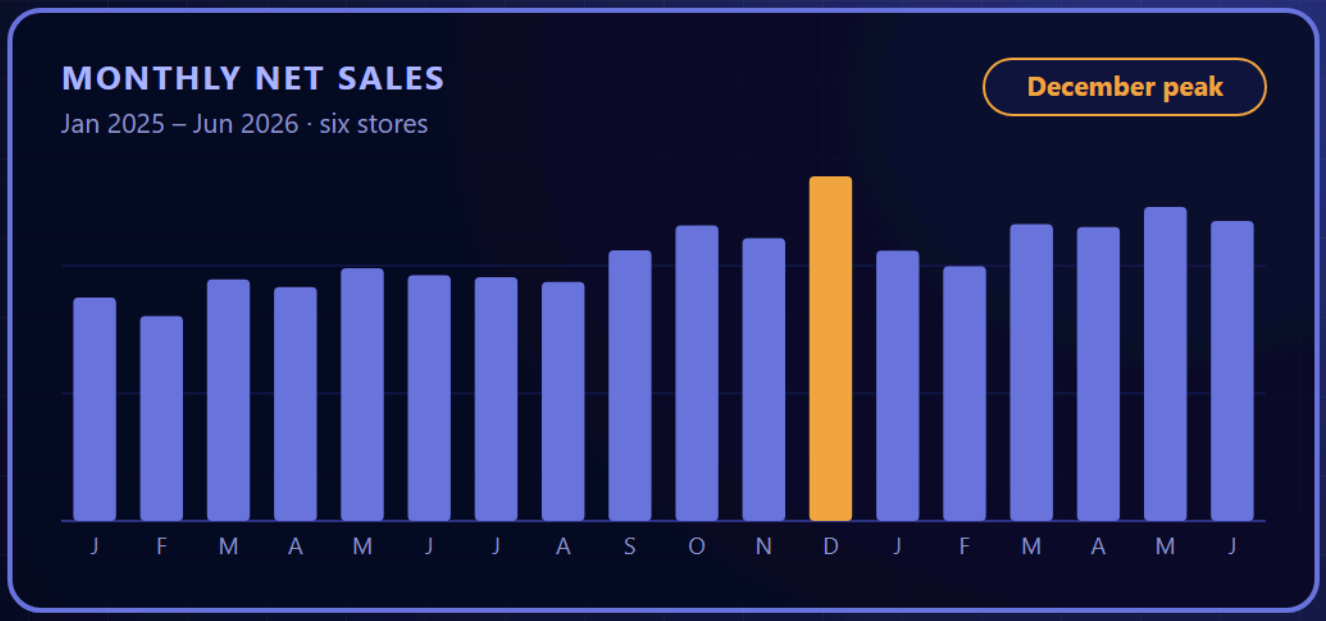



```pandas
(sales
 .groupby("store_id")
 .agg(net=("net_sales",
           "sum")))

# eager · indexed
```

```Polars
(sales
 .group_by("store_id")
 .agg(pl.col("net_sales")
      .sum()))

# expressions · lazy
```

PRACTICAL PYTHON DATA ANALYSIS

Data Analysis with pandas and Polars

Clean, transform, analyze, visualize, and report real business data in Python

CSV · Parquet · JSON · Cleaning · Joins · Window Functions
Time Series · Cohorts · matplotlib · DuckDB · Reports

FOUNDATIONS · CLEANING · ANALYSIS · REPORTING

THENEXBOOKS

Data Analysis with pandas and Polars

Clean, Transform, Analyze, Visualize, and Report Real Business Data in Python
From messy exports to a tested, automated monthly report — every example run and reproducible

First Edition

THENEXBOOKS

Data Analysis with pandas and Polars
Clean, Transform, Analyze, Visualize, and Report Real Business Data in Python

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. Code examples may be used freely in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Analyses can be wrong; important business decisions deserve a second look at the data and the assumptions behind it. Python, pandas, Polars, NumPy, matplotlib, DuckDB, and the other tools discussed in this book evolve quickly; always consult the official documentation for the versions you use.

Python is a trademark of the Python Software Foundation. This book is independent and is not affiliated with, sponsored by, or endorsed by the Python Software Foundation, the pandas project, Polars or the company behind it, the NumPy project, the matplotlib project, the DuckDB project or DuckDB Foundation, the Apache Software Foundation, NumFOCUS, or any other project or company mentioned. Apache Arrow and Apache Parquet are trademarks of the Apache Software Foundation. Microsoft, Windows, Excel, and Visual Studio Code are trademarks of the Microsoft group of companies. macOS is a trademark of Apple Inc. Linux is a registered trademark of Linus Torvalds. PyCharm is a trademark of JetBrains s.r.o. Jupyter is a trademark of LF Charities. Other product names are trademarks of their respective owners. Their use in this book is for identification only and does not imply endorsement.

Cinderwell Coffee, the coffee-shop chain used as the running example in this book, is fictional, as are its stores, employees, loyalty members, products, prices, promotions, and every row of its data. Real city names are used only as settings. The datasets are generated by a seeded Python script included with the book; any patterns in them were designed for teaching and say nothing about real businesses or customers. All email addresses use the reserved example.com domain.

Preface

Most business questions end up as data questions. Which stores are growing? Did the spring promotion pay for itself? How many new loyalty members come back the following month? Are we scheduling staff when customers actually arrive? The answers sit in files that are rarely ready to use: point-of-sale exports with odd headers, membership lists with duplicates and three spellings of "yes", shift logs with missing clock-outs. Turning those files into numbers you can trust, and then into charts and reports people can act on, is the everyday work of data analysis.

This book teaches that work in Python with the two DataFrame libraries you are most likely to meet today. **pandas** is the long-established standard, used in countless tutorials, notebooks, and libraries. **Polars** is a newer library built on Apache Arrow, with an expression-based API, lazy queries, and multithreaded execution. Instead of picking a side, we show every core task in pandas first and then in Polars, and we explain where the two differ and why. By the end, you can read and write both, move data between them, and choose the right one for each job.

Everything is hands-on. We do not collect features in isolation; we work through the real analysis tasks of one company, from the first look at a messy export to an automated monthly report and a quarterly business review. Every example in the book was run, and every number you see was printed by the code next to it.

Who This Book Is For

- **Analysts moving from spreadsheets or SQL** who want repeatable, scriptable analysis in Python.
- **Developers** who need to clean, combine, and summarize data for a product, a report, or a one-off investigation.
- **pandas users curious about Polars** who want an honest, side-by-side comparison rather than hype.
- **Students and career changers** building practical data skills with realistic, messy data.

You should know basic Python: variables, lists, dictionaries, functions, loops, and running a script. You do not need prior experience with pandas, Polars, or statistics; we explain each idea when it first solves a problem.

What You Will Learn

- How pandas and Polars think: Series, DataFrames, indexes, data types, expressions, and lazy queries.
- Reading and writing CSV, Parquet, JSON, Excel, and databases, and why file formats matter.
- Cleaning messy data: column names, types, duplicates, missing values, text, categories, dates, and time zones.

- Validating data with explicit rules, quality reports, and reconciliation checks.
- Transforming and combining data: new columns, grouping, joins, reshaping, nested JSON, and window functions.
- Analyzing business data: time series, descriptive statistics, cohorts and retention, RFM segmentation, product and basket analysis, promotions, and labor productivity.
- Visualizing with matplotlib and designing charts that explain a finding rather than decorate a page.
- Formatting tables and automating a monthly report that rebuilds itself from the data.
- Performance and scale with memory-aware types, lazy evaluation, streaming, and DuckDB SQL.
- Writing reliable analysis code with functions, tests, and reproducible results — and choosing between pandas and Polars.

How This Book Is Organized

Part I — Foundations (Chapters 1–5) introduces the analyst's workflow, sets up your tools and the project, explains the fundamentals of pandas and of Polars, and covers reading and writing data. **Part II — Cleaning and Preparing Data** (Chapters 6–10) covers selecting and filtering, cleaning messy files, text and categorical data, dates and time zones, and validating data quality. **Part III — Transforming and Combining Data** (Chapters 11–15) covers new columns, grouping and aggregation, joins, reshaping and nested data, and window functions. **Part IV — Analyzing Business Data** (Chapters 16–20) applies everything to time series, descriptive statistics, customer analytics, product and basket analysis, and operations analytics. **Part V — Visualizing and Reporting** (Chapters 21–24) covers matplotlib, chart design, formatted tables, and an automated monthly report. **Part VI — Performance and Production** (Chapters 25–28) covers performance and scale, DuckDB with DataFrames, reliable and tested analysis code, and a capstone quarterly business review with a guide to choosing between pandas and Polars. The appendices provide a side-by-side cheat sheet, templates and reference tables, a production checklist, and a glossary.

Each chapter includes exercises under **Try It Yourself** and ends with a **Key Takeaways** box for quick review.

The Running Example: Cinderwell Coffee

Throughout the book you are the new data analyst at **Cinderwell Coffee**, a fictional chain of six coffee shops in Seattle, Portland, Tacoma, Spokane, Boise, and Eugene: three cafes, a commuter kiosk, a drive-thru, and a new campus cafe that opens partway through the data. Dana, the Director of Operations, wants to know how each store is doing and whether staffing matches demand. Marco, who runs marketing and the loyalty program, wants to understand members and promotions. Ingrid, the Finance Manager, wants margins, labor cost, and a quarterly review she can present.

The data covers eighteen months — January 2025 to June 2026 — with more than half a million receipts and over 800,000 line items in Parquet files, plus products, stores, staff, shifts, a nested JSON file of promotions, a loyalty member list with deliberate data-quality problems, and a messy point-of-sale export to clean. A seeded Python script generates all of it, so your numbers match the book exactly.

Versions and Conventions

Examples were tested with **Python 3.12**, **pandas 3.0** (3.0.6), **Polars 1.44**, **NumPy 2.5**, **pyarrow 25**, **matplotlib 3.11**, and **DuckDB 1.5**. pandas 3 changed several long-standing defaults, including Copy-on-Write and a new default string type; where that matters, we say so, and most examples also run on recent pandas 2.x releases. Polars evolves quickly, so where a feature is recent we say "recent versions" rather than promising an exact release. Where an optional tool is not part of the tested setup, we label its code clearly.

> **TIP**
> A recommended practice or a technique that makes your work easier.

> **NOTE**
> Background information, version differences, or a useful aside.

> **WARNING**
> A common mistake or something that can cause wrong results or data loss.

Code appears in `monospace`. The label above each code block names the file, such as `ch12_store_summary.py`, notes whether it uses pandas or Polars, or says **Terminal** or **Output**. Commands are shown as `python script.py`; on Windows you can also type `py script.py`. Money is shown in US dollars. Long lines are wrapped to fit the page.

Contents

Preface

PART I — FOUNDATIONS

1 Data Analysis in Python: pandas, Polars, and the Analyst's Workflow

2 Setting Up Your Workspace: Python, Packages, and the Cinderwell Project

3 pandas Fundamentals: Series, DataFrames, Indexes, and Data Types

4 Polars Fundamentals: DataFrames, Expressions, and Lazy Queries

5 Reading and Writing Data: CSV, Parquet, JSON, Excel, and Databases

PART II — CLEANING AND PREPARING DATA

6 Selecting, Filtering, and Sorting Data

7 Cleaning Messy Data: Columns, Types, Duplicates, and Missing Values

8 Text and Categorical Data: Strings, Categories, and Mappings

9 Dates, Times, and Time Zones

10 Validating Data Quality: Rules, Checks, and Reconciliation

PART III — TRANSFORMING AND COMBINING DATA

11 Creating New Columns: Expressions, Conditions, and Custom Functions

12 Grouping and Aggregation: Split, Apply, Combine

13 Joining and Combining Tables

14 Reshaping Data: Pivot, Unpivot, and Nested Data

15 Window Functions: Rankings, Running Totals, and Period-over-Period Change

PART IV — ANALYZING BUSINESS DATA

16 Time Series Analysis: Resampling, Rolling Windows, and Seasonality

17 Descriptive Statistics: Distributions, Outliers, and Fair Comparisons

18 Customer Analytics: Cohorts, Retention, and RFM Segmentation

19 Product and Basket Analysis: Mix, ABC, Pricing, and Promotions

20 Operations Analytics: Staffing, Labor Cost, and Store Performance

PART V — VISUALIZING AND REPORTING

21 Visualization Fundamentals with matplotlib

22 Business Charts That Explain: Choosing, Designing, and Annotating

23 Tables and Formatted Output: Styling, HTML, and Report-Ready Numbers

24 Automated Reporting: Building the Monthly Sales Report

PART VI — PERFORMANCE AND PRODUCTION

25 Performance and Scale: Memory, Lazy Evaluation, and Streaming

26 SQL Meets DataFrames: DuckDB with pandas and Polars

27 Reliable Analysis Code: Functions, Tests, and Reproducibility

28 Capstone: The Quarterly Business Review — and Choosing Between pandas and Polars

APPENDICES

A pandas and Polars Cheat Sheet

B Templates and Reference

C Analysis Production Checklist

D Glossary

PART I

Foundations

The analyst's workflow and where pandas, Polars, and DuckDB fit, a working Python setup with the Cinderwell Coffee project, the fundamentals of pandas and of Polars side by side, and the file formats that every analysis starts and ends with.

1 · Data Analysis in Python
2 · Setting Up Your Workspace
3 · pandas Fundamentals
4 · Polars Fundamentals
5 · Reading and Writing Data

CHAPTER 1 · PART I — FOUNDATIONS

Data Analysis in Python: pandas, Polars, and the Analyst's Workflow

Before we write serious code, we need a clear picture of what an analyst actually does all day, why Python is such a good place to do it, and how two very different DataFrame libraries, pandas and Polars, fit into that work. By the end of this chapter you will know the coffee company whose data fills this book, and you will have seen one real business question answered three ways.

What Data Analysis Really Is

"Data analysis" sounds like a single activity, but in practice it is a chain of small, very concrete jobs. Someone has a question. Somewhere there is data that might answer it. The data is never quite in the shape you need, so you clean it, combine it with other data, and reshape it. Then you compute something: a total, an average, a trend, a comparison. You turn the result into a table or a chart that a busy person can read in thirty seconds. Finally, that person makes a decision, and the decision usually produces a new question.

We will use eight words for these steps throughout the book:

1. **Question.** What do we want to know, and who wants to know it? "How did June go?" is a mood; "What were net sales per store in June 2026, and how do they compare with June 2025?" is a question you can answer.
2. **Data.** Which files, tables, or systems hold the facts? What does one row mean? What period do they cover?
3. **Clean.** Fix what is wrong: inconsistent codes, text that should be numbers, duplicate rows, missing values, dates in two formats.
4. **Transform.** Add the columns you need (net sales, margins), join tables together, filter to the period of interest, reshape from long to wide.
5. **Analyze.** Group, aggregate, compare, rank, and look for patterns and exceptions.
6. **Visualize.** Draw the few charts that make the pattern obvious.
7. **Report.** Package the numbers, charts, and a short written interpretation for the people who asked.
8. **Decide.** Someone acts: changes a schedule, ends a promotion, opens a new store.

Written as a list, this looks like a straight line. In reality it is a loop, and you will go around it many times for a single question. You start cleaning and discover the data does not cover the period you assumed, so you go back to the data step. You draw a chart and notice a strange spike, so you go back

to cleaning and find a batch of duplicated rows. You present a report and the reader asks a sharper follow-up question, and the loop starts again. Good analysts are not people who never loop back; they are people who loop back quickly and calmly, because their work is organized so that rerunning it is cheap.

Reproducible beats heroic

A good analysis can be run again. When the June numbers arrive, you should be able to rerun May's analysis with a different month and get a trustworthy result in minutes, not rebuild it by hand. That idea shapes the whole book: we write small Python scripts instead of clicking through menus, we keep the data files and the code that produces every number, and by Part V we turn a one-off analysis into a program that writes the monthly report on its own.

Why Python for Analysis?

You can analyze data in many tools. Spreadsheets, SQL databases, and business intelligence products all have their place, and you will probably keep using some of them. Python is not a replacement for everything; it is a general-purpose language with an unusually strong set of data libraries, and that combination is what makes it attractive.

Tool	Where it shines	Where it struggles
Spreadsheet	Quick look at small data, ad-hoc what-if, sharing with non-programmers	Large row counts, repeatable steps, tracking what changed, joining many tables
SQL database	Data that already lives in a database, joins and aggregations, many users	Messy files, complex text cleaning, charts, mixing with other code
Python + DataFrames	Repeatable pipelines from raw file to finished report, cleaning, charts, automation	Needs some programming; very large data needs care (Chapter 25)

Three properties make Python especially good for the kind of work in this book. First, **everything is code**, so every step is written down, can be reviewed, and can be rerun. Second, **the ecosystem is broad**: the same language reads CSV and Parquet files, talks to databases, cleans text, draws charts, writes HTML reports, and schedules jobs. Third, **the libraries are free and widely taught**, so the skills transfer between employers and projects.

DataFrames: The Shape of Business Data

Almost every business dataset is a table: rows of things (receipts, customers, shifts) and columns of attributes (date, store, amount). The data structure that represents such a table in memory is called a **DataFrame**. A DataFrame has named columns, each column holds values of one type (integers, decimals, text, dates), and every column has the same number of rows. If you have used a spreadsheet tab with a header row or a SQL table, you already know the idea.

The power of a DataFrame library is that you operate on whole columns at once. Instead of looping over half a million receipts in Python and adding numbers one by one, you write "multiply the quantity column by the price column" and the library does the work in fast compiled code. This style is called **vectorized** code, and it is the single most important habit to build. We will come back to it many times, starting in Chapter 3.

Python has two popular DataFrame libraries for this kind of work, and this book teaches both.

What pandas Is

pandas is the long-established DataFrame library for Python. It grew up alongside NumPy, the numerical array library, and much of the Python data world is built on top of it or expects its objects. If you search for how to do something with tabular data in Python, the first answers will very likely use pandas, and many other libraries (for plotting, statistics, machine learning, and file formats) accept a pandas DataFrame as input.

Four characteristics of pandas matter for how you write code with it:

- **It has an index.** Every pandas DataFrame and Series carries row labels called the index. By default the labels are 0, 1, 2, and so on, but you can make a column such as `store_id` the index and then look rows up by label. The index is useful for time series and alignment, and it surprises almost everyone at first. Chapter 3 explains it carefully.
- **It is eager.** Every line of pandas code runs immediately and produces a result in memory. That makes it easy to inspect intermediate steps, which is wonderful for exploration.
- **It has a huge, mature API.** There is a method for almost everything, and usually several ways to do the same thing. The flexibility is great once you know which way to choose, and part of our job is to show you a consistent, modern subset.
- **It is evolving.** This book targets pandas 3 (examples tested with 3.0.6). pandas 3 changed several long-standing defaults, such as how text columns are stored and how copies behave. Most code in this book also runs on recent pandas 2.x releases, and where pandas 3 behaves differently we say so.

What Polars Is

Polars is a newer DataFrame library with a different design. Its core is written in Rust and it stores data in the **Apache Arrow** columnar memory format, a standard layout in which each column's values sit next to each other in memory. Polars is used from Python through a Python API, and this book targets Polars 1.x (tested with 1.44).

The design differences show up directly in the code you write:

- **No index.** A Polars DataFrame is just columns and row positions. You never set or reset an index; if you want to look something up by store id, you filter on the `store_id` column.

- **Expressions.** Instead of pulling a column out and operating on it, you describe the computation with an expression such as `pl.col("quantity") * pl.col("unit_price")` and hand that description to a method like `select`, `with_columns`, or `filter`. Expressions are composable, and they let Polars see your whole computation before running it.
- **Eager and lazy.** Polars can run eagerly like pandas, or lazily: you build a query plan with a `LazyFrame`, Polars optimizes it (for example, reading only the columns and rows the query needs), and nothing runs until you call `collect()`. Chapter 4 shows this in detail.
- **Parallel by default.** Polars spreads work across your CPU cores without you asking.

Why Polars can be fast

You may have heard that Polars is "fast." We will not quote a speed-up factor, because the honest answer is that it depends on the data, the operation, the machine, and how well each library's code is written. What we can explain is *why* Polars is often quick on analytical workloads: columnar Arrow memory is efficient to scan; work is split across cores; the lazy engine can skip columns and rows it does not need and combine steps; and because there is no index, there is less bookkeeping to maintain. When we time anything in this book, it is measured on our machine, labeled that way, and presented as an illustration, not as a benchmark.

> **WARNING**
>
> Be skeptical of any speed claim with a single big number in it, including claims about pandas. A fair comparison needs the same data, the same result, warm and cold runs, and idiomatic code in both libraries. Most "X is 50 times faster" posts fail at least one of those tests. When performance really matters to you, measure your own workload (Chapter 25 shows how).

When Each Library Shines

This book does not crown a winner. Both libraries are excellent, and many teams use both. Chapter 28 ends the book with a detailed decision guide; for now, here is the short, honest version.

Reach for pandas when your team already knows it; when you need to hand DataFrames to libraries that expect pandas (many plotting, statistics, and machine learning tools do); when the index genuinely helps, as with some time series and label-aligned arithmetic; or when you are following a tutorial, course, or colleague's code written in pandas. Its ecosystem and the sheer amount of documentation, examples, and answered questions are real advantages.

Reach for Polars when your data is large relative to your machine's memory; when you want a query optimizer to read only what it needs from Parquet files; when you like a consistent, expression-based API with fewer ways to do the same thing; or when you want parallel execution without extra work. Its stricter typing also catches some mistakes earlier.

And because both libraries speak Apache Arrow, moving data between them is cheap: `to_pandas()` and `pl.from_pandas()` convert in one call. Mixing them in one project is normal, and you will see us do it when it makes a task simpler, for example plotting Polars results with matplotlib through pandas.

> **TIP**
>
> If you already use pandas and are curious about Polars, read each chapter's pandas code first, then the Polars version, and pay attention to the paragraph after them that explains the differences. If you are new to both, read both versions anyway: seeing the same idea expressed two ways is one of the fastest routes to understanding what the code is really doing.

DuckDB: The SQL Companion

A third tool appears in Part VI: **DuckDB**, an analytical SQL database that runs inside your Python process. There is no server to install; you import it, and you can run SQL directly against Parquet files on disk or against pandas and Polars DataFrames held in variables. If you come from SQL, DuckDB lets you use that knowledge next to DataFrames, and some questions are simply easier to express as a query. The book targets DuckDB 1.5. Chapter 26 is devoted to it, and you will get a three-line taste at the end of this chapter.

Meet Cinderwell Coffee

Learning data analysis with toy tables of five rows hides the real problems: type surprises, duplicates, missing values, dates that do not parse, and results that look plausible but are wrong. So this book uses one realistic dataset from start to finish.

Cinderwell Coffee is a fictional chain of six coffee shops in the US Pacific Northwest and Idaho, with its head office in Seattle. Every row of its data is synthetic, generated by a script you will run in Chapter 2, but the data was built to behave like the real thing: busy mornings, quiet Sundays at the commuter kiosk, a December peak, a price rise, promotions, a new store opening, and a loyalty program with all the messiness of real sign-up forms. You are Cinderwell's new data analyst.

ch01_stores.py

```
import pandas as pd

pd.set_option("display.width", 88)
stores = pd.read_csv("data/stores.csv")
print(stores[["store_id", "store_name", "city", "state", "format", "opened_on", "seats"]])
```

Output

```
  store_id         store_name      city state      format   opened_on  seats
0     ST01        Pike Street   Seattle    WA        cafe  2019-04-01     48
1     ST02     Pearl District  Portland    OR        cafe  2020-06-15     40
2     ST03      Union Station    Tacoma    WA       kiosk  2021-03-01      6
```

```
 3     ST04         Riverfront   Spokane    WA  drive_thru  2022-08-20     12
 4     ST05  Capitol Boulevard     Boise    ID        cafe  2023-05-10     32
 5     ST06      Campus Corner    Eugene    OR        cafe  2025-09-15     36
```

A few facts about these stores will matter again and again. ST03, Union Station in Tacoma, is a kiosk with six seats that serves commuters from 6:00 to 15:00, so its weekends are quiet. ST04 in Spokane is a drive-thru. ST05 in Boise is the only store in the Mountain time zone (America/Boise); all others use Pacific time. ST06, Campus Corner in Eugene, opened on 2025-09-15, partway through our data, which makes it both an interesting growth story and a trap for anyone comparing stores naively. The cafes and the drive-thru sell from 6:00 to 19:00.

The data files

The data covers 18 months, from 2025-01-01 to 2026-06-30. Here is what lives in the project's `data/` folder and what each file is for:

File	Rows	What one row is
`stores.csv`	6	A store
`products.csv`	40	A menu item with current price and cost
`staff.csv`	62	An employee with role and hourly rate
`members_raw.csv`	9,037	A loyalty sign-up (messy)
`shifts.csv`	12,288	One staff shift with clock times
`promotions.json`	6	A promotion (nested JSON)
`transactions.parquet`	508,411	A receipt (one sale)
`transaction_items.parquet`	814,764	A line on a receipt
`raw/pos_export_2026-06.csv`	50,238	A line from a messy June export

Two of these files are deliberately messy, the way real exports are. The loyalty member list has duplicate rows, people who registered twice, store codes typed in lower case or padded with spaces, dates in two formats, and eight different ways of writing "yes" and "no." The June point-of-sale export has prices stored as text with dollar signs, several spellings of "no member," a handful of receipts from a training till, and lines duplicated when the export retried a page. Part II cleans both files, and Chapter 10 proves the cleaned export matches the main sales data to the cent.

The sales data itself is split the way most point-of-sale systems split it: one table of **receipts** (`transactions`, with the store, timestamp, channel, payment type, staff member, and optional loyalty member and promo code) and one table of **line items** (`transaction_items`, with the product, quantity, the price actually charged, and any line discount). Product names and categories live in `products.csv`. Joining these tables correctly is the subject of Chapter 13.

Three colleagues and their questions

Data analysis is always for someone. At Cinderwell you will work most closely with three people, and their questions drive many of the examples in this book:

- **Dana**, Director of Operations, runs the stores. She wants to know how each store performed last month, when the rush hours are, whether staffing matches demand, and she wants a monthly report she does not have to assemble by hand. ("Which store had the best June, and is that because it is bigger or because it is better?")
- **Marco**, Marketing & Loyalty Lead, owns the loyalty program and promotions. He asks how many members come back, which promotions were used, and which customers are slipping away. ("Of the people who joined in January, how many were still buying three months later?")
- **Ingrid**, Finance Manager, cares about margins, labor cost, and the quarterly business review. She wants numbers that reconcile and definitions that do not change between reports. ("What was our gross margin in Q2, and did the March price rise hurt Latte sales?")

The words we use for money

Business analysis goes wrong quickly when two people use the same word for different numbers. So we fix the definitions now and use them everywhere:

- **Gross sales** of a line = quantity × unit price. **Net sales** = gross sales − discount. When we say "sales" without a qualifier, we mean net sales.
- **COGS** (cost of goods sold) = quantity × unit cost. **Gross profit** = net sales − COGS. **Gross margin** = gross profit / net sales.
- A **receipt** is one transaction. **Average ticket** = net sales / number of receipts.

Later chapters add labor measures such as labor percentage and sales per labor hour, and customer measures such as active members and retention. Each is defined once, where it first appears, and then used consistently.

A First Taste: June Net Sales Three Ways

Let us answer one real question right away: *what were Cinderwell's total net sales in June 2026?* You do not need to understand every detail of the code yet; the point is to see the shape of a pandas solution, a Polars solution, and a SQL solution side by side. You will be able to run all three after you set up the project in Chapter 2.

The logic is the same in every version. Net sales live on the line items, but the date lives on the receipt. So we keep the June receipts, join them to their line items, compute quantity × unit price − discount for each line, and add everything up. Because the data ends on 2026-06-30, "on or after June 1" is the same as "in June."

ch01_first_taste.py (pandas)

```
import pandas as pd

txn = pd.read_parquet("data/transactions.parquet")
items = pd.read_parquet("data/transaction_items.parquet")
june_txn = txn[txn["ts"] >= "2026-06-01"]
june = items.merge(june_txn, on="transaction_id")
net = (june["quantity"] * june["unit_price"] - june["discount"]).sum()

print(f"June 2026 net sales: ${net:,.2f}")
print(f"Receipts: {june['transaction_id'].nunique():,}  Lines: {len(june):,}")
```

Output

```
June 2026 net sales: $272,824.50
Receipts: 31,370  Lines: 50,189
```

Read it top to bottom. `pd.read_parquet` loads each file into a DataFrame. The expression `txn["ts"] >= "2026-06-01"` produces a column of True/False values, one per receipt, and indexing `txn` with it keeps only the True rows. `merge` joins line items to those receipts on their shared `transaction_id`, keeping only lines whose receipt is in June. The next line multiplies and subtracts whole columns at once (vectorized) and sums the result. The f-string `{net:,.2f}` formats the number with a thousands separator and two decimals.

ch01_first_taste_pl.py (Polars)

```
import polars as pl

txn = pl.read_parquet("data/transactions.parquet")
items = pl.read_parquet("data/transaction_items.parquet")
june_txn = txn.filter(pl.col("ts") >= pl.datetime(2026, 6, 1))
june = items.join(june_txn, on="transaction_id")
net = june.select(pl.col("quantity") * pl.col("unit_price") - pl.col("discount")).sum()

print(f"June 2026 net sales: ${net.item():,.2f}")
print(f"Receipts: {june['transaction_id'].n_unique():,}  Lines: {june.height:,}")
```

Output

```
June 2026 net sales: $272,824.50
Receipts: 31,370  Lines: 50,189
```

The structure is identical, and the answer matches to the cent, but the vocabulary differs in instructive ways. Polars uses `filter` with an expression, `pl.col("ts") >= pl.datetime(2026, 6, 1)`, rather than indexing with a True/False column. It uses `join` where pandas uses `merge`. The arithmetic is written as an expression inside `select`, which returns a one-row, one-column DataFrame; `.item()` pulls the single value out. Polars also compares a datetime column with a real datetime value rather than a string, part of its generally stricter attitude to types.

And here is the same question in SQL, run by DuckDB directly against the Parquet files without loading them into a DataFrame first:

ch01_first_taste_duckdb.py

```
import duckdb

net = duckdb.sql("""
    SELECT sum(i.quantity * i.unit_price - i.discount) AS net
    FROM 'data/transaction_items.parquet' AS i
    JOIN 'data/transactions.parquet' AS t USING (transaction_id)
    WHERE t.ts >= '2026-06-01'
""").fetchone()[0]
print(f"June 2026 net sales: ${net:,.2f}")
```

Output

```
June 2026 net sales: $272,824.50
```

Three tools, one number: **$272,824.50** from 31,370 receipts and 50,189 line items. That agreement is not a coincidence or a curiosity; it is a habit we want you to form. Whenever an important number can be computed two independent ways, compute it both ways and check that they agree. You will meet this June total again in Chapter 10, where the messy June export, once cleaned, must reproduce exactly this figure.

> **NOTE**
>
> The first-taste scripts compute net sales line by line without rounding each line first. The project's shared module, which you build from Chapter 5 onward, rounds line amounts to cents, which is how a till would print them. For these data the totals are identical, but money and floating-point numbers have a few traps of their own; Chapter 11 covers them properly.

How to Read the Side-by-Side Code

Most chapters follow the same rhythm, and knowing it will help you move faster:

1. A short explanation of the idea and why it matters for the business question.
2. A **pandas** script, with a caption naming the file, such as `ch12_store_summary.py (pandas)`.
3. Its real **Output**, exactly as the script printed it.
4. The **Polars** version, such as `ch12_store_summary_pl.py (Polars)`.
5. Its real output.
6. A paragraph explaining the differences: eager versus lazy, index versus no index, methods versus expressions.

Every output in this book was produced by running the code shown, with the library versions listed in Chapter 2. We did not type outputs by hand or tidy the numbers. Occasionally we trim long outputs with a `...` line, or set a display option so a table fits the page; when we do, the option is in the code. If you run the same script with the same versions, you should see the same output. With other versions you may see small differences in formatting, column widths, or the order of rows in places where order is not guaranteed; we point out those places when they matter.

Scripts are named `chNN_topic.py` after the chapter that introduces them, and they all live in one project folder, `cinderwell-analytics/`, which you create in Chapter 2. Scripts assume they are run from that folder, so paths such as `"data/transactions.parquet"` are relative to it.

Throughout the book you will also see three kinds of boxes. **Tips** are shortcuts and good habits. **Notes** give background or point to a later chapter. **Warnings** flag mistakes that produce wrong numbers silently, which is the most dangerous kind of mistake in analysis because nothing crashes to warn you.

How This Book Is Organized

The book has six parts, and each builds on the previous ones.

- **Part I — Foundations (Chapters 1–5).** This overview, setting up Python and the project, the core concepts of pandas and of Polars, and reading and writing every file format the project uses.
- **Part II — Cleaning and Preparing Data (Chapters 6–10).** Selecting, filtering, and sorting; cleaning the messy POS export and member list; text and categories; dates, times, and time zones; and validating data quality, ending with a reconciliation that proves the cleaned data is right.
- **Part III — Transforming and Combining Data (Chapters 11–15).** New columns and conditions, grouping and aggregation, joins, reshaping and nested data, and window functions such as rankings and running totals.
- **Part IV — Analyzing Business Data (Chapters 16–20).** Time series, descriptive statistics, customer cohorts and segmentation, product and basket analysis, and operations analytics with staffing and labor cost.
- **Part V — Visualizing and Reporting (Chapters 21–24).** matplotlib fundamentals, business charts that explain, report-ready tables, and an automated monthly sales report.
- **Part VI — Performance and Production (Chapters 25–28).** Memory and lazy evaluation at scale, DuckDB and SQL, tests and reproducibility, and a capstone quarterly business review that ends with an honest guide to choosing between pandas and Polars.

The appendices collect a pandas and Polars cheat sheet, templates and reference material (including the finished `cinderwell.py` module), a production checklist for analysis work, and a glossary.

> **TIP**
>
> Keep a short "analysis log" as you work through the book: a plain text file in which you write down each question, the script that answers it, and the headline number. It takes seconds per entry, and it is exactly the habit that makes real analysis work easy to revisit months later.

> **WARNING**
>
> The people, stores, and customers in this book are fictional. Every email address uses the reserved `example.com` domain. When you apply these techniques to real customer data, remember that real data is about real people: analyze ids rather than names where you can, keep only the columns you need, and follow your organization's privacy rules. Chapter 18 returns to this.

Try It Yourself

1. Pick a question you have recently been asked at work or in study. Rewrite it so it names a measure, a period, and a comparison, the way "How did June go?" became "net sales per store in June 2026 versus June 2025."
2. For each of Dana's, Marco's, and Ingrid's example questions above, list which of the nine data files you would need to answer it. (Hint: Ingrid's margin question needs a file that is not about sales at all.)
3. Using the definitions in this chapter, compute by hand the gross sales, net sales, and average ticket for a day with two receipts: one with a $5.00 Latte and a $3.50 croissant, and one with two $2.75 House Drips and a $0.55 discount.
4. Look at the store table and write down two reasons why simply comparing total net sales of ST01 and ST06 over the whole 18 months would be unfair. You will check your reasons against Chapter 17.
5. After you finish Chapter 2, run the three first-taste scripts and confirm that each prints $272,824.50. Then change the date filter to `"2026-05-01"` and explain why the result is not May's sales.