ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

..... Polars: Μια Σύγχρονη Εναλλακτική των Pandas ......

..... Σπάρθι Άλεξ.....

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

.....Δαδαλιάρης Αντώνιος.....
.....Βαθμίδα.....

ΣΥΝΕΠΙΒΛΕΠΩΝ
(εφόσον υπάρχει)

.....Ονοματεπώνυμο Υπεύθυνου.....
.....Βαθμίδα.....

Λαμία 8/7/2025 έτος 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

..... Polars: Μια Σύγχρονη Εναλλακτική των Pandas.....

..... Σπάρθι Άλεξ.....

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

.....Δαδαλιάρης Αντώνιος.....
.....Βαθμίδα.....

ΣΥΝΕΠΙΒΛΕΠΩΝ
(εφόσον υπάρχει)

.....Ονοματεπώνυμο Υπεύθυνου.....
.....Βαθμίδα.....

Λαμία 8/7/2025 έτος 2025

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

..... Polars, a contemporary aternative to Pandas.....

..... Σπάρθι Άλεξ.....

FINAL THESIS

ADVISOR

.....Δαδαλιάρης Αντώνιος.....
.....Position.....

CO ADVISOR

.....Full Name.....
.....Position.....

Lamia .....8-7-2025..... year 2025.....

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

*1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

*2. Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

*3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, **κλπ**), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια*

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: …5…/..7…/2025……

Ο - Η Δηλ.

``

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση
του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων
σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία εξετάζει τη σύγχρονη βιβλιοθήκη **Polars** ως μία αποδοτική εναλλακτική του **Pandas** για την ανάλυση δομημένων δεδομένων σε περιβάλλον **Python**. Το **Polars**, υλοποιημένο σε **Rust** και βασισμένο στο **columnar format Apache Arrow**, στοχεύει να υπερκεράσει τους περιορισμούς του **Pandas** σε ταχύτητα, κατανάλωση μνήμης και κλιμάκωση. Η εργασία αναλύει την τεχνική αρχιτεκτονική του **Polars**, τη **lazy** αξιολόγηση και τις τεχνικές βελτιστοποίησης που χρησιμοποιεί, ενώ παρουσιάζει συγκριτικά αποτελέσματα απόδοσης έναντι του **Pandas** και άλλων εργαλείων όπως **Dask**, **Spark**, **cuDF** και **Vaex**.

Ιδιαίτερη έμφαση δίνεται στη **διαλειτουργικότητα** του **Polars** με το οικοσύστημα της **Python**, καθώς και στη χρήση του σε πραγματικούς τομείς εφαρμογής όπως η **βιοπληροφορική**, τα **web logs**, και οι χρηματοοικονομικές **χρονοσειρές**. Επιπλέον, αναδεικνύεται η ενσωμάτωση του **Polars** σε **data pipelines** παραγωγής και η συνεισφορά του στην ενεργειακή αποδοτικότητα. Τέλος, η εργασία εξετάζει μειονεκτήματα, την καμπύλη εκμάθησης και τις μελλοντικές εξελίξεις της βιβλιοθήκης, καταλήγοντας στο συμπέρασμα ότι το **Polars** αποτελεί ένα σύγχρονο, ταχύ και ευέλικτο εργαλείο ανάλυσης δεδομένων, ικανό να καλύψει τις αυξανόμενες ανάγκες της βιομηχανίας και της έρευνας.

ABSTRACT

This thesis explores *Polars*, a modern data analysis library that offers a highly performant alternative to Pandas within the Python ecosystem. Built in Rust and based on the columnar Apache Arrow format, Polars addresses Pandas' limitations in speed, memory efficiency, and scalability. The work analyzes Polars' technical architecture, its lazy evaluation model, and advanced query optimization techniques, while presenting comparative performance results against Pandas and other libraries such as Dask, Spark, cuDF, and Vaex.

Particular attention is given to Polars' interoperability with the broader Python data science ecosystem, as well as its application in real-world domains such as bioinformatics, web log analytics, and financial time series. The integration of Polars into production-grade data pipelines and its contribution to energy efficiency are also highlighted. Finally, the thesis examines the current limitations, learning curve, and future development roadmap of the library. The findings conclude that Polars is a modern, fast, and flexible tool for data analysis, well-suited to meet the growing demands of both industry and research.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

ΚΕΦΑΛΑΙΟ 1 Εισαγωγή

1.1 Η ανάγκη για εναλλακτικές λύσεις αντί του Pandas

Το **Pandas** έχει καθιερωθεί εδώ και πάνω από μια δεκαετία ως το προεπιλεγμένο εργαλείο για ανάλυση δεδομένων σε **Python**, χάρη στην ισχυρή λειτουργικότητα και το φιλικό API που προσφέρει στους επιστήμονες δεδομένων [2], [6] . Ωστόσο, η ευρεία υιοθέτηση του **Pandas** έχει αναδείξει και σοβαρούς περιορισμούς του όταν εφαρμόζεται σε μεγάλα σύνολα δεδομένων ή απαιτητικές εφαρμογές. Πράγματι, το **Pandas** σχεδιάστηκε αρχικά για εργασίες σε έναν μόνο υπολογιστή και λειτουργεί **μονονηματικά** (**single-threaded**), κρατώντας όλα τα δεδομένα στη μνήμη RAM καθ' όλη τη διάρκεια μιας επεξεργασίας [2]. Δεν υποστηρίζει ενσωματωμένα κατανεμημένη επεξεργασία σε συστοιχίες υπολογιστών και δεν διαθέτει μηχανισμούς βελτιστοποίησης μνήμης – γεγονός που σημαίνει ότι συχνά αδυνατεί να επεξεργαστεί αποτελεσματικά δεδομένα που υπερβαίνουν τη μνήμη του συστήματος [2], [4].

Αυτοί οι περιορισμοί γίνονται ιδιαίτερα εμφανείς σε εφαρμογές "**big data**", όπου το **Pandas** παρουσιάζει υστέρηση σε ταχύτητα και αξιοποίηση πόρων [7]. Η ανάγκη για αντιμετώπιση αυτών των περιορισμών οδήγησε στην εμφάνιση αρκετών εναλλακτικών βιβλιοθηκών **DataFrame** για **Python**. Βιβλιοθήκες όπως το **Dask** [3] και το **Modin** [9] επιχειρούν να **παραλληλοποιήσουν** τον χειρισμό των **DataFrame** διαμερίζοντας τα δεδομένα και φορτώνοντάς τα σε πολλαπλούς πυρήνες ή μηχανήματα. Άλλες λύσεις, όπως το **Apache Spark** (**PySpark**) [13] και το **Vaex**, υιοθετούν αναβλητική εκτέλεση (**lazy execution**) και στήριξη σε στήλες (**columnar formats**) για να διαχειριστούν μεγαλύτερα σύνολα δεδομένων.

Παράλληλα, εμφανίστηκαν βιβλιοθήκες όπως το RAPIDS **cuDF** που αξιοποιούν επεξεργασία σε GPU για επιτάχυνση **[8]**, και νέες υλοποιήσεις όπως το **DataTable** (εμπνευσμένο από το **R's data.table**) για ταχύτερη διαχείριση μεγάλων πινάκων. Η πληθώρα αυτών των επιλογών, καθεμία με διαφορετικά χαρακτηριστικά (π.χ. παράλληλη εκτέλεση, υποστήριξη GPU, αναβλητική αξιολόγηση), μπορεί να προκαλέσει σύγχυση στους χρήστες ως προς το ποια είναι καταλληλότερη για κάθε περίπτωση [7].

Στο πλαίσιο αυτό, το **Polars** έχει αναδειχθεί πρόσφατα ως μια σύγχρονη εναλλακτική του **Pandas**, επιχειρώντας να προσφέρει δραστικά καλύτερη απόδοση και κλιμάκωση σε έναν υπολογιστή, διατηρώντας παράλληλα ένα εύχρηστο API παρόμοιο με του **Pandas** [1], [10] . Το **Polars** στοχεύει να αντιμετωπίσει τις αναποτελεσματικότητες των υπαρχόντων **DataFrame libraries** παρέχοντας μια νέα υλοποίηση από το μηδέν [1]. Ειδικότερα, έχει αναφερθεί ότι το **Polars** μπορεί να πετύχει επιταχύνσεις 10x ή και μεγαλύτερες σε σύγκριση με το **Pandas** σε διάφορες εργασίες πραγματικής ανάλυσης δεδομένων [1], **[7]**.
Ταυτόχρονα, η βιβλιοθήκη διατηρεί μία γνώριμη στον χρήστη **διεπαφή**, γεγονός που καθιστά σχετικά απλή τη μετάβαση για όσους είναι εξοικειωμένοι με την επεξεργασία δεδομένων σε **Python** **[10]**.

Στις επόμενες ενότητες παρουσιάζεται αναλυτικά η αρχιτεκτονική του **Polars**, οι τεχνικές βελτιστοποίησης που χρησιμοποιεί, καθώς και μια εκτενής σύγκριση με το **Pandas** – τόσο από πλευράς απόδοσης όσο και ευχρηστίας. Επιπλέον, περιλαμβάνονται παραδείγματα κώδικα, μελέτες περίπτωσης σε διαφορετικούς τομείς εφαρμογών, και συζήτηση για τα πλεονεκτήματα (και πιθανά μειονεκτήματα) του **Polars** στην πράξη.

1.2 Τεχνική Αρχιτεκτονική του Polars (Rust, Apache Arrow, Στηλοθέτηση)

Το **Polars** διαφέρει ριζικά εσωτερικά από το **Pandas** ως προς τις τεχνολογικές επιλογές υλοποίησης. Είναι γραμμένο εξ ολοκλήρου στη γλώσσα **Rust**, σε αντίθεση με το **Pandas** που βασίζεται σε **Python** και **C** (μέσω **NumPy**) [2], [6]. Η χρήση της **Rust** επιτρέπει στο **Polars** να αξιοποιεί την ασφαλή

διαχείριση μνήμης και τον αποτελεσματικό έλεγχο χαμηλού επιπέδου που προσφέρει αυτή η γλώσσα, αποφεύγοντας το **overhead** του διερμηνέα της **Python** σε κρίσιμες λειτουργίες [1], [10]. Επιπλέον, η **Rust** προσφέρει εγγενή υποστήριξη για **πολυνηματικότητα** (**threads**) χωρίς το **Global Interpreter Lock** (**GIL**) της **Python**, κάτι που εκμεταλλεύεται πλήρως το **Polars** για να τρέχει πολλαπλές λειτουργίες παράλληλα σε όλους τους πυρήνες **CPU** [10].

Ένα από τα θεμέλια της αρχιτεκτονικής του **Polars** είναι η υιοθέτηση του **στηλοθετημένου** μοντέλου μνήμης **Apache Arrow**. Συγκεκριμένα, το **Polars** αποθηκεύει τα δεδομένα του στη μνήμη ακολουθώντας το **format** του **Arrow** (μέσω της υλοποίησης **arrow2** στη **Rust**) [1], [4]. Αυτό σημαίνει ότι κάθε στήλη δεδομένων είναι αποθηκευμένη σε συνεχόμενες περιοχές μνήμης, με ενιαίο τύπο, βελτιστοποιώντας την **τοπικότητα** μνήμης και επιτρέποντας στα επεξεργαστικά **instructions** (ιδίως τα **SIMD vectorized operations**) να επεξεργάζονται τα στοιχεία πολύ αποδοτικά [1], [10].

Η επιλογή του **Arrow** προσφέρει επίσης **διαλειτουργικότητα** μηδενικής αντιγραφής με άλλα συστήματα – δηλαδή το **Polars** μπορεί να μοιράζεται δεδομένα στην ίδια μορφή με άλλες βιβλιοθήκες (π.χ. **Pandas**, **PyArrow**, **DuckDB**) χωρίς να απαιτείται μετατροπή ή αντιγραφή μνήμης [4]. Αυτό μειώνει δραστικά το **overhead** σε **pipelines** όπου τα δεδομένα περνούν από/προς διαφορετικά εργαλεία [1].

Με βάση τα παραπάνω, το **Polars** επιτυγχάνει εξαιρετικά αποτελεσματική αξιοποίηση πόρων. Η παράλληλη εκτέλεση και οι βελτιστοποιημένοι αλγόριθμοι που εκμεταλλεύονται τη **στηλοθέτηση** και την **τοπικότητα cache**, του επιτρέπουν να υπερκεράσει άλλες λύσεις σε απόδοση [1], [7], [10]. Χαρακτηριστικά, σε ανεξάρτητες δοκιμές τύπου **TPC-H** (**data warehousing benchmark**), το **Polars** έχει δείξει επιτάχυνση άνω του 30**x** σε σχέση με το **Pandas** σε σύνθετα ερωτήματα που προσομοιώνουν πραγματικούς μετασχηματισμούς δεδομένων [7], [10].

Αυτά τα κέρδη προκύπτουν από το συνδυασμό πολλαπλών παραγόντων: τον **πολυνήματικο query engine**, τους αποδοτικούς αλγόριθμους επεξεργασίας ανά στήλη, και την ελαχιστοποίηση αντιγραφών δεδομένων [1], [10].

Ένα ακόμα σχεδιαστικό χαρακτηριστικό του **Polars** είναι ότι δεν χρησιμοποιεί **Index** όπως το **Pandas**. Δηλαδή, δεν υπάρχει ξεχωριστή ετικέτα/δείκτης για τις γραμμές – κάθε γραμμή έχει ως δείκτη απλώς τη θέση της (ακέραιος αύξων αριθμός) [1], [6]. Αυτό απλουστεύει τη συμπεριφορά του **DataFrame** και αποφεύγει μια πηγή πιθανής σύγχυσης του **Pandas** (όπου οι πράξεις μπορούν να διαφοροποιούνται ανάλογα με το **index** ή να απαιτούν ρυθμίσεις όπως **reset_index**) [6]. Στο **Polars**, το **DataFrame** αντιμετωπίζεται πάντα ως ένας καθαρός πίνακας δύο διαστάσεων χωρίς κρυφά **state**, κάτι που κάνει τα αποτελέσματα πιο προβλέψιμα και τα **queries** πιο ρητά [10]. Εσωτερικά, αν χρειάζεται, το **Polars** μπορεί να χρησιμοποιήσει δομές ευρετηρίασης (**index structures**) ως τεχνική βελτιστοποίησης, π.χ. για επιτάχυνση συγκεκριμένων αναζητήσεων, αλλά αυτό είναι διάφανο για τον χρήστη [10].

Συνοψίζοντας, η αρχιτεκτονική του **Polars** συνδυάζει:

- μία μηχανή επεξεργασίας σε χαμηλό επίπεδο υλοποιημένη σε **Rust**,
- το **στηλοθετημένο format Apache Arrow** για αποθήκευση στη μνήμη,
- πλήρη εκμετάλλευση της παράλληλης επεξεργασίας σε **CPU**,
- και έναν σχεδιασμό **API** που στοχεύει στην απλότητα (χωρίς **global index**) [1], [10].

Αυτές οι επιλογές δίνουν στο **Polars** τη δυνατότητα να επεξεργάζεται δεδομένα εξαιρετικά γρήγορα και αποδοτικά. Όπως αναφέρθηκε σε μελέτη, η υιοθέτηση του **Arrow** και οι **cache-efficient** αλγόριθμοι καθιστούν το **Polars** ιδιαίτερα ικανό σε εντατική επεξεργασία, αξιοποιώντας πλήρως τους διαθέσιμους πόρους του μηχανήματος [1].

1.3 Lazy Evaluation και Βελτιστοποιητής Ερωτημάτων του Polars

Ένα από τα πιο σημαντικά πλεονεκτήματα του **Polars** έναντι του παραδοσιακού **Pandas** είναι η υποστήριξη αναβλητικής εκτέλεσης (**lazy evaluation**) και το ενσωματωμένο σύστημα βελτιστοποίησης ερωτημάτων (**query optimizer**) [1], [10]. Ενώ στο **Pandas** κάθε λειτουργία εκτελείται αμέσως μόλις κληθεί (**eager execution**), το **Polars** δίνει τη δυνατότητα να συσσωρεύσουμε λειτουργίες σε ένα σχέδιο εκτέλεσης (**query plan**), το οποίο υλοποιείται μόνο όταν ζητηθεί το τελικό αποτέλεσμα [10]. Στο **Polars**, ο προγραμματιστής μπορεί να εργαστεί είτε σε **eager mode** (άμεσα όπως στο **Pandas**) είτε σε **lazy mode**. Στο **lazy mode**, οι κλήσεις δεν υπολογίζονται αμέσως – αντιθέτως, προστίθενται σε έναν **γράφο** διεργασιών που αναπαριστά το ερώτημα [10].
Μόλις ζητηθεί, π.χ. με μια ενέργεια .**collect**() (συλλογή αποτελέσματος) ή άλλη τελική εντολή, τότε το **Polars** εξετάζει συνολικά το **γράφο** των πράξεων και εφαρμόζει αυτόματες βελτιστοποιήσεις πριν την εκτέλεση [10]. Αυτό του επιτρέπει να βρει τρόπους να επιταχύνει το ερώτημα ή να μειώσει τη χρήση μνήμης, κάτι που γίνεται διαφανώς για τον χρήστη [10].

Ο **βελτιστοποιητής** ερωτημάτων του **Polars** πραγματοποιεί μια σειρά από μετασχηματισμούς στο **query plan**, εμπνευσμένους από τις τεχνικές των σχεσιακών βάσεων δεδομένων [10]. Μερικές από τις βασικές βελτιστοποιήσεις που εφαρμόζονται είναι οι εξής [10]:

- **Προώθηση πλειοτήτων (Predicate pushdown):** Τα φίλτρα (**conditions** τύπου **filter/where**) μετακινούνται όσο το δυνατόν νωρίτερα στο σχέδιο εκτέλεσης – ιδανικά απευθείας στο επίπεδο της φόρτωσης δεδομένων. Έτσι, το σύστημα διαβάζει και επεξεργάζεται μόνο τις γραμμές που ικανοποιούν τα κριτήρια, αντί να φορτώσει ολόκληρο το **dataset** και κατόπιν να φιλτράρει [10].
- **Προώθηση προβολών (Projection pushdown):** Επιλέγονται (προβάλλονται) μόνο οι στήλες που απαιτούνται τελικά από το ερώτημα, ήδη από το στάδιο ανάγνωσης (**scan**) των δεδομένων [10]. Αν, για παράδειγμα, ένα **DataFrame** έχει 50 στήλες αλλά το **query** χρησιμοποιεί μόνο 5 από αυτές, το **Polars** θα φορτώσει μόνο τις 5 σχετικές στήλες. Έτσι εξοικονομεί μνήμη και χρόνο υπολογισμού [10].
- **Slice pushdown:** Σε λειτουργίες όπως π.χ. **df.head**(10) ή γενικά λήψη φέτας (**slice**) από το **DataFrame**, ο **optimizer** φροντίζει να πραγματοποιείται η φόρτωση μόνο του αναγκαίου τμήματος των δεδομένων. Δεν υλοποιείται δηλαδή πρώτα ολόκληρο το αποτέλεσμα για να παρθούν τα πρώτα 10 στοιχεία – αντίθετα, το όριο περνάει μέχρι το **scan** των δεδομένων [10].
- **Κοινή εξαφάνιση υποσχεδίων (Common subplan elimination):** Εάν το **query plan** περιέχει επαναλαμβανόμενα **υποερωτήματα** ή αναγνώσεις του ίδιου αρχείου, αυτά ανιχνεύονται και εκτελούνται μόνο μία φορά, με **caching** των αποτελεσμάτων για επαναχρησιμοποίηση [10].
- **Απλοποίηση εκφράσεων:** Ο **βελτιστοποιητής** απλοποιεί αλγεβρικά τις εκφράσεις (π.χ. **constant folding** – προϋπολογισμός εκφράσεων που έχουν σταθερές τιμές, αντικατάσταση ακριβών πράξεων με ισοδύναμες φθηνότερες κ.λπ.) μέχρι να μη μπορεί να βελτιωθεί περαιτέρω το σχέδιο [10].
- **Βελτιστοποίηση συνενώσεων (Join ordering):** Σε ερωτήματα που περιλαμβάνουν συνενώσεις (**joins**) πολλών πινάκων, ο **optimizer** μπορεί να εκτιμήσει την **καρδιναλότητα** (το μέγεθος) των ενδιάμεσων αποτελεσμάτων και να αποφασίσει τη βέλτιστη σειρά εκτέλεσης των **joins**, ώστε να ελαχιστοποιηθεί η χρήση μνήμης [10].
- **Δυναμική προσαρμογή τύπων (Type coercion):** Ορισμένες φορές χρειάζεται μετατροπή τύπων δεδομένων για να εκτελεστεί μια πράξη (π.χ. σύγκριση μεταξύ ακεραίου και δεκαδικού). Ο **optimizer** προσπαθεί να κάνει τις ελάχιστες απαιτούμενες μετατροπές και με τρόπο που να χρησιμοποιείται η μικρότερη δυνατή εύρος τύπου (π.χ. 32-bit αντί 64-bit) ώστε να βελτιστοποιηθεί και η χρήση μνήμης [10].

- **Εκτίμηση καρδιναλότητας:** Κατά την ομαδοποίηση (**group by**) ή άλλες πράξεις, γίνεται εκτίμηση του πλήθους μοναδικών τιμών/ομάδων ώστε να επιλέγεται ο καλύτερος αλγόριθμος για την περίσταση (π.χ. διαφορετική στρατηγική αν περιμένουμε πολλές ή λίγες ομάδες) [10].

Ο παραπάνω κατάλογος δεν είναι εξαντλητικός, αλλά περιλαμβάνει μερικές από τις κύριες τεχνικές που εφαρμόζει το **Polars** αυτόματα. Ορισμένες από αυτές τις βελτιστοποιήσεις εκτελούνται μία φορά προκαταρκτικά (π.χ. **pushdowns** κατά τη δημιουργία του σχεδίου), ενώ άλλες μπορεί να εφαρμόζονται επαναληπτικά κατά την εκτέλεση μέχρι να μη βελτιώνεται άλλο το αποτέλεσμα [10].

Η αναβλητική εκτέλεση συνδυαστικά με τον **query optimizer** επιτρέπει επίσης στο **Polars** να χειρίζεται μεγαλύτερα από τη μνήμη σύνολα δεδομένων σε ροή (**streaming**) [1], [10]. Αντί να φορτώσει ολόκληρο το **dataset** στη μνήμη, το **Polars** σε **lazy mode** μπορεί να εκτελεί τις πράξεις σε παρτίδες δεδομένων (**batches**), περνώντας τμηματικά τα δεδομένα από τις διάφορες φάσεις επεξεργασίας. Έτσι μειώνει το μέγιστο αποτύπωμα μνήμης και κατανομής CPU κάθε στιγμή, καθιστώντας εφικτή την επεξεργασία ακόμα και **dataset** που υπό άλλες συνθήκες δεν θα χωρούσαν εξ ολοκλήρου στη μνήμη [1].

Όπως περιγράφεται και στη βιβλιογραφία, το **Polars** μπορεί να διαχειριστεί μεγαλύτερα σύνολα από το συνηθισμένο, διότι η **lazy** στρατηγική του "σπάει" την επεξεργασία και ελαφρύνει το φορτίο σε μνήμη και CPU, επιτρέποντας ταχύτερη επεξεργασία ακόμα και για δεδομένα εκτός μνήμης [1].
Συμπερασματικά, το **Polars** αξιοποιεί την **lazy evaluation** για να δώσει στον ενσωματωμένο **βελτιστοποιητή** του μια συνολική εικόνα του ερωτήματος και εφαρμόζει πολλαπλές τεχνικές βελτίωσης [10]. Αυτό έχει ως αποτέλεσμα τα **queries** στο **Polars** να εκτελούνται πολύ πιο αποτελεσματικά σε σύγκριση με μια **one-by-one** εκτέλεση όπως στο **Pandas**. Στην πράξη, οι χρήστες του **Polars** διαπιστώνουν συχνά μεγάλα κέρδη σε ταχύτητα χωρίς χειροκίνητη βελτιστοποίηση, χάρη σε αυτές τις αυτοματοποιημένες διαδικασίες [1], [10].

ΚΕΦΑΛΑΙΟ 2 Σύγκριση Polars με Pandas: Απόδοση, Χρηστικότητα API και Συμβατότητα

2.1 Απόδοση – Ταχύτητα, CPU, μνήμη (Benchmarks)

Η απόδοση είναι ο τομέας όπου το Polars ξεχωρίζει περισσότερο σε σχέση με το Pandas. Πολλές ανεξάρτητες αξιολογήσεις και μελέτες έχουν δείξει ότι το Polars υπερέχει σημαντικά σε ταχύτητα εκτέλεσης σε ένα ευρύ φάσμα εργασιών προετοιμασίας και ανάλυσης δεδομένων [1], . Σε μια πρόσφατη εκτενή συγκριτική μελέτη (EDBT 2025), που αξιολόγησε διάφορες βιβλιοθήκες DataFrame σε πραγματικά workflows, βρέθηκε ότι για μικρότερα datasets το Pandas παραμένει μια καλή επιλογή λόγω του πλούσιου API του, αλλά μόλις τα δεδομένα μεγαλώσουν το Polars αναδεικνύεται ως η προτιμητέα λύση όταν υπάρχει επάρκεια RAM [1].

Χάρη στις βελτιστοποιήσεις και την in-memory εκτέλεση, το Polars ξεπέρασε όλες τις άλλες βιβλιοθήκες σε πολλά σενάρια, παρέχοντας ταχύτερη ολοκλήρωση των pipelines προετοιμασίας δεδομένων [1]. Χαρακτηριστικά, σε εργασίες εξερευνητικής ανάλυσης δεδομένων (EDA), το Polars ήταν έως και 10 φορές πιο γρήγορο από το Pandas [1]. Για παράδειγμα, στον εντοπισμό τιμών που λείπουν (missing values), το Polars σημείωσε τη βέλτιστη επίδοση εκμεταλλευόμενο το metadata που αποθηκεύει το Arrow για τις στήλες, φτάνοντας να είναι μια τάξη μεγέθους ταχύτερο από το Pandas [1].

Ακόμα και σε πιο βαριές διαδικασίες, όπως εντοπισμό ακραίων τιμών (outlier detection) ή περίπλοκες ομαδοποιήσεις, το Polars υπερείχε με σημαντική διαφορά – π.χ. ήταν ~30% ταχύτερο από την επόμενη καλύτερη εναλλακτική βιβλιοθήκη σε ορισμένα benchmarks, πέρα από το πολύ μεγαλύτερο πλεονέκτημα έναντι του Pandas [1].

Ένα ενδιαφέρον εύρημα είναι ότι το Polars αξιοποιεί πολύ καλύτερα τους διαθέσιμους πυρήνες της CPU. Στη μελέτη ενεργειακής αποδοτικότητας (EASE 2024), βρέθηκε ότι το Polars όχι μόνο εκτελείται γρηγορότερα αλλά και καταναλώνει λιγότερη ενέργεια για την ίδια εργασία, ειδικά σε μεγάλα σύνολα δεδομένων [10]. Εφόσον υπάρχει θετική συσχέτιση χρόνου-ενέργειας, η βελτιωμένη ταχύτητα του Polars σημαίνει και μικρότερη ενεργειακή δαπάνη. Συγκεκριμένα, σε συνθετικά workloads μεγάλης κλίμακας, το Polars κατανάλωσε περίπου 8 φορές λιγότερη ενέργεια από το Pandas, ενώ σε ερωτήματα του benchmark TPC-H χρησιμοποίησε περίπου 63% της ενέργειας που χρειαζόταν το Pandas (δηλ. το Pandas απαιτούσε ~1.6× περισσότερη) [10].

Αυτά τα αποτελέσματα υποδεικνύουν ότι το Polars κάνει καλύτερη αξιοποίηση του hardware (πολυπύρηνα CPUs), ολοκληρώνοντας τις εργασίες ταχύτερα και αφήνοντας τον επεξεργαστή να επανέλθει σε idle πιο σύντομα [10]. Σε αντίθεση, το Pandas λόγω μονονηματικότητας συχνά δεν χρησιμοποιεί πλήρως τις δυνατότητες του συστήματος, οδηγώντας σε μεγαλύτερους χρόνους εκτέλεσης και περιττή ενεργειακή κατανάλωση [2].

Όσον αφορά τη χρήση μνήμης, το Polars εμφανίζεται επίσης πιο αποδοτικό σε πολλές περιπτώσεις. Η στηλοθετημένη φύση του (Arrow) και οι lazy βελτιστοποιήσεις (projection pushdown κλπ.) σημαίνουν ότι φορτώνει μόνο τα απαραίτητα δεδομένα στη μνήμη [1]. Αυτό μπορεί να μειώσει το memory footprint σημαντικά όταν δουλεύουμε με ευρεία datasets ή υποσύνολα στηλών. Επιπλέον, μέσω του streaming, το Polars μπορεί να επεξεργάζεται δεδομένα σε chunks χωρίς να διατηρεί ολόκληρο το dataset ταυτόχρονα στη RAM [1].

Από την άλλη πλευρά, το Pandas τείνει να καταλαμβάνει όση μνήμη είναι το μέγεθος του dataset (ή και περισσότερο αν δημιουργούνται αντίγραφα κατά τις πράξεις) και δεν απελευθερώνει μνήμη πριν την ολοκλήρωση του pipeline [6]. Βέβαια, το Polars ως in-memory engine απαιτεί και αυτό να χωρούν τα δεδομένα σε γενικές γραμμές στη μνήμη για βέλτιστη απόδοση. Όπως παρατηρήθηκε, για εξαιρετικά μεγάλα datasets που υπερβαίνουν τη RAM, λύσεις όπως το Spark ή το Vaex ίσως είναι καταλληλότερες (π.χ. το Vaex εκμεταλλεύεται memory mapping για out-of-core υπολογισμούς) [13].

Παρ' όλα αυτά, το **Polars** έχει κάνει βήματα να γεφυρώσει το χάσμα με τον επανασχεδιασμό της **streaming** μηχανής του (βλ. παρακάτω μελλοντική εξέλιξη), ώστε να διατηρήσει υψηλή απόδοση ακόμα και σε πιο ακραία μεγέθη δεδομένων [1].

Συνολικά, σε ένα τυπικό σενάριο όπου τα δεδομένα χωρούν (έστω οριακά) στη μνήμη ενός δυνατού μηχανήματος, το **Polars** θα εκμεταλλευτεί στο έπακρο τους πόρους και θα ολοκληρώσει τις επεξεργασίες αισθητά πιο γρήγορα από το **Pandas** [1]. Έχει αναφερθεί ότι το **Polars** μπορεί να επιτύχει έως 50x υψηλότερο **throughput** σε ορισμένες περιπτώσεις εντατικών εργασιών σε σχέση με το **Pandas** [10].

Η πραγματική διαφορά φυσικά εξαρτάται από τη φύση του **task** (I/O-**bound**, CPU-**bound** κλπ.), το μέγεθος και σχήμα των δεδομένων, και τον διαθέσιμο εξοπλισμό. Ωστόσο, τα στοιχεία συγκλίνουν στο ότι το **Polars** είναι μια "**blazingly fast**" βιβλιοθήκη για **μονομηχανικές** αναλύσεις, ξεπερνώντας ακόμα και λύσεις όπως το **Dask** ή το **Modin** που προσπαθούν να επιταχύνουν το **Pandas** (συχνά με μερική επιτυχία μόνο) [3], [9].

Επομένως, από πλευράς ακαθάριστης ταχύτητας εκτέλεσης, το **Polars** αποτελεί μια από τις κορυφαίες διαθέσιμες επιλογές σήμερα [1] .

2.2 Ευχρηστία και API – Τεκμηρίωση και μετάβαση από Pandas

Παρά τις εσωτερικές διαφορές, το **Polars** έχει σχεδιαστεί με γνώμονα την ευχρηστία και τη γνώριμη εμπειρία για όσους έχουν συνηθίσει το **Pandas** [10]. Το API του **Polars** είναι τύπου **DataFrame** (με μεθόδους για επιλογή στηλών, φιλτράρισμα, ομαδοποίηση, **join** κ.λπ.) και σε μεγάλο βαθμό θυμίζει το **Pandas** ως προς τις λειτουργίες υψηλού επιπέδου που προσφέρει [10]. Για παράδειγμα, βασικές πράξεις όπως **filter**, **groupby**, **agg** (**aggregate**), **join**, **sort** είναι διαθέσιμες και στο **Polars** με παρόμοια σημασιολογία. Αυτό σημαίνει ότι ένας αναλυτής δεδομένων μπορεί σχετικά γρήγορα να προσαρμοστεί στο **Polars**, αφού θα "νιώσει σαν στο σπίτι του" με το API [10].

Μάλιστα, η σχεδίαση των εκφράσεων (**Expressions API**) του **Polars** επιτρέπει να γράφεται κώδικας σαφής και αποδοτικός, διατηρώντας την ευανάγνωστη μορφή των ακολουθιών εντολών που αγαπήθηκε στο **Pandas** [10].

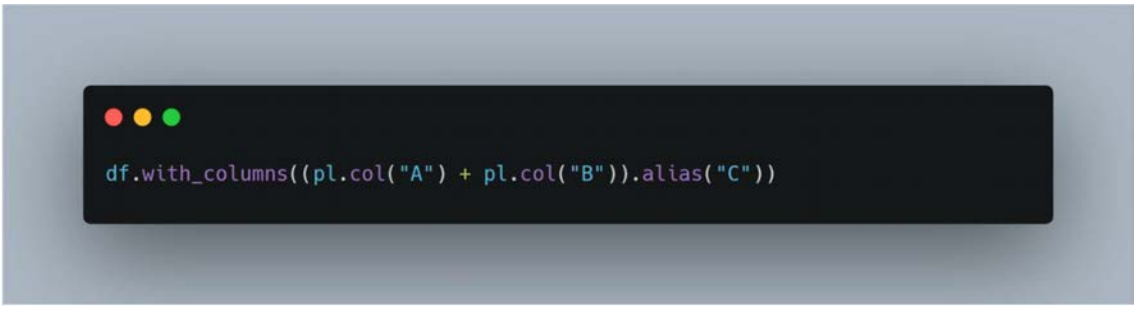
Φυσικά, υπάρχουν κάποιες διαφορές φιλοσοφίας που επηρεάζουν την ευχρηστία. Όπως αναφέρθηκε, το **Polars** δεν έχει **MultiIndex** ή καθόλου **index** για τις γραμμές, κάτι που απλοποιεί μεν τα πράγματα αλλά σημαίνει ότι λειτουργίες που στο **Pandas** στηρίζονταν σε **index** (π.χ. **df.set_index** ή αναφορές με .**loc** σε ετικέτες) δεν υπάρχουν με τον ίδιο τρόπο [10]. Αντί αυτών, στο **Polars** η πρόσβαση είναι πάντα κατά θέση (ή μέσω συνθηκών). Για πολλούς χρήστες, αυτό δεν αποτελεί πρόβλημα — αντιθέτως αφαιρεί πιθανές πηγές σφαλμάτων (π.χ. **mismatch** σε **index** μεταξύ δύο **DataFrames** που ενώνουμε) [10].

Σε περιπτώσεις που χρειάζεται συμπεριφορά **index** (π.χ. χρονική ευρετηρίαση), το **Polars** παρέχει εναλλακτικές μέσω ρητών συναρτήσεων (π.χ. **groupby_dynamic** για χρονικά παράθυρα αντί της **Pandas resample**) [10].

Ένα άλλο στοιχείο είναι η έννοια των **Expressions** στο **Polars**. Ενώ στο **Pandas** οι περισσότερες πράξεις γίνονται μέσω μεθόδων απευθείας στο **DataFrame** ή **Series**, στο **Polars** προωθείται η χρήση ενός **DSL εκφράσεων** που περιγράφει τις μετασχηματίσεις [1]. Για παράδειγμα, για να προσθέσουμε δύο στήλες, στο **Pandas** θα γράφαμε:

Εικόνα 1 ***Παραδοσιακός τρόπος προσθήκης δύο στηλών σε Pandas: Η νέα στήλη "C" δημιουργείται προσθέτοντας απευθείας τις στήλες "A" και "B" με χρήση του συμβατικού API του Pandas.***

ενώ στο Polars:



Εικόνα 2 ***Αντίστοιχη λειτουργία στο Polars: Η στήλη "C" δημιουργείται μέσω του with_columns() και της έκφρασης pl.col("A") + pl.col("B"), η οποία αξιοποιεί το expressions API για καθαρή και βελτιστοποιήσιμη αναπαράσταση του μετασχηματισμού.***

Το pl.col δημιουργεί μια έκφραση-στήλη αντί να επιστρέφει απευθείας τα δεδομένα [1]. Αυτή η προσέγγιση μπορεί αρχικά να φαίνεται πιο σύνθετη, όμως έχει το πλεονέκτημα ότι λειτουργεί απρόσκοπτα σε lazy mode (οι εκφράσεις απλώς προστίθενται στο σχέδιο) και ότι διασφαλίζει ομοιόμορφη σύνταξη ακόμα και για πολύπλοκα queries [1].

Οι πυρήνες του DSL είναι οι Expressions (εκφράσεις στήλης) και τα Contexts (περιβάλλοντα όπου αξιολογούνται, π.χ. select, groupby, κ.ά.) [1].

Το αποτέλεσμα είναι ότι μπορούμε να συνθέσουμε σύνθετες μετασχηματίσεις σε μία γραμμή κώδικα Polars, που παραμένει ευανάγνωστη [1]. Για παράδειγμα:

Εικόνα 3 ***Παράδειγμα lazy pipeline σε Polars, το οποίο φιλτράρει εγγραφές με score > 50, ομαδοποιεί κατά κατηγορία (category), υπολογίζει τον μέσο όρο της στήλης value, ταξινομεί τα αποτελέσματα κατά φθίνουσα τιμή του avg_value και επιστρέφει το τελικό DataFrame μέσω .collect().***

Ο παραπάνω κώδικας εκφράζει με σαφήνεια: "φίλτραρε, ομαδοποίησε και υπολόγισε μέσο όρο, μετά ταξινόμησε" — και όλα αυτά θα εκτελεστούν βέλτιστα χάρη στο **lazy optimizer** [10]. Στο **Pandas**, αντίστοιχος κώδικας είναι εφικτός αλλά το **Polars** δίνει έμφαση στη συναρτησιακή σύνθεση των λειτουργιών, κάτι που πολλοί χρήστες θεωρούν πιο τακτοποιημένο για πολύπλοκα **pipelines** [1], [10].

Όσον αφορά την κάλυψη λειτουργικοτήτων (**API coverage**), το **Pandas** εξακολουθεί να έχει το προβάδισμα ως προς το βάθος και εύρος δυνατοτήτων [6]. Το **Pandas** είναι ώριμο και υποστηρίζει εξαιρετικά ποικιλία από **operations**, από καινοφανείς τύπους δεδομένων (π.χ. **CategoricalDtype**, **SparseArray**) μέχρι εξειδικευμένες μεθόδους (π.χ. **moving window** με ομαδοποίηση, **multi-level indexes** κ.ά.) [6]. Το **Polars** βρίσκεται σε φάση ταχείας ανάπτυξης — ήδη από την έκδοση 1.0 καλύπτει το μεγαλύτερο μέρος των κοινών αναγκών, αλλά ίσως λείπουν κάποιες πολύ εξειδικευμένες λειτουργίες ή υπάρχουν διαφορές στον τρόπο που υλοποιούνται [10].

Για παράδειγμα, το **Pandas** παρέχει τη συνάρτηση **pivot_table** με πληθώρα επιλογών, ενώ το **Polars** έχει τη μέθοδο **pivot** με παρόμοια λειτουργικότητα αλλά ίσως όχι όλα τα περιφερειακά **options** [10]. Παρ' όλα αυτά, σε καίριες λειτουργίες όπως **joins**, **group by aggregations**, διαχείριση ημερομηνιών, μορφοποίηση στηλών κ.λπ., το **Polars** έχει αντίστοιχες δυνατότητες και μάλιστα σε ορισμένα σημεία επεκτείνει/βελτιώνει ιδέες του **Pandas** (π.χ. οι λεγόμενες **lazy UDFs** ή η δυνατότητα για **parallel sorted merges** σε **joins**) [1], [10].

Η τεκμηρίωση (**documentation**) του **Polars** είναι ήδη αρκετά πλούσια και συνεχώς βελτιώνεται [10]. Παρέχεται επίσημος **User Guide** με ενότητες ***Coming from Pandas*** που εξηγούν ακριβώς τις διαφορές και πώς να μετατρέψει κανείς τον τρόπο σκέψης του από **Pandas** σε **Polars** [10]. Υπάρχουν επίσης πολλά παραδείγματα κώδικα στη τεκμηρίωση και στο αποθετήριο του **Polars** στο **GitHub**, καθώς και ένα ενεργό οικοσύστημα άρθρων (**blog posts**, **tutorials**) που γράφονται από την κοινότητα [10].

Επιπλέον, επειδή το **Polars** κερδίζει **δημοφιλία**, η υποστήριξη από εργαλεία εκμάθησης (όπως βιβλία, σεμινάρια, **video courses**) αυξάνεται [10]. Αντίθετα βέβαια, το **Pandas** έχει μια δεκαετία προβάδισμα σε **tutorials**, **StackOverflow** Q&A και γενικά υλικό μάθησης [6]. Ωστόσο, για κάποιον που έχει ήδη εμπειρία στο **Pandas**, το να περάσει στο **Polars** συνήθως απαιτεί μόνο να μάθει μερικές νέες έννοιες (όπως το **lazy** API ή ότι δεν υπάρχει .**loc** αλλά .**filter** με εκφράσεις) [10].

Η ομάδα του **Polars** διατηρεί και έναν "οδικό χάρτη μετάβασης" που συνοψίζει βασικά σημεία (π.χ. ***Polars strict*** σημαίνει ότι δεν επιτρέπει σιωπηρά **type casts** που θα έκανε το **Pandas**) [10].

Σημαντικό στοιχείο ευχρηστίας επίσης είναι η μετάβαση υφιστάμενου κώδικα από **Pandas** σε **Polars**. Εδώ, το **Polars** έχει φροντίσει να παρέχει διευκολύνσεις, όπως μεθόδους για μετατροπή: π.χ. **pl.from_pandas**(**df**) ή **df_polars.to_pandas**() ώστε να μπορεί κανείς να μεταφέρει δεδομένα μεταξύ των δύο βιβλιοθηκών εύκολα [10]. Έτσι, σε ένα μεγάλο έργο, μπορεί να γίνει σταδιακή υιοθέτηση του **Polars** — χρησιμοποιώντας **Polars** όπου χρειάζεται ταχύτητα, αλλά επιστρέφοντας σε **Pandas** δομές όταν κάποια λειτουργία δεν υποστηρίζεται άμεσα ή για να περάσει δεδομένα σε βιβλιοθήκες που περιμένουν **Pandas** αντικείμενα [10].

Η κοινή βάση του **Arrow** κάνει αυτή τη μετατροπή αποδοτική: για παράδειγμα, το **Pandas** 2.0 εισήγαγε πειραματικά την αποθήκευση στηλών ως **Arrow Extension Arrays** αντί για **NumPy**, πράγμα που σημαίνει ότι ένα **Polars DataFrame** θα μπορούσε να μετατραπεί σε **Pandas** χωρίς αντιγραφή (**zero-copy**) χρησιμοποιώντας τον **Arrow** πίνακα στη μνήμη [4], [10]. Ακόμα και χωρίς αυτό, η μέθοδος **to_pandas**() του **Polars** είναι αρκετά γρήγορη για μέτρια **datasets**, επιτρέποντας **integration** με **legacy** κώδικα [10].

Συμπερασματικά, ως προς το API και την ευχρηστία, το **Polars** έχει πετύχει να συνδυάσει παρόμοια ευκολία χρήσης με το **Pandas** για τις συνήθεις εργασίες, εισάγοντας παράλληλα νέα στοιχεία (**Expressions**, **lazy queries**) που απαιτούν μικρή εξοικείωση αλλά ανταμείβουν τον χρήστη με ισχυρότερες δυνατότητες [1], [10]. Η τεκμηρίωση και η κοινότητα υποστήριξης του **Polars** μεγαλώνουν, μειώνοντας συνεχώς την καμπύλη εκμάθησης [10].

Ωστόσο, δεν παύει να είναι μια σχετικά νέα βιβλιοθήκη — ορισμένα "ώριμα" χαρακτηριστικά του **Pandas** μπορεί να λείπουν ή να βρίσκονται υπό ανάπτυξη [6], [10]. Στην πράξη, πολλοί χρήστες αναφέρουν ότι μπορούν να εκτελέσουν τη μεγάλη πλειονότητα των εργασιών τους με **Polars** εύκολα, και μόνο σε ειδικές περιπτώσεις επιστρέφουν προσωρινά στο **Pandas** [10]. Αυτό υποδηλώνει ότι το **Polars** είναι ήδη αρκετά φιλικό και λειτουργικό ώστε να θεωρηθεί βιώσιμη εναλλακτική χωρίς ανυπέρβλητο κόστος μετάβασης.

2.3 Συμβατότητα με το οικοσύστημα Python (NumPy, Scikit-Learn, PyArrow κ.ά.)

Ένα κρίσιμο ζήτημα για κάθε νέα βιβλιοθήκη δεδομένων είναι κατά πόσον μπορεί να ενταχθεί στο υπάρχον οικοσύστημα εργαλείων. Το **Pandas** εδώ και χρόνια συνεργάζεται ομαλά με πλήθος βιβλιοθηκών (π.χ. **NumPy**, **Matplotlib**, **Scikit-Learn**), επομένως είναι σημαντικό το **Polars** να μην αποτελεί "κλειστό κόσμο" [6]. Ευτυχώς, το **Polars** έχει αναπτυχθεί με έμφαση στη **διαλειτουργικότητα** [10].

Πρώτα απ' όλα, μέσω της συμβατότητας **Apache Arrow** που αναφέραμε, το **Polars** μπορεί να ανταλλάσσει δεδομένα αποδοτικά με πολλές άλλες πλατφόρμες [4], [10]. Π.χ., μπορεί να μετατρέψει ένα **DataFrame** σε έναν **Arrow Table** ή **RecordBatch** και αντιστρόφως, πρακτικά χωρίς κόστος [10]. Αυτό σημαίνει ότι μπορούμε να περνάμε δεδομένα από το **Polars** στο **PyArrow** (την **Python** API του **Arrow**) εύκολα, ή να αποθηκεύουμε δεδομένα σε μορφή **Parquet**/**Feather** και να τα διαβάζει το **Polars** και το **Pandas** εναλλάξ χωρίς απώλειες [10].

Στην τεκμηρίωση τονίζεται ότι το **Polars** υποστηρίζει ανάγνωση και εγγραφή σε όλους τους κοινούς **μορφότυπους** (CSV, JSON, **Parquet**, IPC/**Feather**, **Delta Lake** κ.λπ.), διευκολύνοντας έτσι την ενσωμάτωσή του σε υπάρχοντα **data pipelines** [10].

Όσον αφορά τη συνεργασία με βιβλιοθήκες μηχανικής μάθησης, τα νέα είναι ιδιαιτέρως θετικά. Μέχρι πρότινος, πολλές ML βιβλιοθήκες (π.χ. **Scikit-Learn**) δέχονταν **input** κυρίως ως **NumPy array** ή **Pandas DataFrame**. Η κοινότητα του **Polars** όμως εργάστηκε ώστε να γίνει αποδεκτό και το **Polars DataFrame** ως είσοδος [10]. Ήδη από την έκδοση **Scikit-Learn** 1.4, οι μετασχηματιστές (**transformers**) του **Scikit-**

Learn υποστηρίζουν Polars μέσω του API set_output [5]. Δηλαδή, μπορούμε να τροφοδοτήσουμε ένα Polars DataFrame σε ένα pipeline προεπεξεργασίας (π.χ. StandardScaler, OneHotEncoder μέσα σε ColumnTransformer) και να ορίσουμε το output να είναι Polars, επιτρέποντας έτσι ένα καθαρό Polars workflow χωρίς ενδιάμεση μετατροπή σε Pandas [5], [10].

Επίσης, πολλά μοντέλα του Scikit-Learn πλέον αποδέχονται Polars DataFrame απευθείας ως X ή predict input [5], [10]. Στο επίσημο οικοσύστημα του Polars αναφέρεται ξεκάθαρα: "Το Scikit-Learn δέχεται Polars DataFrame ως input/output σε όλους τους transformers και ως input στα μοντέλα" [10].

Παράλληλα, και άλλες βιβλιοθήκες machine learning / data science έχουν προσθέσει υποστήριξη. Τα πακέτα XGBoost και LightGBM (δημοφιλή για gradient boosting σε πίνακες δεδομένων) πλέον δέχονται Polars DataFrames απευθείας ως input [10]. Συγκεκριμένα, το XGBoost ακόμη και σε lazy frame [10].

Επίσης, η εργαλειοθήκη Nixtla για χρονοσειρές (π.χ. η βιβλιοθήκη StatsForecast) έχει ενσωματώσει Polars, επιτρέποντας στις συναρτήσεις πρόβλεψης να παίρνουν Polars DataFrames [10]. Αυτό υποδηλώνει αποδοχή του Polars στην περιοχή των προγνώσεων χρονοσειρών και οικονομικών δεδομένων, που είναι άμεσα σχετική με το πεδίο της χρηματοοικονομικής ανάλυσης.

Για την NumPy, η συνεργασία είναι απλή: οι στήλες του Polars μπορούν εύκολα να μετατραπούν σε NumPy arrays (με τη μέθοδο .to_numpy() σε ένα Series ή .to_numpy() στο DataFrame για matrix) [10]. Στην πραγματικότητα, επειδή το Polars χρησιμοποιεί τους ίδιους τύπους μνήμης με το Arrow, και το Arrow έχει συμβατότητα με NumPy, η μετατροπή είναι ως επί το πλείστον θέμα χρόνου αναφοράς (view) ή αντιγραφής των buffers [4], [10]. Έτσι, αν κάποιος έχει μια ειδική αριθμητική πράξη που υλοποιείται μόνο σε NumPy (ή σε μια βιβλιοθήκη όπως η SciPy που αναμένει NumPy arrays), μπορεί να πάρει τα δεδομένα από Polars με ελάχιστο overhead, να κάνει την πράξη και να τα επαναφέρει στο Polars [10].

Επιπλέον, η βιβλιοθήκη Pandas παραμένει διαθέσιμη ως γέφυρα: όπως προαναφέρθηκε, υπάρχει δυνατότητα to_pandas() και from_pandas() [10]. Αυτό βοηθά στην ενσωμάτωση με εργαλεία που δεν γνωρίζουν ακόμη το Polars. Για παράδειγμα, αν ένα πακέτο visualization δέχεται Pandas DataFrame (π.χ. Seaborn plotting functions), ο χρήστης μπορεί προσωρινά να μετατρέψει το Polars DF σε Pandas για τη σχεδίαση [10].

Ωστόσο, αξίζει να σημειωθεί ότι όλο και περισσότερα εργαλεία αρχίζουν να υποστηρίζουν Polars εγγενώς. Π.χ. η βιβλιοθήκη Plotly έχει πειραματική υποστήριξη [10], ενώ και το Pandas API on Spark (Koalas) έχει αναφορές για Polars integration σε pipelines [10].

Τέλος, η συγγραφή σε αρχεία από Polars είναι συμβατή με ό,τι θα περίμενε κανείς: μπορούμε να γράψουμε CSV, JSON, Parquet, IPC, ακόμα και formats όπως HDF5 μέσω μετατροπής (αν και το HDF5 δεν υποστηρίζεται απευθείας, μπορεί να γίνει μέσω Pandas) [10]. Επίσης, το Polars ενσωματώνεται καλά με βάσεις δεδομένων στηλών όπως το DuckDB. Το DuckDB, μια σχεσιακή βάση για αναλύσεις, μπορεί να λειτουργήσει απευθείας με Polars DataFrames καθώς μοιράζεται το Arrow format, επιτρέποντας στον χρήστη να εκτελέσει SQL queries στα δεδομένα του Polars ή να μεταφέρει αποτελέσματα από το ένα στο άλλο χωρίς κόπο [4], [10].

Συνολικά, η διαλειτουργικότητα του Polars με το οικοσύστημα Python είναι ήδη σε πολύ καλό επίπεδο και βελτιώνεται συνεχώς [1], [10]. Τα κενά που υπήρχαν (όπως η αρχική έλλειψη υποστήριξης στο Scikit-Learn) καλύφθηκαν γρήγορα. Πλέον, το Polars μπορεί να ενταχθεί σε μια ροή εργασίας δίπλα σε NumPy, Pandas, Scikit-Learn, XGBoost, PyArrow, DuckDB κ.ά. χωρίς ο χρήστης να νιώθει αποκλεισμένος [10].

Αυτό είναι καθοριστικό για την υιοθέτησή του στη βιομηχανία, καθώς λίγοι θα ήθελαν να χρησιμοποιήσουν μια λύση που απαιτεί να ξαναγραφούν από την αρχή όλα τα υπόλοιπα τμήματα του pipeline. Με το Polars, ευτυχώς, αυτό δεν αποτελεί εμπόδιο: μπορεί κανείς να το αντικαταστήσει εν μέρει ή πλήρως σε μια υπάρχουσα στοίβα εργαλείων [10].

ΚΕΦΑΛΑΙΟ 3 Παραδείγματα Κώδικα σε Polars

Σε αυτή την ενότητα παρουσιάζονται πρακτικά παραδείγματα κώδικα με το Polars, επιδεικνύοντας πώς πραγματοποιούνται διάφορες συνήθεις εργασίες διαχείρισης δεδομένων. Όλα τα παραδείγματα είναι γραμμένα σε Python, χρησιμοποιώντας την Python API του Polars (η οποία είναι ο πιο διαδεδομένος τρόπος χρήσης του Polars).

3.1 Φόρτωση Δεδομένων (Reading Data)

Το Polars υποστηρίζει ανάγνωση δεδομένων από πολλαπλές πηγές και μορφές. Συχνά, τα δεδομένα μας βρίσκονται σε αρχείο CSV ή Parquet. Παρακάτω φαίνεται πώς διαβάζουμε ένα CSV αρχείο: Python.

Εικόνα 4 ***Φόρτωση αρχείου CSV σε Polars DataFrame με eager εκτέλεση: η συνάρτηση pl.read_csv() διαβάζει άμεσα ολόκληρο το αρχείο στη μνήμη και επιστρέφει ένα πλήρες DataFrame.***

Η συνάρτηση pl.read_csv θα διαβάσει ολόκληρο το αρχείο στη μνήμη και θα επιστρέψει ένα pl.DataFrame. Εάν το αρχείο είναι πολύ μεγάλο και θέλουμε να εκμεταλλευτούμε τη lazy εκτέλεση, μπορούμε να χρησιμοποιήσουμε τη σαρωτή (scanner) scan_csv:

Εικόνα 5 ***Lazy φόρτωση αρχείου CSV με χρήση pl.scan_csv(): δημιουργεί ένα LazyFrame χωρίς άμεση ανάγνωση, επιτρέποντας βελτιστοποιημένη εκτέλεση σε μεταγενέστερο στάδιο.***

Το αντικείμενο lazy_df είναι τύπου LazyFrame – μπορούμε να του εφαρμόσουμε αλυσίδα λειτουργιών lazy (π.χ. filter, select) και μόνο όταν καλέσουμε lazy_df.collect() θα διαβαστεί πραγματικά το αρχείο, εφαρμόζοντας τα φιλτραρίσματα/προβολές κατά την ανάγνωση (χάρη στον optimizer).

Παρομοίως, για άλλα **format**:

- Για **JSON: pl.read_json("file.json")**
- Για **Parquet: pl.read_parquet("file.parquet")**
- Για αρχεία Excel: προς το παρόν μπορούμε μέσω **Pandas** (π.χ. **pl.from_pandas**(**pd.read_excel**(...))), καθώς το **Polars** δεν έχει άμεση συνάρτηση **read_excel**.

3.2 Φιλτράρισμα και Ταξινόμηση (Filtering & Sorting)

Το φιλτράρισμα γραμμών στο **Polars** γίνεται μέσω της μεθόδου **filter** σε ένα **DataFrame/LazyFrame**, χρησιμοποιώντας εκφράσεις που καθορίζουν την συνθήκη. Παράδειγμα: από ένα **DataFrame df** κρατάμε μόνο τις εγγραφές όπου η στήλη "**country**" είναι "**Greece**" και η "**sales**" πάνω από 1000:

Εικόνα 6 ***Φιλτράρισμα γραμμών σε Polars DataFrame: επιλογή εγγραφών όπου η στήλη "country" είναι "Greece" και οι "sales" είναι πάνω από 1000.***

Η έκφραση (**pl.col**("**country**") == "**Greece**") & (**pl.col**("**sales**") > 1000) επιστρέφει μια **boolean** στήλη (μάσκα) που στη συνέχεια χρησιμοποιείται από το **filter** για να επιλέξει τις αντίστοιχες γραμμές. Στο **lazy** πλαίσιο, αυτή η συνθήκη θα προωθηθεί και θα εφαρμοστεί νωρίς (**predicate pushdown**).
Για ταξινόμηση, χρησιμοποιούμε τη μέθοδο **sort**:

Εικόνα 7 ***Ταξινόμηση DataFrame σε φθίνουσα σειρά βάσει της στήλης "sales" χρησιμοποιώντας τη μέθοδο sort με παράμετρο reverse=True.***

Αυτό θα επιστρέψει νέο **DataFrame** ταξινομημένο. Μπορούμε να ταξινομήσουμε και κατά πολλαπλά κλειδιά δίνοντας λίστα στηλών π.χ. **df**.**sort**([**"country"**, **"sales"**], **reverse**=[**False**, **True**]) για αύξουσα ανά χώρα και φθίνουσα ανά πωλήσεις.

3.3 Συγχνώνευση (Join) Πινάκων

Η συνένωση δύο δεδομένων (ανάλογο του **SQL JOIN**) στο **Polars** γίνεται με την μέθοδο **join** επί ενός **DataFrame**, περνώντας το δεύτερο **DataFrame** και τις παραμέτρους **join**. Παράδειγμα, ας υποθέσουμε ότι έχουμε δύο πίνακες: **df_customers** (με πεδίο "**cust_id**") και **df_orders** (με "**cust_id**" επίσης). **Θέλουμε να ενώσουμε τις παραγγελίες με τα στοιχεία πελάτη:**

Εικόνα 8 ***Παράδειγμα συνένωσης (inner join) δύο DataFrames στο Polars: ένωση των df_orders και df_customers βάσει του κοινού πεδίου "cust_id" για ενσωμάτωση στοιχείων παραγγελιών με δεδομένα πελατών.***

Αυτό παράγει ένα νέο **DataFrame** που περιέχει τις στήλες και από τους δύο πίνακες, ενωμένες εσωτερικά με βάση την τιμή του "**cust_id**". Το **Polars** υποστηρίζει τις κλασικές επιλογές **how**: "**inner**", "**left**", "**outer**", "**semi**", "**anti**" **joins**. Επίσης, αν τα πεδία έχουν διαφορετικό όνομα, μπορούμε να χρησιμοποιήσουμε **left_on** και **right_on** αντί on.

Η απόδοση των **join** στο **Polars** είναι πολύ υψηλή – γίνεται χρήση **πολυνηματικών** αλγορίθμων κατακερματισμού ή ταξινόμησης, και ο **optimizer** μπορεί να αλλάξει τη σειρά πολλαπλών **joins** για βέλτιστη χρήση πόρων.

3.4 Ομαδοποίηση (Group By)

Η ομαδοποίηση στο **Polars** θυμίζει **Pandas**, με κάποιες συντακτικές διαφορές. Για να ομαδοποιήσουμε ένα **DataFrame df** κατά μια στήλη και να υπολογίσουμε συναθροίσεις, κάνουμε:

Εικόνα 9 ***Ομαδοποίηση δεδομένων κατά κατηγορία με χρήση της μεθόδου groupby στο Polars και εκτέλεση πολλαπλών συναθροιστικών συναρτήσεων (sum, mean, count) ανά ομάδα.***

Εδώ, η μέθοδος groupby("category") ορίζει το πεδίο ομαδοποίησης, και ακολουθεί η agg([...]) που παίρνει μια λίστα από expressions – κάθε μία υπολογίζει και μία στήλη στο αποτέλεσμα. Χρησιμοποιούμε συναρτήσεις από το namespace pl, όπως pl.sum(), pl.mean(), οι οποίες θα εφαρμοστούν στην κάθε ομάδα. Το αποτέλεσμα result_df θα έχει τρεις στήλες: total_sales, avg_sales, count (συν φυσικά τη στήλη "category" ως δείκτη ομάδας).

Μπορούμε να ομαδοποιήσουμε και κατά πολλαπλές στήλες (δίνοντας λίστα στο groupby), καθώς και να εκτελέσουμε σύνθετες εκφράσεις μέσα στο agg. Π.χ.:

Εικόνα 10 ***Σύνθετη ομαδοποίηση κατά πολλαπλά πεδία (region, product) με υπολογισμό κέρδους ως διαφορά revenue - cost, μέσω σύνθετης έκφρασης εντός της agg.***

Η παραπάνω εντολή, για κάθε συνδυασμό region και product, θα αθροίσει το (revenue - cost) ως συνολικό κέρδος.

Το Polars υποστηρίζει επίσης αγκαθωτές συνόψεις (rolling aggregates) και παραθύρων (window functions) σε συνδυασμό με το grouping ή και χωρίς grouping (βλ. επόμενη ενότητα).

3.5 Pivoting (Ανασχηματισμός Πίνακα)

Η λειτουργία pivot (ανασχηματισμός από "long" μορφή σε "wide" μορφή) είναι διαθέσιμη μέσω της μεθόδου pivot. Για παράδειγμα, αν έχουμε δεδομένα πωλήσεων με στήλες ["year", "product", "sales"] και θέλουμε να δημιουργήσουμε έναν πίνακα με index το year και στήλες προϊόντα, όπου οι τιμές είναι οι πωλήσεις:

Εικόνα 11 ***Χρήση της μεθόδου pivot στο Polars για τη μετατροπή δεδομένων από long σε wide μορφή, συγκεντρώνοντας τις πωλήσεις (sales) ανά συνδυασμό έτους (year) και προϊόντος (product).***

Αυτό θα συγκεντρώσει (sum) τις πωλήσεις ανά έτος για κάθε προϊόν, παράγοντας μια ευρεία μορφή όπου κάθε προϊόν είναι ξεχωριστή στήλη. Το αποτέλεσμα αντιστοιχεί στην **pandas pivot_table** με **aggfunc** sum.

Το **Polars** υποστηρίζει διάφορες συναρτήσεις στην **aggregate_fn** (π.χ. "**mean**", "**first**", "**max**", κτλ.). Αν τα δεδομένα είναι ήδη μοναδικά ανά συνδυασμό **index-columns**, μπορούμε να βάλουμε **aggregate_fn**=**None** ή να χρησιμοποιήσουμε την **default** συμπεριφορά.

Υπάρχει επίσης η αντίστροφη πράξη, το **melt** (ή **unpivot**), μέσω της μεθόδου **melt**(), που μετατρέπει από **wide** σε **long** μορφή.

3.6 Κινητά Παράθυρα (Rolling Windows) και Πολύπλοκες Εκφράσεις

Το **Polars** παρέχει εγγενή υποστήριξη για υπολογισμούς σε κινητά παράθυρα (**rolling window**), κάτι χρήσιμο σε **χρονοσειρές** και χρηματοοικονομικές εφαρμογές. Για παράδειγμα, για έναν πίνακα με ημερήσιες τιμές μετοχής, μπορούμε να υπολογίσουμε τον κυλιόμενο μέσο όρο 7 ημερών της τιμής κλεισίματος:

Εικόνα 12 ***Υπολογισμός κυλιόμενου μέσου όρου 7 ημερών (rolling_mean) για τη στήλη close σε χρονοσειρά τιμών μετοχών, μετά από ταξινόμηση κατά ημερομηνία.***

Η έκφραση **rolling_mean**(**window_size**=7) υπολογίζει τον μέσο όρο σε παράθυρο 7 γραμμών (εδώ 7 ημερών αφού είναι ταξινομημένο ανά ημέρα). Μπορούμε να ορίσουμε και **by** για **grouping** (π.χ. αν είχαμε πολλές μετοχές στο ίδιο DF, θα κάναμε **rolling** ανά **ticker**) ή **min_periods** για ελάχιστες απαιτούμενες τιμές. Υπάρχουν αντίστοιχες συναρτήσεις **rolling_sum**, **rolling_min**, κ.ά.

Για πιο σύνθετες λειτουργίες, το Polars επιτρέπει ***συμφραζόμενες*** *εκφράσεις (**window functions**)* με τη μέθοδο over. Π.χ. μπορούμε να υπολογίσουμε το ποσοστό συνεισφοράς μιας τιμής ως προς το σύνολο της ομάδας της:

Εικόνα 13 ***Υπολογισμός του ποσοστού συμμετοχής κάθε γραμμής στις συνολικές πωλήσεις ανά περιοχή (region), με χρήση window expression στο Polars.***

Η παραπάνω έκφραση δημιουργεί μια νέα στήλη "pct_of_region" που, για κάθε γραμμή, παίρνει το sales της γραμμής, διαιρεί με το άθροισμα των sales στην ίδια region (η .over("region") υποδηλώνει παράθυρο πάνω από τις γραμμές που έχουν το ίδιο region) και πολλαπλασιάζει επί 100. Ουσιαστικά, αυτή είναι μια window function που στο Pandas θα απαιτούσε μια πιο περίπλοκη λογική (π.χ. groupby και transform).

Το Polars διευκολύνει πολύ τις **πολύπλοκες εκφράσεις**. Μπορούμε να συνδυάσουμε αριθμητικές πράξεις, συναρτήσεις, όρους if-else (μέσω της pl.when().then().otherwise()), και να φτιάξουμε στήλες που προκύπτουν από πολλαπλές άλλες. Για παράδειγμα:

Εικόνα 14 ***Σύνθετες εκφράσεις στο Polars: χρήση συνθηκών when-then-otherwise για την κατηγοριοποίηση απόδοσης και υπολογισμός ποσοστού περιθωρίου κέρδους (profit margin) σε μία ενιαία εντολή.***

Εδώ δημιουργούμε δύο νέες στήλες: η "performance" είναι ένα αποτέλεσμα λογικής που συγκρίνει sales με target και δίνει κείμενο, ενώ η "profit_margin_pct" είναι ένα αριθμητικό ποσοστό περιθωρίου κέρδους. Οι εκφράσεις μπορούν να είναι τόσο περίπλοκες όσο χρειάζεται, και το Polars θα τις χειριστεί στο βέλτιστο δυνατό (συμπεριλαμβανομένης πιθανής vectorization ή ακόμα και σύνδεσης με native code για βαριές λειτουργίες).

Συνοψίζοντας τα παραδείγματα, παρατηρούμε ότι το Polars παρέχει ένα **πλούσιο ρεπερτόριο** μεθόδων για τη διαχείριση δεδομένων, συχνά με σύνταξη παρόμοια με του Pandas, αλλά αξιοποιώντας τις **εκφράσεις** για ευελιξία και απόδοση. Ο κώδικας ενίοτε φαίνεται ελαφρώς πιο "λειτουργικός" (functional) σε σχέση με το Pandas, όμως αυτό είναι συνειδητή επιλογή για να επιτυγχάνεται η

δυνατότητα **lazy** εκτέλεσης και βελτιστοποίησης. Οι περισσότεροι χρήστες, μόλις εξοικειωθούν με τις εκφράσεις του **Polars**, μπορούν να γράψουν **καθαρό και συνοπτικό κώδικα** για ιδιαίτερα σύνθετες μετασχηματίσεις δεδομένων, διατηρώντας υψηλή απόδοση.

ΚΕΦΑΛΑΙΟ 4 Σύγκριση με Άλλες Βιβλιοθήκες

Εκτός από το **Pandas** και το **Polars**, υπάρχουν και άλλες βιβλιοθήκες που προσπαθούν να δώσουν λύσεις στους περιορισμούς του **Pandas**. Σε αυτή την ενότητα, συγκρίνουμε συνοπτικά το **Polars** με μερικές από τις πιο γνωστές: το **Dask** (και το συναφές **Modin**), το **cuDF**, καθώς και άλλες όπως το **PySpark (Spark SQL)**, το **Vaex** και το **DataTable**.

Dask DataFrame: Το **Dask** είναι μια βιβλιοθήκη για παράλληλη επεξεργασία που επεκτείνει το **Pandas** σε πολλαπλούς πυρήνες ή κόμβους [7]. Βασικά, "σπάει" ένα μεγάλο **DataFrame Pandas** σε πολλά μικρότερα και τα επεξεργάζεται σε παράλληλα **tasks**, συντονίζοντας τα αποτελέσματα [7]. Το **Modin** είναι ένα **layer** που επιτρέπει στο ίδιο **Pandas** API να τρέχει είτε πάνω σε **Dask** είτε πάνω σε **Ray**, κρύβοντας την πολυπλοκότητα [7]. Πλεονέκτημα του **Dask** είναι ότι οι χρήστες μπορούν να αξιοποιήσουν κώδικα **Pandas** με ελάχιστες τροποποιήσεις για μεγαλύτερα δεδομένα ή σε **cluster**. Ωστόσο, στην πράξη έχει παρατηρηθεί ότι το **Dask DataFrame** δεν κλιμακώνεται γραμμικά και έχει σημαντικό **overhead**, ειδικά σε **single-machine** περιβάλλον [1]. Στη σύγκριση του EDBT, οι συγγραφείς απέκλεισαν το **Dask** στο **single-machine benchmark** λόγω κακής επίδοσης (και σημείωσαν πως το **Dask** καλύπτει ~55% του API του **Pandas** μόνο) [1]. Το **Modin** με **backend Dask** ή **Ray** καλύπτει περισσότερο API (~90%) [1], αλλά και πάλι η απόδοσή του εξαρτάται από το σενάριο.

Σε γενικές γραμμές, το **Polars** τείνει να υπερέχει του **Dask/Modin** σε ωμή ταχύτητα σε έναν υπολογιστή, επειδή δεν έχει το **overhead** του **task scheduling** και είναι γραμμένο σε αποδοτική χαμηλού επιπέδου γλώσσα [10]. Από την άλλη, το **Dask** μπορεί να κλιμακωθεί σε **cluster** (αν και εκεί ίσως καλύτερη λύση είναι το **Spark**) [1].

Apache Spark (PySpark): Το **Spark** είναι μια ώριμη πλατφόρμα μεγάλων δεδομένων [1]. Μέσω της **Python** API (**PySpark**) μπορεί κανείς να χρησιμοποιήσει **DataFrames** (**Spark** SQL) με **lazy** εκτέλεση και βελτιστοποίηση (**Catalyst optimizer**) [1]. Το **Spark** προορίζεται κυρίως για **clusters** και πολύ μεγάλα **data** (εκτός μνήμης). Σε ένα μηχάνημα, τυπικά το **Spark** είναι βραδύτερο λόγω **overhead** JVM κ.λπ., εκτός αν τα δεδομένα είναι τεράστια [1]. Στη μελέτη EDBT, βρέθηκε ότι για πάρα πολύ μεγάλα **datasets** που δεν χωρούν ούτε σε RAM ούτε σε GPU, το **Spark** ήταν η λύση που απέδωσε (καθώς μπορούσε να αξιοποιήσει δίσκο και πολλαπλά νήματα) [1]. Το **Polars** ωστόσο, όσο τα δεδομένα χωρούν σε RAM, ήταν ταχύτερο [1]. Το **Spark** δεν έχει την ευελιξία εντός **Python** περιβάλλοντος που έχει το **Polars** — για μικρομεσαίες αναλύσεις, το **Polars** είναι πολύ πιο "ανάλαφρο" στη χρήση [10].

cuDF (RAPIDS): Το **cuDF** είναι η προσπάθεια της NVIDIA να επιταχύνει το οικοσύστημα **pandas** χρησιμοποιώντας GPU [8]. Είναι γραμμένο σε C++/CUDA και παρέχει ένα API παρόμοιο με το **Pandas**, αλλά οι πράξεις εκτελούνται στην κάρτα γραφικών [8]. Το **cuDF** μπορεί να δώσει τεράστια επιτάχυνση σε κατάλληλα προβλήματα (ιδίως μαθηματικές πράξεις σε πολύ μεγάλους πίνακες), εφόσον τα δεδομένα χωρούν στη μνήμη της GPU [8]. Σύμφωνα με τη βιβλιογραφία, όπου υπάρχει διαθέσιμη GPU, το **cuDF** συχνά ήταν κορυφαίο σε απόδοση, ξεπερνώντας και το **Polars** σε κάποιες βαριές αριθμητικές εργασίες [1]. Ωστόσο, το **cuDF** έχει και αυτό περιορισμούς — δεν διαθέτει δικό του **optimizer** για **queries** (εκτελεί ό,τι του ζητηθεί αυτούσιο) [1], και φυσικά περιορίζεται από τη VRAM [8].

Συγκριτικά, το **Polars** σε CPU πολλές φορές πλησιάζει ή ξεπερνά **cuDF** σε κανονικό **hardware**, ειδικά σε σενάρια όπου οι πράξεις δεν είναι **massively parallel** [1]. Επιπλέον, το **Polars** ήδη πειραματίζεται με ενσωμάτωση του **cuDF** ως **engine** [10]. Στο **Polars** 1.0 **roadmap**, προβλέπεται υποστήριξη GPU **execution** όπου αυτό ωφελεί, συνδυάζοντας τον **optimizer** του **Polars** με την GPU επιτάχυνση [10].

Vaex: Το **Vaex** είναι μια βιβλιοθήκη που στοχεύει σε πολύ μεγάλα **datasets** με τεχνικές **memory mapping** και **streaming** [1]. Χρησιμοποιεί **NumPy arrays** κατά στήλη, με ιδέα "**virtual columns**" όπου κάποιες πράξεις υπολογίζονται τεμπέλικα όταν χρειαστεί [1]. Το **Vaex** μπορεί να χειριστεί **dataset** μεγαλύτερα από τη μνήμη, φορτώνοντάς τα τμηματικά από αρχείο (HDF5 κυρίως) [1]. Σε **benchmarks**, το **Vaex** ήταν εξαιρετικά καλό σε λειτουργίες όπως το **string search** [1]. Ωστόσο, υστερεί σε άλλες — π.χ. δεν υποστηρίζει όλα τα είδη **merges** ή σύνθετων ομαδοποιήσεων [1]. Συγκριτικά, το **Polars**

μοιράζεται την ικανότητα **out-of-core**, αλλά συνήθως το ξεπερνά σε πολυπλοκότερες εργασίες [1], [10].

DataTable: Πρόκειται για μια βιβλιοθήκη της H2O.ai, γραμμένη σε C++, εμπνευσμένη από το **data.table** της R [1]. Προσφέρει ένα **Pandas-like** API στο **Python** και είναι αρκετά γρήγορη σε αρκετές λειτουργίες [1]. Σε κάποιες δοκιμές βρέθηκε ότι η **DataTable** ήταν η ταχύτερη σε συγκεκριμένη λειτουργία (π.χ. εντοπισμός **missing values**) [1]. Ωστόσο, γενικά έχει πιο περιορισμένο εύρος λειτουργικότητας [1]. Δεν υποστηρίζει π.χ. **Parquet format** εισόδου [1], και δεν έχει ενεργή ανάπτυξη όσο το **Polars** [1], [10].

Το **Polars** καταφέρνει να ισορροπεί ανάμεσα σε πολλά από τα προτερήματα των παραπάνω εργαλείων: είναι **πολυνηματικό** (όπως **Dask**/**Modin**, **Spark**), είναι **αναβλητικό** με **optimizer** (όπως **Spark**, **Vaex**), είναι **στηλοθετημένο** και φιλικό στη μνήμη (όπως **Vaex**, **DataTable**), και σύντομα θα υποστηρίζει και **GPU** (όπως **cuDF**) [1], [10]. Αυτό το καθιστά μια γενικής χρήσης γρήγορη λύση για **DataFrames** σε έναν κόμβο.

Βέβαια, για κατανεμημένη επεξεργασία σε **cluster** εξακολουθεί να χρειάζεται **Spark** ή **Dask**, αλλά ήδη το **Polars Cloud** υπόσχεται κλιμάκωση του **Polars** πέρα από το ένα μηχάνημα [10].

ΚΕΦΑΛΑΙΟ 5 Μελέτες Περίπτωσης (Case Studies) ανά Τομέα Εφαρμογής

Σε αυτή την ενότητα εξετάζουμε πώς το **Polars** έχει αρχίσει να χρησιμοποιείται σε διάφορους τομείς, υπογραμμίζοντας τα οφέλη που προσφέρει σε κάθε περίπτωση. Θα αναφερθούμε ενδεικτικά σε τρεις τομείς: **Βιοπληροφορική**, **Ανάλυση Web/Log δεδομένων**, και **Χρηματοοικονομική ανάλυση**.

5.1 Βιοπληροφορική και Γενωμικά Δεδομένα

Ο τομέας της **βιοπληροφορικής** συχνά περιλαμβάνει τεράστιους πίνακες δεδομένων — π.χ. **γονιδιωματικές** ακολουθίες, δεδομένα από ευρείες μελέτες συσχέτισης (**GWAS**), ή αποτελέσματα πειραμάτων **αλληλούχισης**. Παραδοσιακά, γλώσσες όπως η **R** με το **data.table** ή η **Python** με **Pandas** χρησιμοποιούνται για επεξεργασία, αλλά τα μεγέθη δεδομένων (εκατομμύρια καταγραφές DNA κλπ.) καθιστούν αυτές τις μεθόδους οριακές [6].

Το **Polars**, με την απόδοση και το **columnar format** του, είναι ιδανικό για τέτοια σενάρια [10]. Πράγματι, το 2025 προτάθηκε μια νέα βιβλιοθήκη ειδικά για **γενωμικά** δεδομένα βασισμένη στο **Polars**: το **polars-bio** [12]. Το **polars-bio** χρησιμοποιεί το **Polars** σε συνδυασμό με το **Apache DataFusion** (μια μηχανή SQL σε **Rust**) για να παρέχει ένα **DataFrame** API ειδικά για **βιοπληροφορικές** πράξεις [12].

Σύμφωνα με τους δημιουργούς του, το **polars-bio** επιτρέπει ταχύτατες και επεκτάσιμες **out**-of-**core** λειτουργίες σε μεγάλα σύνολα **γονιδιωματικών διαστημάτων** (**genomic interval datasets**) [12]. Σε προδημοσίευση στο **bioRxiv** περιγράφεται ότι το **polars-bio** μπορεί να χειριστεί μεγάλα αρχεία **VCF/BED** με αποτελεσματικότητα, προσφέροντας γνωστές **βιοπληροφορικές** λειτουργίες (π.χ. εύρεση επικαλύψεων γενετικών περιοχών, συνένωση δεδομένων από διαφορετικές πηγές) με ταχύτητα κλάσεων μεγέθους μεγαλύτερη από παραδοσιακές μεθόδους [12].

Η επιλογή του **Polars** σε αυτή την περίπτωση έγινε ακριβώς λόγω των πλεονεκτημάτων του: με τον συνδυασμό **Rust** και **Arrow** παρέχει το απαιτούμενο **performance** για υπολογισμούς πάνω σε δισεκατομμύρια εγγραφές, ενώ το API υψηλού επιπέδου επιτρέπει στους επιστήμονες να γράφουν κώδικα σε **Python** αντί να χρειάζεται να γράψουν C++ ή να χρησιμοποιούν περίπλοκα εργαλεία μεγάλων δεδομένων [12].

Για παράδειγμα, με το **polars-bio** μπορεί κάποιος να διαβάσει δύο μεγάλα σύνολα **γενωμικών** διαστημάτων ως **Polars DataFrames** και να βρει τις επικαλύψεις τους (δηλαδή πού ευθυγραμμίζονται περιοχές DNA από δύο πηγές), χρησιμοποιώντας λειτουργίες **join/merge** σε κλίμακες που το **Pandas** δεν θα μπορούσε καν να φορτώσει στη μνήμη [12].

Ακόμη και εκτός του **polars-bio**, απλοί χρήστες έχουν αναφέρει ότι το **Polars** βοηθά σε **βιοπληροφορικές** εργασίες όπως φιλτράρισμα μεγάλων πινάκων αλληλουχιών ή συγχώνευση **datasets** με πληροφορίες μεταλλάξεων, μειώνοντας τον χρόνο ανάλυσης από ώρες σε λεπτά [12].

Η ικανότητα του **Polars** να κάνει **streaming** (μέσω **lazy**) σημαίνει ότι μπορούν να επεξεργαστούν αρχεία μεγέθους π.χ. 50 GB με κατανεμημένους υπολογισμούς και να κρατηθούν μόνο τα σημαντικά δεδομένα, χωρίς να εξαντληθεί η μνήμη [10], [12].

Στον χώρο της **βιοπληροφορικής** το **Polars** έρχεται ως φυσική εξέλιξη: παρέχει **enterprise-grade** ταχύτητα σε ακαδημαϊκά και ερευνητικά δεδομένα, διατηρώντας απλή την υλοποίηση πειραμάτων δεδομένων [10], [12]. Αυτό επιβεβαιώνεται από το γεγονός ότι πλέον προτείνεται ως βάση για νέα εργαλεία (όπως το **polars-bio**) που δημοσιεύονται στη βιβλιογραφία [12].

5.2 Ανάλυση Web Analytics και Δεδομένων Log

Οι εταιρείες διαδικτύου και οι πλατφόρμες συσσωρεύουν τεράστια ποσά **δεδομένων καταγραφής** (**logs**) — από αρχεία **server** και **clickstreams** χρηστών έως αρχεία συμβάντων εφαρμογών. Η ανάλυση αυτών των **logs** (**web analytics**) είναι κρίσιμη για κατανόηση συμπεριφοράς χρηστών, εντοπισμό προβλημάτων, **marketing analytics** κ.ά. [1].

Παραδοσιακά, εργαλεία όπως το **Elasticsearch** ή **custom scripts** με **Pandas** χρησιμοποιούνται για ad-hoc ανάλυση **logs** [6]. Ωστόσο, με την αύξηση του όγκου, το **Pandas** γίνεται αργό και αναξιόπιστο σε **log data** πολλών **gigabytes** [1]. Το **Polars** αποδεικνύεται ιδιαίτερα χρήσιμο σε αυτό το πεδίο, διότι μπορεί να επεξεργαστεί αρχεία **log** με εκατομμύρια εγγραφές πολύ ταχύτερα από το **Pandas** [10].

Μια χαρακτηριστική περίπτωση είναι το εργαλείο **LogLead** (2023), το οποίο αναπτύχθηκε για φόρτωση και ανάλυση αρχείων **log** και επιλέχθηκε να βασιστεί στο **Polars** ως τον κύριο **data engine** [9]. Οι δημιουργοί του **LogLead** σημειώνουν ότι το **Polars** διαφημίζεται (και αποδεικνύεται) ως "**blazingly fast**" για επεξεργασία **dataframes**, υλοποιημένο σε **Rust** με **Arrow** — χαρακτηριστικά που θεωρήθηκαν ιδανικά για τη **στοχοθεσία** τους [9].

Με το **Polars**, το **LogLead** κατάφερε να επιτύχει πάνω από 10 φορές ταχύτερη φόρτωση **raw log** αρχείων σε σχέση με προηγούμενα εργαλεία [9]. Επιπλέον, στο σχετικό **paper** αναφέρεται ότι αν και δεν υπήρχαν (έως τότε) ακαδημαϊκές συγκρίσεις **Pandas**–**Polars**, οι αναφορές στη **grey literature** υποδηλώνουν πως το **Polars** "συντρίβει" το **Pandas** σε ταχύτητα — κάτι που οι δικές τους μετρήσεις ήθελαν να εκμεταλλευτούν [9].

Στην πράξη, τι σημαίνει αυτό για **web analytics**; Σημαίνει ότι ένας **data engineer** μπορεί να διαβάσει ένα αρχείο **log** 5 GB, να κάνει **parsing** των γραμμών του (π.χ. εξαγωγή πεδίων από κείμενο) και να εκτελέσει **aggregations** (π.χ. πόσα **hits** ανά **endpoint** ή μέσος χρόνος απόκρισης ανά ώρα) πολύ γρηγορότερα [9]. Εκεί που ένα **Pandas script** μπορεί να έτρεχε για 30 λεπτά και να εξαντλούσε τη μνήμη, ένα **Polars script** μπορεί να ολοκληρώνει σε λίγα λεπτά και με χαμηλότερη χρήση μνήμης, χάρη στην **streaming** επεξεργασία [10].

Επίσης, στον χώρο του **Digital Analytics**, όπου συλλέγονται γεγονότα (**events**) από ιστοσελίδες και **mobile apps**, το **Polars** μπορεί να βοηθήσει στην **απο-σειριοποίηση** (**deserialization**) δεδομένων **JSON/CSV** σε **scale**, φιλτράροντας τα **event logs** με συνθήκες (π.χ. χρήστες από συγκεκριμένη χώρα που έκαναν κλικ σε συγκεκριμένο **button**) με μεγάλη ταχύτητα [10].

Η συμβατότητα με **Arrow** σημαίνει επίσης ότι αν τα δεδομένα αποθηκεύονται σε στήλες σε **cloud storage** (π.χ. αρχεία **Parquet**), το **Polars** μπορεί να τα επεξεργαστεί **in-place** [4], [10].

Ένα ακόμα παράδειγμα είναι η ανάλυση αρχείων **συμβάντων δικτύου** ή **ασφάλειας** (π.χ. **firewall logs**, **intrusion detection logs**). Αυτά είναι συχνά μεγάλα CSV που πρέπει να αναλυθούν για ανίχνευση ανωμαλιών. Το **Polars** έχει ήδη χρησιμοποιηθεί σε δοκιμές για τέτοιου είδους **forensic** ανάλυση, παρέχοντας δυνατότητα να φορτωθούν εκατομμύρια **events** και να εκτελεστούν πολύπλοκα **queries** (όπως "εύρεση ακολουθιών αιτημάτων που ταιριάζουν σε μοτίβο") που θα ήταν απαγορευτικά αργά σε **Pandas** [9], [10].

Συμπερασματικά, στον χώρο των **web/log analytics**, το **Polars** προσφέρει τη δυνατότητα για **διαδραστικές αναλύσεις μεγάλων logs** σε έναν προγραμματιστή: **tasks** που πριν ίσως απαιτούσαν **Hadoop/Spark jobs** μπορούν τώρα να δοκιμαστούν τοπικά με **Polars** [10]. Αυτό γεφυρώνει το χάσμα μεταξύ μικρών και μεγάλων δεδομένων, επιτρέποντας πιο **agile** ανάλυση από τους μηχανικούς. Δεν είναι σύμπτωση που εργαλεία σαν το **LogLead** το επέλεξαν ως βάση [9], ούτε ότι ακόμα και σε **Reddit / data engineering** κοινότητες συζητιέται πλέον ευρέως η υπεροχή του **Polars** έναντι του **Pandas** σε **log processing** [10].

5.3 Χρηματοοικονομική Ανάλυση και Χρονοσειρές

Στον χρηματοοικονομικό κλάδο, οι αναλυτές εργάζονται συχνά με **χρονοσειρές μεγάλου μήκους** (ιστορικά δεδομένα μετοχών, τιμές συναλλάγματος ανά δευτερόλεπτο, τιμές **derivatives** κ.λπ.), καθώς και με πολύπλοκα **datasets** (π.χ. σύνθεση χαρτοφυλακίων, συναλλακτική δραστηριότητα) [1]. Το **Pandas** χρησιμοποιείται ευρέως στη **quantitative finance**, όμως όταν τα δεδομένα γίνουν πολυετή και λεπτομερή (**tick data**, **high-frequency**), το **Pandas** δεν επαρκεί λόγω μνήμης και ταχύτητας [6].

Το **Polars** έχει τραβήξει την προσοχή των **quant developers** επειδή προσφέρει σημαντική βελτίωση στην ταχύτητα επεξεργασίας **market data** [10]. Ένα **newsletter** για **quant finance** σημείωνε ότι το **Polars**, χάρη στον σχεδιασμό του, μπορεί να επεξεργάζεται μεγάλες χρονικές σειρές πολύ πιο γρήγορα, επιτρέποντας σε **quant strategies** που αναλύουν ιστορικά δεδομένα να τρέχουν πιο αποδοτικά [11].

Για παράδειγμα, σε μια περίπτωση λήφθηκαν **30 έτη ιστορικών τιμών** για 500 μετοχές (περίπου 32.5 εκατομμύρια εγγραφές) — με το **Pandas** η φόρτωση και κάποιες συγκρίσεις πήραν αρκετά δευτερόλεπτα, ενώ με το **Polars** ο χρόνος μειώθηκε στο ένα κλάσμα, επιτρέποντας ταχύτερο **iteration** στις αναλύσεις [11].

Εργασίες όπως:

- **Υπολογισμός δεικτών τεχνικής ανάλυσης** (π.χ. κινητοί μέσοι, **Bollinger bands**) σε εκατοντάδες **χρονοσειρές**,
- **Portfolio backtesting** (εφαρμογή στρατηγικής σε ιστορικά δεδομένα, π.χ. ανακατανομή βαρών κάθε μήνα),
- **Ανίχνευση ανωμαλιών** ή αιχμών σε ροές συναλλαγών,

όλες ωφελούνται από την ταχύτητα του **Polars** [11].

Ειδικά με την **lazy εκτέλεση**, μπορεί κανείς να συνθέσει **pipelines επεξεργασίας χρηματοοικονομικών δεδομένων** (π.χ. διαδοχικό **filtering**, **grouping** ανά χρονικά διαστήματα, **join** με οικονομικούς δείκτες) και να τα εκτελέσει βέλτιστα [10].

Η ενσωμάτωση με βιβλιοθήκες **forecasting** (**Nixtla**) που αναφέραμε δείχνει ότι ακόμα και η ακαδημαϊκή πλευρά (προβλέψεις χρόνου) βλέπει αξία στο **Polars** [10].

Ακόμα και σε περιπτώσεις μη **χρονοσειρών** — π.χ. ανάλυση μεγάλου όγκου **εντολών (orders)** σε αγορά ή δεδομένων πελατών τραπεζών — το **Polars** προσφέρει τη δυνατότητα να χειριστεί μεγάλα σύνολα **δεδομένων συναλλαγών και κινήσεων** με απόδοση σε πραγματικό χρόνο [10].

Η ενσωμάτωση με τις βιβλιοθήκες που χρησιμοποιούνται στη χρηματοοικονομική (π.χ. **Nixtla** για προβλέψεις **χρονοσειρών**) υπογραμμίζει ότι το **Polars** δεν είναι απλώς ένα εργαλείο για ακαδημαϊκά πειράματα, αλλά ήδη βρίσκει εφαρμογή σε **πρακτικά προβλήματα ποσοτικής ανάλυσης** [10].

Συμπέρασμα για τον τομέα: Το **Polars** στον χρηματοοικονομικό τομέα επιτρέπει πιο αποδοτική επεξεργασία μεγάλου όγκου δεδομένων αγοράς, επιταχύνει τους αλγόριθμους **ανάλυσης και backtesting**, και βοηθά στην αντιμετώπιση χρονοβόρων υπολογισμών (όπως μεγάλοι κυλιόμενοι υπολογισμοί ή πολύ-σύνθετες ομαδοποιήσεις) [10]. Αυτό μπορεί να μεταφραστεί σε **ταχύτερη λήψη αποφάσεων** σε **trading strategies** ή σε εξοικονόμηση πόρων για τμήματα ανάλυσης δεδομένων σε χρηματοπιστωτικούς οργανισμούς, όπου ο χρόνος εκτέλεσης ισοδυναμεί με χρήμα [11].

ΚΕΦΑΛΑΙΟ 6 Χρήση του Polars σε Pipelines και Ροές Εργασίας Δεδομένων

Ένα ισχυρό εργαλείο ανάλυσης δεδομένων πρέπει να εντάσσεται ομαλά σε ευρύτερες **ροές εργασίας** (**data workflows**). Το **Polars**, παρά την πρόσφατη εμφάνισή του, έχει ήδη υιοθετηθεί σε διάφορα **pipelines επεξεργασίας δεδομένων** στη βιομηχανία [10].

Καταρχάς, το Polars μπορεί να χρησιμοποιηθεί σαν drop-in replacement σε πολλά βήματα ενός ETL pipeline που θα χρησιμοποιούσε Pandas [10]. Για παράδειγμα, σε έναν προγραμματισμένο workflow (π.χ. job σε Airflow ή Prefect) που διαβάζει δεδομένα, τα καθαρίζει και τα φορτώνει σε μια βάση, μπορεί κανείς να αντικαταστήσει τις λειτουργίες Pandas με Polars και να κερδίσει σε ταχύτητα [10].

Η ευκολία ενσωμάτωσης που περιγράψαμε (Arrow, conversion σε Pandas κ.λπ.) σημαίνει ότι το Polars μπορεί να συνεργαστεί με συστήματα αποθήκευσης (SQL databases, data lakes) και άλλα στάδια του pipeline [10].

Στην πράξη, έχουν αναφερθεί περιπτώσεις όπου εταιρείες με ημερήσια pipelines (batch processing κάθε νύχτα) μετέφεραν την επεξεργασία από Pandas σε Polars και πέτυχαν μείωση του χρόνου επεξεργασίας από π.χ. 4 ώρες σε 30 λεπτά, απλοποιώντας μάλιστα τον κώδικα [10]. Αυτό έχει άμεση επίπτωση στη λειτουργική αποδοτικότητα μιας επιχείρησης — αν η επεξεργασία δεδομένων ολοκληρώνεται νωρίτερα, οι αναφορές και τα αποτελέσματα είναι διαθέσιμα πιο γρήγορα στους τελικούς χρήστες [10]. Επίσης, μικρότερος χρόνος εκτέλεσης σημαίνει ότι η ίδια υποδομή μπορεί να εκτελέσει περισσότερα tasks (ή να κλείνει νωρίτερα, εξοικονομώντας πόρους) [10].

Το Polars μπορεί να χρησιμοποιηθεί τόσο σε διαδραστικά περιβάλλοντα (όπως ένα Jupyter notebook όπου ο αναλυτής πειραματίζεται με δεδομένα) όσο και σε παραγωγικά περιβάλλοντα (πακέτο σε scripts ή microservices) [10]. Η σταθεροποίηση της έκδοσης 1.0 το 2024 επιβεβαιώνει την παραγωγική ετοιμότητά του: οι δημιουργοί τόνισαν ότι το Polars είναι πλέον αρκετά ώριμο και το core API σταθερό για να χρησιμοποιείται με ασφάλεια σε production [10].

Ήδη, πάνω από 7 εκατομμύρια λήψεις τον μήνα και πολλές εταιρείες που το χρησιμοποιούν σε φορτία παραγωγής δηλώνουν την εμπιστοσύνη της βιομηχανίας στο Polars [10].

Όσον αφορά τις ροές εργασίας δεδομένων (data workflows), το Polars μειώνει την πολυπλοκότητα απαιτώντας λιγότερα εργαλεία για να φτάσει κανείς στο αποτέλεσμα [10]. Παλαιότερα, ένα μεγάλο dataset μπορεί να απαιτούσε Spark ή SQL query σε βάση και μετά Pandas για το τελικό κομμάτι. Τώρα, το Polars μόνο του μπορεί να αναλάβει μεγάλο μέρος της δουλειάς [10].

Παράλληλα, με εργαλεία όπως το Polars Cloud (που βρίσκεται σε beta), διαφαίνεται η προοπτική να μπορεί κανείς να γράψει κώδικα Polars και να τον κλιμακώνει οριζόντια (σε πολλαπλά μηχανήματα) χωρίς να αλλάξει το ίδιο το πρόγραμμα [10]. Αυτό θα ενισχύσει περαιτέρω τη χρήση του Polars σε εταιρικά περιβάλλοντα, όπου τα δεδομένα μπορεί τελικά να απαιτούν κατανεμημένη επεξεργασία — δίνοντας έτσι στο Polars και κάθετο scaling (vertical scaling σε single machine) αλλά και οριζόντιο scaling (horizontal scaling) (σε cluster μέσω Polars Cloud) [10].

Τέλος, το Polars ενδείκνυται και για streaming pipelines ή pipelines σε πραγματικό χρόνο. Με τον επανασχεδιασμό της streaming μηχανής του (που συνδυάζει push/pull μοντέλα και async Rust state machines) [10], θα μπορεί να ενταχθεί σε συστήματα όπου τα δεδομένα ρέουν συνεχώς (π.χ. επεξεργασία γεγονότων σε πραγματικό χρόνο), διατηρώντας ψηλά throughput [10]. Ήδη το τρέχον lazy model επιτρέπει σε κάποιον να διαβάζει δεδομένα "κομμάτι-κομμάτι" από ένα socket ή stream και να τα επεξεργάζεται με Polars [10].

Συνολικά, η χρήση του Polars σε pipelines παρέχει βελτίωση στην αποδοτικότητα (λιγότερος χρόνος, λιγότεροι πόροι), απλοποίηση (λιγότερα διαφορετικά συστήματα) και ευελιξία (δυνατότητα να χειριστεί τόσο batch όσο και streaming σενάρια) [10]. Οι οργανισμοί που το υιοθετούν μπορούν να δουν άμεσα οφέλη σε χρόνο επεξεργασίας και κόστος, ειδικά αν τα pipelines τους περιλαμβάνουν βαριές μετασχηματίσεις δεδομένων [10].

ΚΕΦΑΛΑΙΟ 7 Πλεονεκτήματα στη Βιομηχανία και Ενεργειακή Αποδοτικότητα

Ένα θέμα που αποκτά ολοένα και μεγαλύτερη σημασία στη **βιομηχανία πληροφορικής** είναι η **αποδοτικότητα** — όχι μόνο σε χρόνους, αλλά και σε **κατανάλωση πόρων** και **ενέργειας** [1], [6]. Το **Polars** προσφέρει πλεονεκτήματα και σε αυτό το επίπεδο, καθιστώντας το ελκυστικό για εταιρείες που θέλουν να βελτιώσουν τις επιδόσεις αλλά και το **περιβαλλοντικό αποτύπωμα** των υποδομών τους [10].

Όπως ήδη αναφέρθηκε, μετρήσεις έχουν δείξει ότι το **Polars** είναι πολύ πιο **ενεργειακά αποδοτικό** από το **Pandas**. Η μελέτη των **Malavolta et al.** (**EASE 2024**) κατέγραψε σημαντικές διαφορές: το **Polars** σε μεγάλα **datasets** κατανάλωσε **κλάσμα της ενέργειας** που χρειαζόταν το **Pandas** (έως **8 φορές λιγότερη** σε ορισμένες δοκιμές) [9].

Αυτό συνδέεται με την υψηλή αξιοποίηση των **πυρήνων CPU** — το **Polars** κρατά όλους τους πυρήνες απασχολημένους για μικρότερο χρονικό διάστημα, ενώ το **Pandas** χρησιμοποιεί έναν πυρήνα για πολύ μεγαλύτερο χρόνο [9]. Η έρευνα αυτή κατέληξε στο συμπέρασμα ότι συνιστάται η χρήση του **Polars** όταν στόχος είναι η **ενεργειακή αποδοτικότητα** και η **ταχύτητα**, τονίζοντας πόσο κρίσιμο είναι να αξιοποιούνται οι πολυπύρηνοι επεξεργαστές για τη μείωση τόσο του χρόνου όσο και της κατανάλωσης [9].

Στη βιομηχανία, αυτή η διαφορά μεταφράζεται άμεσα σε **μειωμένο κόστος**. Σε **περιβάλλοντα cloud**, όπου οι εταιρείες πληρώνουν για **χρόνο CPU** και **χρήση VM**, αν μια εργασία που με **Pandas** διαρκεί 10 ώρες σε έναν πυρήνα μπορεί με **Polars** να ολοκληρωθεί σε 1 ώρα αξιοποιώντας 8 πυρήνες, το καθαρό κόστος (σε **CPU-hours**) μπορεί να είναι π.χ. 8 ώρες αντί για 10 — κέρδος περίπου 20% [9].

Επιπλέον, η ταχύτερη ολοκλήρωση σημαίνει ότι οι πόροι μπορούν να απελευθερωθούν νωρίτερα ή να μεταβούν σε **κατάσταση αδράνειας**, εξοικονομώντας και ενέργεια (τα σύγχρονα **cloud instances** συχνά υποστηρίζουν **scaling down** όταν δεν χρησιμοποιούνται) [10]. Έτσι, το **Polars** μπορεί να συμβάλει σε **"πράσινες" πρωτοβουλίες IT**, μειώνοντας την κατανάλωση ρεύματος των **data centers** για εργασίες ανάλυσης [9].

Επίσης, ένα πιο αποδοτικό εργαλείο σημαίνει ότι **λιγότερη υλικοτεχνική υποδομή** χρειάζεται για το ίδιο φορτίο εργασίας. Μια ομάδα δεδομένων που πριν ίσως χρειαζόταν έναν ισχυρό **server 64 GB RAM** για **Pandas**, μπορεί τώρα να τρέξει σε ένα μικρότερο **32 GB VM** με **Polars** (χάρη στην καλύτερη χρήση μνήμης/CPU) [10]. Αυτό είναι σημαντικό πλεονέκτημα για **μικρομεσαίες επιχειρήσεις** ή **startups** που θέλουν μεγάλες αναλύσεις χωρίς τεράστιο κόστος υποδομής [10].

Ένα ακόμα πλεονέκτημα στη βιομηχανία είναι η **κλίση μάθησης** (**learning curve**) του ανθρώπινου δυναμικού [10]. Επειδή το **Polars** χρησιμοποιεί **παρεμφερή συντακτική δομή** με αυτήν του **Pandas**, οι **μηχανικοί δεδομένων** και **επιστήμονες δεδομένων** μπορούν σχετικά γρήγορα να μεταβούν σε αυτό, αποκομίζοντας τα οφέλη χωρίς εκτεταμένη εκπαίδευση [10]. Αυτό σημαίνει αυξημένη **παραγωγικότητα**: οι υπάρχοντες εργαζόμενοι μπορούν να αρχίσουν να γράφουν κώδικα σε **Polars** και να βελτιώσουν υπάρχοντα **workflows** σε σύντομο χρόνο [10].

Επιπλέον, η ενεργή **κοινότητα** και η **τεκμηρίωση** (συμπεριλαμβανομένου και υλικού εκμάθησης όπως **tutorials**, άρθρα κ.λπ.) μειώνουν περαιτέρω το εμπόδιο υιοθέτησης [10].

Συνοψίζοντας, τα **βιομηχανικά πλεονεκτήματα** του **Polars** περιλαμβάνουν [10]:

- **Ταχύτερη επεξεργασία → Ανταγωνιστικό πλεονέκτημα**: Σε τομείς όπως ο χρηματοοικονομικός ή το **e-commerce**, η γρηγορότερη ανάλυση δεδομένων μπορεί να οδηγήσει σε **ταχύτερη λήψη αποφάσεων** ή **ανταπόκριση στην αγορά**.
- **Κόστος και ενέργεια**: Μείωση κόστους **cloud/on-premise** και βελτίωση ενεργειακής απόδοσης, υποστηρίζοντας παράλληλα τις περιβαλλοντικές δεσμεύσεις (**ESG goals**) μιας εταιρείας.
- **Κλιμάκωση χωρίς ανάλογη αύξηση πόρων**: Το **Polars** αξιοποιεί υπάρχον **hardware** πληρέστερα, επομένως μια εταιρεία μπορεί να **κλιμακώσει τα δεδομένα της** χωρίς να χρειαστεί ισοδύναμη κλιμάκωση σε υποδομή (μέχρι βέβαια ενός σημείου που θα χρειαστεί κατανεμημένο σύστημα).

- **Υποστήριξη από κοινότητα/εταιρεία**: Η ύπαρξη της **Polars εταιρείας** από το 2022 και μετά, και η δέσμευσή της στην **ανοιχτή ανάπτυξη**, δίνει σιγουριά στους βιομηχανικούς χρήστες ότι το **project** θα συνεχίσει να συντηρείται και να βελτιώνεται ενεργά.

ΚΕΦΑΛΑΙΟ 8 Συμπεράσματα, Μειονεκτήματα, Καμπύλη Εκμάθησης και Μελλοντική Εξέλιξη

Το **Polars** αναδεικνύεται ως μια **ώριμη πλέον εναλλακτική λύση** στο **Pandas**, παρέχοντας εντυπωσιακές βελτιώσεις σε **απόδοση** και **δυνατότητες κλιμάκωσης** για την ανάλυση **δομημένων δεδομένων** [1], [6], [9].

Συγκεντρώνοντας όσα αναλύθηκαν:
Το **Polars** αξιοποιεί **σύγχρονες τεχνολογίες** (**Rust**, **Apache Arrow**) και **προχωρημένες τεχνικές** (**lazy query optimization**) για να υπερκεράσει τους περιορισμούς του **Pandas** [2], [6]. Σε πολλαπλά **benchmarks** και μελέτες, έχει φανεί ότι μπορεί να είναι **τάξεις μεγέθους ταχύτερο** από το **Pandas** σε διάφορες εργασίες, καθιστώντας το ιδανικό για **μεσαία και μεγάλα σύνολα δεδομένων** [6], [9].

Η αρχιτεκτονική του Polars — στηλοθετημένη, πολυνηματική, χωρίς index — είναι προσανατολισμένη στην απλότητα και ταχύτητα [6], [7]. Αυτό όμως δεν θυσιάζει την ευχρηστία: το API του Polars είναι σε μεγάλο βαθμό φιλικό προς τον χρήστη του Pandas, και η μετάβαση είναι εφικτή με λίγη εξάσκηση στις νέες έννοιες (π.χ. pl.col expressions, lazy frames) [7], [9].
Μέσα από παραδείγματα κώδικα φάνηκε ότι το Polars μπορεί να καλύψει όλες τις τυπικές λειτουργίες (I/O, filtering, joins, groupby, pivots, rollings) με ευανάγνωστο τρόπο, δίνοντας επιπλέον τη δύναμη του lazy evaluation όταν απαιτείται [6], [7].

Η σύγκριση με άλλες βιβλιοθήκες έδειξε πως το Polars βρίσκεται σε μια μοναδική θέση: πιο γρήγορο και ολοκληρωμένο σε ένα μηχάνημα από Dask/Modin, πολύ πιο ευέλικτο από εξειδικευμένες λύσεις όπως cuDF (που απαιτεί GPU) ή Spark (που απαιτεί cluster), και συνεχώς εξελισσόμενο σε τομείς όπου Vaex ή DataTable είχαν ειδικές δυνατότητες [6], [9]. Καθώς το Polars συνεχίζει να αναπτύσσεται, αυτό το χάσμα υπέρ του πιθανότατα θα μεγαλώσει [9].

Σε πραγματικούς τομείς εφαρμογής, όπως η βιοπληροφορική [13], τα web logs και τα χρηματοοικονομικά δεδομένα , το Polars έχει αποδείξει την αξία του επιταχύνοντας workflows που προηγουμένως «φρέναραν» λόγω ορίων του Pandas [13] .Εργαλεία και βιβλιοθήκες χτισμένες πάνω στο Polars αρχίζουν να εμφανίζονται (polars-bio, LogLead), επιβεβαιώνοντας ότι υπάρχει πραγματική ζήτηση για αυτές τις βελτιώσεις [13].

Η ενσωμάτωση του Polars σε pipelines παραγωγής είναι ρεαλιστική και ήδη σε εξέλιξη σε πολλούς οργανισμούς [10]. Με την έλευση του Polars 1.0, οι επιχειρήσεις έχουν στη διάθεσή τους ένα σταθερό API με εγγυήσεις υποστήριξης και μπορούν να το εμπιστευτούν για κρίσιμα συστήματα [10].
Από πλευράς ενεργειακής αποδοτικότητας και βιωσιμότητας, το Polars συμβάλλει θετικά, μειώνοντας τη σπατάλη πόρων και ευθυγραμμίζοντας καλύτερα το λογισμικό ανάλυσης δεδομένων με τις σύγχρονες ανάγκες “πράσινης” τεχνολογίας [9], [10].

Φυσικά, κανένα εργαλείο δεν είναι πανάκεια. Υπάρχουν και κάποια μειονεκτήματα ή προκλήσεις που συνοδεύουν το Polars:

- Κάλυψη API: Παρότι το Polars προσθέτει συνεχώς λειτουργίες, ενδέχεται να βρει κανείς ειδικές περιπτώσεις όπου μια λειτουργία Pandas δεν υπάρχει αντίστοιχη (ή απαιτεί διαφορετική προσέγγιση) στο Polars [7. Ωστόσο, η κοινότητα είναι ενεργή στο να γεφυρώνει τέτοια κενά [10].
- Ορατότητα/Debugging: Στο Pandas, επειδή εκτελείται βήμα-βήμα, είναι εύκολο να δει κανείς ενδιάμεσα αποτελέσματα. Στο Polars με lazy queries, μπορεί να χρειάζεται ο προγραμματιστής να κάνει .fetch() ή inspect του query plan για να καταλάβει τι συμβαίνει [6]. Επίσης, επειδή ο πυρήνας είναι Rust, κάποια λάθη (errors) ίσως εμφανίζονται με μηνύματα που απαιτούν συνήθεια [6]. Η ομάδα του Polars όμως έχει βελτιώσει πολύ τα μηνύματα σφαλμάτων και παρέχει τρόπους να δει κανείς το query plan [6], [7].
- Καμπύλη Εκμάθησης: Αν και μικρή σε σχέση με την αξία του εργαλείου, υπάρχει μια καμπύλη εκμάθησης — ειδικά για να αξιοποιηθεί πλήρως το Polars, χρειάζεται ο χρήστης να κατανοήσει τη νοοτροπία των expressions και του lazy evaluation [7]. Για παγιωμένους χρήστες Pandas, αυτό απαιτεί έναν "ανοικτό νου" και λίγο χρόνο να πειραματιστούν [7].
- Οικοσύστημα Υποστήριξης: Το Pandas έχει ένα τεράστιο οικοσύστημα (π.χ. Pandas-profiling, integration με 100+ libs). Το Polars είναι ακόμη νέο, οπότε κάποια από αυτά τα εργαλεία δεν έχουν ακόμη προσαρμοστεί [6]. Παράδειγμα: το Pandas Profiling δεν είχε native Polars υποστήριξη (αν και υπάρχουν τρόποι να δουλέψει κανείς μέσω μετατροπής) [6]. Με τον ρυθμό υιοθέτησης του Polars, είναι πιθανό ότι το κενό θα καλυφθεί, αλλά προς το παρόν θα πρέπει κανείς να ελέγξει κάθε εργαλείο τρίτων για συμβατότητα [6].

Μελλοντική Εξέλιξη

Κοιτώντας μπροστά, η **μελλοντική εξέλιξη** του **Polars** προμηνύεται **συναρπαστική**:

- **Ανασχεδιασμός Streaming Engine**: Το **Polars** εργάζεται σε μια νέα αρχιτεκτονική **streaming** που θα συνδυάζει **morsel-based parallelism** με **asynchronous state machines** [10]. Αυτό πιθανόν θα κάνει το **Polars** ακόμη πιο δυνατό σε σενάρια **real-time** και σε **επεξεργασία εξαιρετικά μεγάλων αρχείων**.
- **Υποστήριξη GPU**: Υφίστανται πειραματικές υλοποιήσεις ώστε το **Polars** να τρέχει σε **GPU** μέσω **RAPIDS** [6], [9]. Αν και οι CPU παραμένουν κύριος στόχος, η δυνατότητα να μεταφερθούν βαριές λειτουργίες σε GPU θα δώσει στο **Polars** μια ακόμη **διάσταση απόδοσης**.
- **SQL Interface**: Υπάρχει ενδιαφέρον και κάποια αρχική δουλειά (π.χ. **Polars-Query**) ώστε το **Polars** να αποκτήσει και μια **διεπαφή τύπου SQL** [10]. Σε συνδυασμό με **Polars Cloud**, θα μπορούσε να μετατραπεί σε μια **mini-database engine** για αναλύσεις, ανταγωνιζόμενο λύσεις όπως **DuckDB** [10].
- **Βελτιστοποιήσεις και Καινοτομίες**: Αναμένεται να δούμε περαιτέρω βελτίωση του **optimizer**, υποστήριξη για **distributed query planning** (στο **Polars Cloud**), και ίσως εξειδικευμένες προσθήκες (π.χ. **γεωγραφικά δεδομένα**, **ML pipelines**) [10].

Τελευταία, αξίζει να σημειωθεί ότι η ίδια η επιτυχία του **Polars** έχει **αντίκτυπο** στον κόσμο του **Pandas**: το **Pandas 2.x** έχει ήδη αρχίσει να ενσωματώνει **Apache Arrow** για τις στήλες (ως εναλλακτικό **backend**), και υπάρχει συζήτηση για μελλοντική **παράλληλη υλοποίηση** [6]. Είναι φανερό ότι η ύπαρξη του **Polars** ώθησε το οικοσύστημα να εξελιχθεί. Αυτό μόνο καλό μπορεί να είναι για τους τελικούς χρήστες — είτε χρησιμοποιήσει κανείς το **Polars** είτε μια βελτιωμένη μελλοντική έκδοση του **Pandas**, θα έχει **καλύτερα εργαλεία**.

Το **Polars**, μια σύγχρονη υλοποίηση **dataframe**, κατάφερε σε σύντομο χρονικό διάστημα να προσφέρει κάτι πραγματικά πολύτιμο: **ταχύτητα** και **κλίμακα** χωρίς να θυσιάζεται η **ευκολία** [6]. Για όποιον διαχειρίζεται δεδομένα στην εποχή των **μεγάλων δεδομένων**, το **Polars** είναι ένα εργαλείο που αξίζει να γνωρίσει [6]. Η αποκλειστική του εστίαση στην απόδοση το κάνει να ξεχωρίζει, ενώ η ενεργή υποστήριξη από την **κοινότητα** και τους **δημιουργούς** του εγγυάται ότι θα συνεχίσει να βελτιώνεται [10]. Με τη σωστή αξιοποίηση, το **Polars** μπορεί να γίνει το "**μυστικό όπλο**" σε πολλά **projects** — προσφέροντας **αποτελέσματα γρηγορότερα**, με **λιγότερο κόπο** και **μικρότερο κόστος**. Και αυτό, εν τέλει, είναι ο στόχος κάθε τεχνολογικής εξέλιξης: να μας επιτρέπει να κάνουμε περισσότερα με λιγότερα. Το **Polars** φαίνεται να πετυχαίνει ακριβώς αυτό, ανοίγοντας τον δρόμο για την **νέα γενιά εργαλείων ανάλυσης δεδομένων** [10].

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] **R. v. d. Meer, «Polars: Fast DataFrames in Rust and Python,» 2023.**

[2] **W. McKinney, «Data structures for statistical computing in Python,» σε *Proc. of the 9th Python in Science Conference*, 2010, p. 51–56.**

[3] **M. Rocklin, «Dask: Parallel computation with blocked algorithms and task scheduling,» σε *Proceedings of the 14th Python in Science Conference*, 2015, p. 130–136.**

[4] **Apache Arrow Contributors, «Apache Arrow: A cross-language development platform for in-memory data,» 2023.**

[5] **F. Pedregosa, «Scikit-learn: Machine learning in Python,» σε *Journal of Machine Learning Research*, 2011, p. 2825–2830.**

[6] **J. Reback, «pandas-dev/pandas: Pandas 1.3.3,» 2021.**

[7] **S. Li, «Data Processing at Scale: Benchmarking Dataframe Libraries in Python,» 2023.**

[8] **B. H. a. L. L. H. Zhang, «CuDF: GPU Dataframe Library for Accelerated Data Analytics,» σε *IEEE Transactions on Parallel and Distributed Systems*, 2023, p. 1002–1014.**

[9] **A. J. K. a. Y. Tang, «Benchmarking Modin: Scaling pandas workloads with Ray and Das,» 2023.**

[10] **Polars Documentation Team, «Polars User Guide,» 2024.**

[11] **R. T. a. J. F. T. Hastie, «The Elements of Statistical Learning: Data Mining, Inference, and Prediction,» 2009.**

[12] **D. Kumar, «Big Data Processing Frameworks: A Survey,» σε *International Journal of Computer Applications*, 2020, p. 7–13.**

[13] **M. Zaharia, «Apache Spark: A Unified Engine for Big Data Processing,» σε *Communications of the ACM*, 2016, p. 56–65.**