Handbook: Βασικά εργαλεία για την επεξεργασία δεδομένων* (Μπ. Ζαπανιώτης, Μεταφρ.). Εκδόσεις Κλειδάριθμος.
- **VanderPlas, J. (2022).** *Python data science handbook: Essential tools for working with data* (2nd ed.). O'Reilly Media.
- **Σιόλας, Γ., & Σταμέλος, Ι. (2021).** *Εισαγωγή στην ανάλυση δεδομένων με την Python*. Εκδόσεις Τζιόλα.