Fundamentals of Data Science

From Raw Data to Responsible Decisions — Volume 1: Main Text

Mohammad Sabokrou

Table of Contents

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

Preface

Data science is not merely the ability to run an algorithm. It is the disciplined practice of turning imperfect evidence into useful, honest, and appropriately limited conclusions. This

book introduces that practice from the beginning. It assumes only basic Python familiarity and explains statistical and machine-learning ideas in plain language before introducing notation or code.

The text is organized around a recurring case: a university wishes to understand student learning and improve support. The case is intentionally treated with care. Grades, attendance, study mode, and financial circumstances can be sensitive. Students therefore learn that a technically possible analysis is not automatically a legitimate one.

Chapters, weeks, and labs

Fifteen chapters run across twelve teaching weeks. Two weeks carry two chapters each, so the mapping is not one-to-one:

Week	Chapters	Lab
1	1, 2	1
2	3	2
3	4	3
4	5	4
5	6	5
6	7	6
7	8	7
8	9	8
9	10	9
10	11, 12	10
11	13	11
12	14, 15	12

How to use this book

Read the concept, run the code, change one thing and predict what will happen, then explain the result in your own words. If code runs but you cannot explain what the output means, the analysis is not finished.

Work the faded variants with a pen before running anything. The point of a worked example is not that you have read a solution — it is that you can produce the next one.

Academic integrity and responsible use

All explanatory prose, examples, and exercises in this edition were newly written. The topics were informed by established university texts and open educational resources listed in the references. Cite datasets, code you did not write, and ideas you adapt. If you use a generative AI assistant, protect confidential data, verify every output, acknowledge substantial assistance, and never present unexamined output as your own reasoning.

What you will be able to do

By the end of this course you will be able to:

CLO1. Explain fundamental concepts in data science, including data collection, data preprocessing, exploratory data analysis, and basic statistical methods.

CLO2. Analyse different data types — numerical, categorical, time-series, and text — and apply appropriate cleaning and transformation techniques to prepare data for analysis.

CLO3. Develop and apply basic data analytics and machine-learning techniques, both supervised and unsupervised, to draw meaningful insights and support data-driven decision-making.

CLO4. Communicate data-driven findings effectively through visualization, dashboards, and reports, demonstrating awareness of ethical considerations, data privacy, and professional standards.

Start Here — Your Roadmap

Data science is learned by repeated cycles of questioning, coding, checking, explaining, and improving. You are not expected to become a statistician, software engineer, and domain expert all at once. The goal is a reliable foundation and a clear path for continued growth.

Phase	Chapters	What you build
Foundation	1–2	Python, tables, files, reproducible notebooks, provenance
Data work	3–5	Description, quality, cleaning, transformation
Reasoning	6–9	Exploration, visualization, probability, inference
Modelling	10–13	Baselines, regression, classification, clustering
Delivery	14–15	Time series, text, dashboards, responsible reporting

A realistic promise. By the end, you should be able to complete a small end-to-end project responsibly. Expertise comes later through many projects, feedback, mathematics, and domain experience.

The course dataset

Everything in this book runs against one synthetic university dataset, in four linked tables.

File	Rows	One row is
students.csv	1,200	one student
enrolments.csv	4,665	one student-course registration
activity.csv	16,165	one student-week of platform activity

File	Rows	One row is
outcomes.csv	1,200	one student, with weeks 1–4 features and the withdrawal outcome

Plus feedback.csv (free-text comments), platform_daily.csv (a daily time series), and students_dirty.csv (a deliberately damaged copy for the cleaning chapter).

The data are synthetic and released under CC0. No real student record appears anywhere in this course — which is consistent with the argument the book makes about analysing data on people.

```
import pandas as pd

students = pd.read_csv("data/students.csv")
enrolments = pd.read_csv("data/enrolments.csv")
activity = pd.read_csv("data/activity.csv")
outcomes = pd.read_csv("data/outcomes.csv")

print(students.shape, enrolments.shape, activity.shape, outcomes.shape)
# (1200, 7) (4665, 5) (16165, 6) (1200, 7)
```

The Data Scientist's Toolbox

Python is the language used throughout. **pandas** handles tables. **NumPy** handles arrays and simulation. **matplotlib** and **seaborn** draw figures. **scikit-learn** provides models and the pipeline machinery that keeps evaluation honest. **SciPy** supplies statistical tests.

You will work in **Google Colab**, a notebook that runs in your browser on Google's computers. Nothing is installed on your laptop, every student has an identical environment, and all the packages above are already there. You need only a Google account.

Go to **https://colab.research.google.com**, click **New notebook**, and you are ready.

One package is missing and you will need it in Week 12 only:

```
!pip install streamlit
```

One thing to know about Colab before you start. Files you create live on a temporary machine that is wiped after about 90 minutes of inactivity. Keep the course data in your Google Drive, and save your notebook regularly with **File → Save a copy in Drive**. Work that is not saved is gone, and nothing can recover it.

Load the data at the start of every notebook. Put the data folder in your Drive once, and this works all term:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
DATA = "/content/drive/MyDrive/fds/data/"
students = pd.read_csv(DATA + "students.csv")
```

Also record your environment at the top of every notebook. When a result cannot be reproduced six weeks later, the version numbers are usually the first clue.

```
import sys, platform
import numpy as np, pandas as pd, sklearn

print("Python:", sys.version.split()[0])
print("Platform:", platform.platform())
print("NumPy:", np.__version__, "| pandas:", pd.__version__, "| sklearn:",
sklearn.__version__)
```

Using AI assistants responsibly

You will use AI coding assistants. Three rules make that defensible rather than corrosive:

1. **Never paste confidential or personal data into a third-party assistant.** This includes real student records, which is one reason this course uses synthetic data.
2. **Verify every output.** An assistant that produces plausible-looking pandas code will occasionally produce a silent logic error — a wrong join key, a leaked test set — that runs perfectly and gives a wrong answer.
3. **Acknowledge substantial assistance.** If an assistant wrote a function, say so in a comment. The professional standard is not “did you get help” but “can you explain and defend every line you submitted”.

The explain-it test applies to assistant output exactly as it applies to your own: for every important line, what enters, what transformation occurs, what leaves, and what could make it wrong?

Chapter 1 — Thinking Like a Data Scientist

> *A useful analysis begins with a question, not with an algorithm.*

After this chapter you can: explain the data-science lifecycle; distinguish description, inference, prediction, and causal claims; frame a question that data can genuinely address; and set up a reproducible notebook.

1.1 What data science is

Data science combines computation, statistics, and domain knowledge to learn from data. Computation makes large or complicated operations repeatable; statistics helps quantify uncertainty; domain knowledge tells us whether variables and conclusions make sense. Removing any one of these elements weakens the result. A fast model can still answer the wrong question, and a correct calculation can still be irrelevant to the decision.

This is why a simple, carefully documented comparison can be better science than a complicated model trained on poorly understood data.

1.2 From a vague concern to an analytical question

"Students are struggling" is a concern, not yet a data question. A useful question identifies a population, variables, time period, and purpose. For example: *"Among first-year students enrolled in 2026, which weeks 1–4 engagement indicators are associated with withdrawing before the end of term?"*

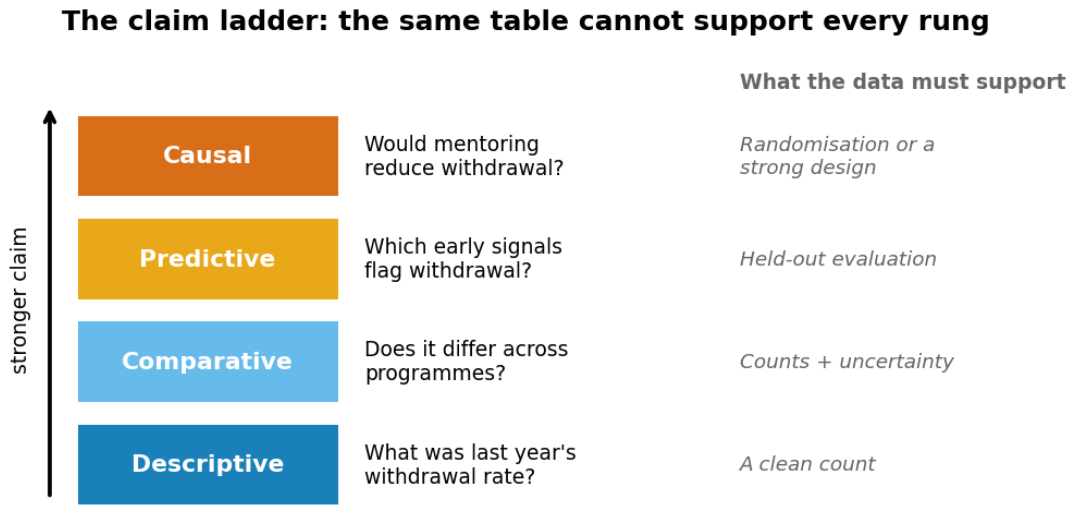
The word *associated* is deliberate. Observational data alone rarely establishes causation.

1.3 The claim ladder

Consider a university that wants to improve first-year retention. Four questions look similar and are not:

- *"What was last year's withdrawal rate?"* — **descriptive**
- *"Does it differ across programmes?"* — **comparative**; needs uncertainty
- *"Which early indicators predict withdrawal?"* — **predictive**; needs held-out evaluation
- *"Would a mentoring programme reduce withdrawal?"* — **causal**; normally needs randomisation

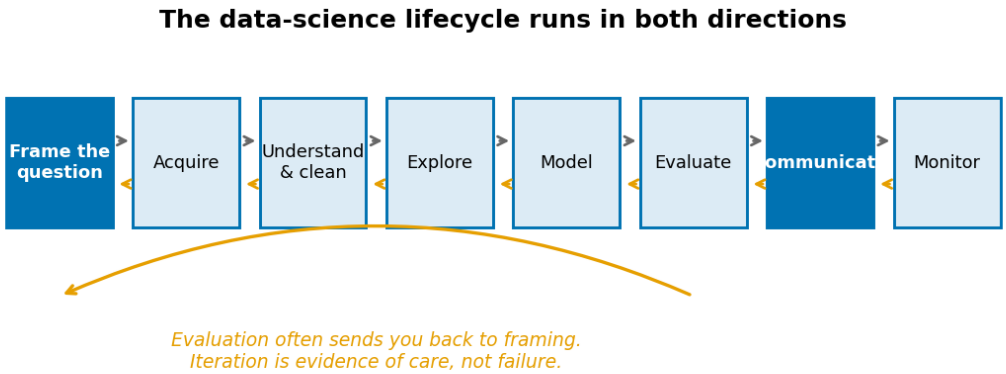
The same table cannot automatically support all four. Before opening Python, write the exact question and label the intended claim. This small habit prevents a common failure: choosing an attractive method first and quietly changing the question to match the output.



The claim ladder. Each rung requires evidence the rung below does not.

1.4 The lifecycle

A practical lifecycle contains framing, acquisition, understanding, cleaning, exploration, modelling, evaluation, communication, and monitoring. The arrows run in both directions. Exploration may reveal that the target was poorly defined; evaluation may reveal that more representative data are needed. Iteration is evidence of care, not failure.



The data-science lifecycle. In practice you will traverse it several times.

1.5 Objects, variables, and levels of measurement

A dataset is a representation of a process, not the process itself. Rows identify **units of observation**; columns describe attributes. Variables also have **measurement levels**:

Level	Meaning	Example in our data
Nominal	Categories, no order	programme, study_mode
Ordinal	Ordered, unequal gaps	midterm_satisfaction (1–5)
Discrete	Countable quantities	logins, submissions
Continuous	Any value in a range	entry_score, minutes_on_platform

A variable's *meaning* matters more than its storage type. student_id is stored as an integer and is nominal. If you can compute a meaningful mean of it, it is not an identifier.

1.6 Reproducible work

A reproducible analysis records data sources, software versions, transformations, random seeds, assumptions, and decisions. A notebook should function like a transparent argument: Markdown explains the purpose and decisions, code performs transformations, output supplies evidence, and interpretation connects evidence to the question.

A reader should be able to restart the notebook, run it top to bottom, and obtain the same result. In Colab that is **Runtime → Restart and run all**, and it is the single most useful check you can run on your own work. Reproducibility does not guarantee correctness — a

reproducible mistake remains a mistake — but it allows mistakes to be found. An irreproducible result cannot be responsibly audited.

1.7 Responsible framing before analysis

Questions about people can affect opportunity, reputation, and access to support. Before collecting variables, ask who benefits, who bears risk, whether participation is voluntary, and whether a less intrusive analysis could meet the purpose.

A prediction of academic difficulty should trigger helpful support, not automatic punishment. A variable such as first_generation may be useful for **auditing unequal outcomes** yet inappropriate as a **decision feature**. This distinction runs through the whole course and is graded in the rubric.

Ethical judgement cannot be postponed until the final report. It shapes the question, the data, the evaluation, and the action from the beginning.

Worked example 1.1 — Turning a concern into a question

The Dean says: *"Students seem disengaged this year. Can you look into it?"*

That sentence contains no population, no variable, no time period, and no decision. Here is the repair, in four passes.

Pass 1 — Who? "Students" could mean all 1,200 enrolled, or first-years only, or one programme. The Dean's concern is about *this year*, so restrict to the 2026 entry cohort. Check the size first, because a question about 40 people is a different question from one about 400:

```
cohort = students[students.entry_cohort == 2026]
print(len(cohort))        # 431
```

Pass 2 — What is "disengaged"? It is not a column. We must choose an **operational definition**. Three candidates in activity.csv: logins, submissions, minutes_on_platform. These are not interchangeable. A student who reads offline and submits everything has low minutes and is not disengaged. A student who leaves a tab open has high minutes and may be. Submissions are the closest to something the university actually cares about, so define engagement as *submissions in weeks 1–4*, and state that choice explicitly.

Pass 3 — Compared with what? "Seems disengaged" implies a comparison the Dean has not stated. Against last year's cohort? Against a target? Without a comparison there is no finding, only a number.

Pass 4 — What decision follows? The Dean will do something differently depending on the answer. If the answer is "engagement is down 15% versus 2025", the action is probably an intervention. If it is "engagement is flat but concentrated in two programmes", the action is targeted. Naming the decision tells you which comparison matters.

The repaired question:

> Among students in the 2026 entry cohort, is mean assessment submission over teaching weeks 1–4 lower than for the 2025 cohort at the same point, and does any difference concentrate in particular programmes?

This is **comparative**, not causal. Even a large difference would not establish that anything the university did caused it — the cohorts differ in entry scores, programme mix, and much else.

Limitation to state up front: submissions measure compliance with assessment deadlines, not learning. A cohort could become more engaged and submit less if the assessment schedule changed. Before reporting any difference, check whether it did.

> **Faded variant.** The Careers Office says: *"Our graduates aren't getting good jobs."* Work the same four passes. Write out the population, the operational definition of "good job", the comparison, and the decision. Then label your final question descriptive, comparative, predictive, or causal — and say what evidence the next rung up would require.

Worked example 1.2 — One row, one meaning

A colleague sends you this and asks you to check it:

```
enrolments = pd.read_csv("data/enrolments.csv")
print("Number of students:", len(enrolments))     # 4665
```

The number is wrong, and it is wrong in a way that will not raise an error.

Establish what a row is. Count how many times each student appears:

```
print(enrolments.student_id.value_counts().head())
# 100626    4
# 100001    4
# 100394    4
# ...
print("rows:", len(enrolments))                   # 4665
print("distinct students:", enrolments.student_id.nunique())  # 1200
```

One row is a **student-course registration**, not a student. Each student registers for three or four courses, so len() overcounts people by a factor of about 3.9.

Why it matters numerically. Suppose we ask what proportion of students are part-time.

```
merged = enrolments.merge(students[["student_id", "study_mode"]], on="student_id")

# Wrong: one vote per registration
print((merged.study_mode == "Part-time").mean())          # 0.119
```

```
# Right: one vote per student
print((students.study_mode == "Part-time").mean())        # 0.144
```

The answers differ — 11.9% versus 14.4% — and the wrong one is *systematically* wrong, not randomly wrong. Part-time students defer the hard quantitative courses, so they hold fewer registrations each, so registration-level counting under-weights them. Anyone checking your arithmetic will find it correct. Only checking your *unit* finds the error.

The general rule. Before any count, mean, or groupby, finish this sentence out loud: *"One row of this table is one _____."* If the sentence does not match the noun in your question, aggregate first.

```
per_student = (enrolments.groupby("student_id")
                  .agg(n_courses=("course_code", "count"),
                       mean_grade=("final_grade", "mean"))
                  .reset_index())
print(len(per_student))      # 1200 — now one row is one student
```

Limitation: collapsing to one row per student discards within-student variation. A student averaging 62 across four courses is not the same as one scoring 90, 90, 34, 34. Report the spread, or keep the long table for questions that need it.

> **Faded variant.** Compute mean logins per student from activity.csv in two ways: activity.logins.mean() and a per-student aggregate first. Predict *before running* whether they will differ, and if so in which direction. Then explain the result in terms of what one row means. (Hint: check whether every student has the same number of weeks.)

Worked example 1.3 — Reading a claim off the ladder

A draft report contains this sentence:

> "Students who log in more often get better grades, so the university should require weekly logins."

There are two claims here and one of them is unsupported. Separate them.

Claim A (observed): login frequency and grades are positively associated. We can check this directly.

```
per_student = (activity.groupby("student_id")
                  .logins.sum().rename("total_logins").reset_index())
grades = enrolments.groupby("student_id").final_grade.mean().rename("mean_grade")
d = per_student.merge(grades, on="student_id")

print(d[["total_logins", "mean_grade"]].corr().round(3))
```

```
#              total_logins  mean_grade
# total_logins        1.000       0.681
# mean_grade          0.681       1.000
```

r = 0.68. Real, substantial, and **descriptive**. It sits on rung one.

Claim B (asserted): requiring logins would raise grades. That is **causal** — rung four. Nothing in the calculation above supports it.

Why the leap fails. At least three explanations fit r = 0.68 equally well:

1. *Logging in causes learning.* Requiring logins would help.
2. *Reverse direction.* Students who understand the material find the platform rewarding and return to it. Requiring logins changes nothing.
3. *A common cause.* Motivation, prior preparation, or free time drive both. This is the confounding case, and in our data it is demonstrably present: entry_score correlates with both logins and grades.

```
d = d.merge(students[["student_id", "entry_score"]], on="student_id")
print(d.corr(numeric_only=True).round(3).loc["entry_score"])
# total_logins    0.402
# mean_grade      0.622
# entry_score     1.000
```

Entry score predicts both. A student's preparation before arriving explains part of what looks like a login effect.

What would support Claim B. Randomly assign a login requirement to some students and not others, then compare. Absent that, an honest report stops at rung one and says so:

> *Total logins and mean grade are positively associated (r = 0.68, n = 1,200). Entry score is associated with both, so this figure should not be read as an effect of logging in. Establishing whether a login requirement changes grades would require an experiment.*

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

Responsible-use note. A mandatory-login policy justified by this correlation would fall hardest on students with jobs or caring responsibilities — who log in less for reasons unrelated to engagement. A weak causal claim is not merely imprecise; it selects who bears the cost of being wrong.

> **Faded variant.** Rewrite the sentence *"Part-time students pass at a higher rate, so part-time study improves outcomes"* to the highest rung the data actually support. Compute the two pass rates first. You will meet this specific example again in Chapter 6, where it turns out to be worse than merely overstated.

In code: your first reproducible inspection

```
import sys, platform
import pandas as pd, numpy as np

# 1. Record the environment. Do this in every notebook.
print("Python", sys.version.split()[0], "| pandas", pd.__version__)

# 2. Load, and immediately establish shape and meaning.
students = pd.read_csv("data/students.csv")
print("shape:", students.shape)
print(students.dtypes)

# 3. Check the identifier really identifies.
assert students.student_id.is_unique, "student_id is not unique!"

# 4. Look at real rows, not just summaries.
print(students.head(3).to_string())

# 5. Summarise numeric columns — but note what is missing from this view.
print(students.describe().round(1).to_string())

# 6. Categorical columns need a different summary.
for col in ["programme", "study_mode", "entry_cohort"]:
    print(f"\n{col}:")
    print(students[col].value_counts().to_string())
```

describe() will happily report the mean of student_id. That number is meaningless and its presence in the output is a useful reminder: the software does not know what your columns mean, and it will never warn you.

Common mistakes

Mistake	Why it happens	How to catch it
Counting rows as people	len() is easy and always returns something	Say “one row is one ___” out loud before every count
Averaging an identifier	describe() includes it automatically	Set IDs to str on load, or drop them before describing
Claiming causation from correlation	The causal sentence is the interesting one	Label every claim’s rung before writing it down
Analysing before defining the target	“Disengaged” feels like it means something	Write the operational definition in Markdown first
Notebook works for you,	Cells were run out of	**Runtime → Restart and run**

Mistake	Why it happens	How to catch it
fails for everyone else	order	**all** before every submission

The explain-it test. For every important line: What enters? What transformation occurs? What leaves? What could make it wrong? If you cannot answer, reduce the input, inspect intermediate values, and test the smallest case.

Chapter summary

Data science connects computation, statistics, and domain knowledge. Good questions specify population, variables, period, purpose, and the decision that follows. Claims form a ladder — descriptive, comparative, predictive, causal — and each rung demands evidence the one below does not. A row's meaning determines which summaries are valid. Reproducibility makes an analysis auditable, and responsible framing must come before, not after, the analysis.

Check your understanding

1. Why is "use machine learning to improve education" not yet a good analytical question?

2. Give one example each of a nominal, ordinal, discrete, and continuous variable from the course dataset.

3. What would another analyst need in order to reproduce your work?

4. A colleague reports that "the average student ID is 100,600." What went wrong, and what does it tell you about their process?

> **Model answers are in the Reference volume, Part 1.** Write your own answer first — comparing a finished attempt against a model answer teaches far more than reading the model answer cold.

Lab work — Lab 1: A transparent first inspection

> The full two-hour version of this lab, with timings, is **Lab 1** in the Lab Manual (shared with Chapter 2).

Deliverable: one notebook plus a 150-word reflection.

1. Set up your environment and print the version block.
2. Load all four tables. For each, state in Markdown what one row is, and verify your claim in code.
3. For students.csv, verify that student_id is unique. For activity.csv, verify that every student has exactly 14 weeks.
4. Write one descriptive and one predictive question this dataset could answer.
5. Write one question it **cannot** answer, and explain what evidence would be needed.
6. Use **Runtime → Restart and run all**, and confirm every output is identical.

What you will create: one notebook, saved to your Drive, containing a short data-source note and a reflection naming one limitation you would flag to a decision-maker.

Chapter 2 — Collecting and Understanding Data

> *Data are produced by people, systems, and measurement choices; they are never simply "found."*

After this chapter you can: compare sources and sampling methods; read common formats; build a data dictionary; and assess provenance, consent, and representativeness.

2.1 Where data come from

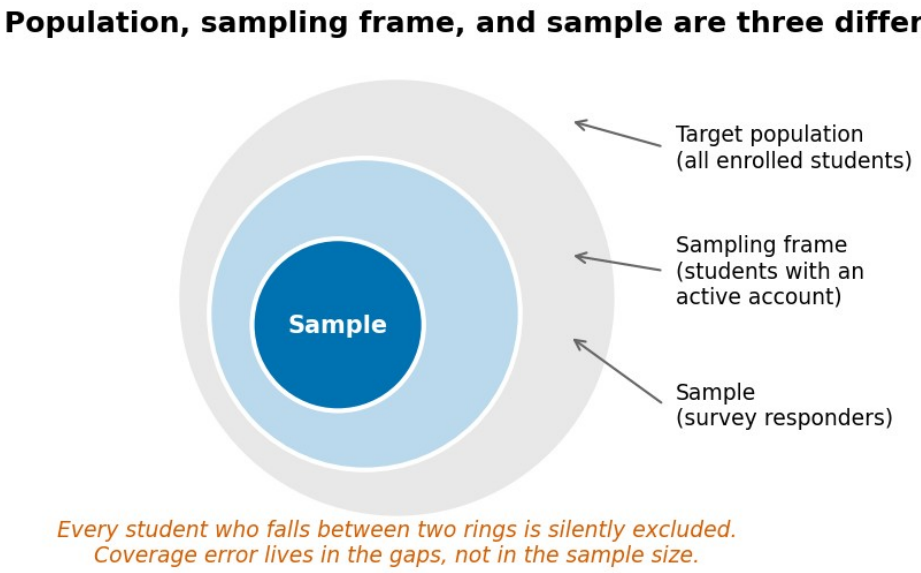
Data arrive from files (CSV, JSON, Excel), from APIs, from databases, from sensors, from surveys, and from the exhaust of systems built for other purposes. That last category — **administrative** or **found** data — is where most institutional data science happens, and it carries a specific hazard: the data were collected to run a process, not to answer your question.

Our activity.csv exists because the learning platform logs events for operational reasons. Nobody designed it to measure engagement. It measures *what the platform can see*, which is not the same thing.

2.2 Population, sampling frame, and sample

Three distinct things get confused constantly.

- The **target population** is everyone the conclusion is meant to describe.
- The **sampling frame** is the list you can actually draw from.
- The **sample** is who ends up in your data.



Coverage error lives between the rings, and no increase in sample size will fix it.

Our midterm_satisfaction column illustrates all three. The target population is all enrolled students. The frame is students with active platform accounts. The sample is those who answered. Each gap excludes someone systematically — and Chapter 4 will show that the students who did not answer are exactly the ones the analysis most needs.

2.3 Sampling strategies and trade-offs

Strategy	How	Strength	Cost
Simple random	Every unit equally likely	Unbiased, easy to analyse	Needs a complete frame
Stratified	Random within groups	Guarantees small-group coverage	Need group membership in advance
Cluster	Sample whole groups	Cheap when units are grouped	Larger uncertainty per unit
Convenience	Whoever is available	Fast, free	Bias of unknown size and direction

Convenience samples are not merely less precise. Their bias does not shrink as the sample grows. A survey of 5,000 volunteers can be more misleading than a random sample of 200, because a bigger convenience sample gives a tighter confidence interval around the *wrong* number.

2.4 Operational definitions and measurement validity

An **operational definition** is the rule that turns a concept into a column. “Engagement” is a concept; submissions >= 1 in a given week is an operational definition.

Two questions test any operational definition:

- **Validity:** does it measure the concept, or something correlated with it? minutes_on_platform measures a browser tab, not attention.
- **Reliability:** would it give the same value if measured again? A login count is highly reliable and only moderately valid — a good reminder that the two are independent.

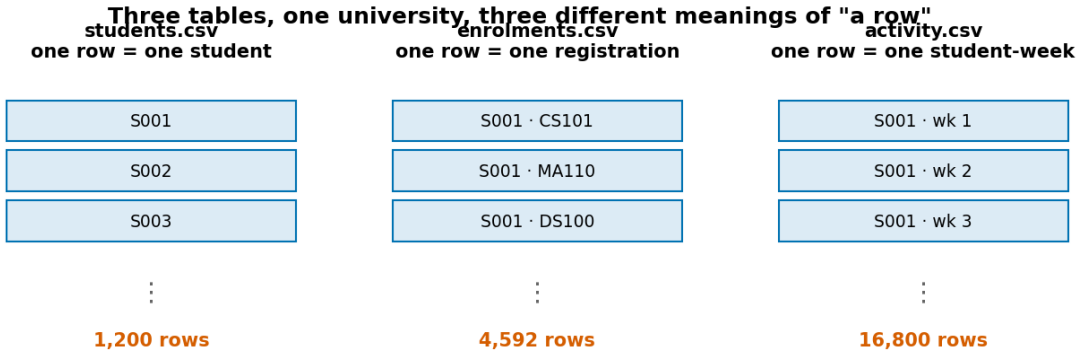
A **proxy** is a variable standing in for something unmeasured. Proxies are often necessary and always worth naming. When a proxy is used as a model input, the model learns the proxy, not the concept, and will fail exactly where the two diverge.

2.5 Formats, APIs, and databases

```
# CSV — the workhorse. Always specify dtypes for identifiers.
students = pd.read_csv("data/students.csv", dtype={"student_id": str})

# JSON — nested by nature; flatten deliberately
import json
with open("data/config.json") as f:
    cfg = json.load(f)
records = pd.json_normalize(cfg["records"])

# From an API (illustrative — always check terms of use and rate limits)
import requests
resp = requests.get("https://api.example.org/v1/courses",
                    params={"term": "2026-T3"}, timeout=10)
resp.raise_for_status()
courses = pd.json_normalize(resp.json()["data"])

# From a database
import sqlite3
con = sqlite3.connect("university.db")
df = pd.read_sql_query(
    "SELECT student_id, course_code, final_grade FROM enrolments WHERE term = ?",
    con, params=("2026-T3",))
con.close()
```

Two habits worth forming now. Set identifier columns to str on load — student_id read as an integer will lose leading zeros and invite meaningless arithmetic. And raise_for_status() on every request, so a 500 error fails loudly instead of silently parsing an error page as data.

2.6 Unit of observation, joins, and cardinality



Three tables from one university. "A row" means something different in each.

Joining tables with different units is the single most common way to silently corrupt an analysis. Before every merge, state the cardinality you expect and let pandas check it:

```
merged = enrolments.merge(
    students, on="student_id",
    how="left",
    validate="many_to_one")    # raises if students has duplicate IDs
print(len(enrolments), "→", len(merged))    # 4665 → 4665, as expected
```

validate= is the cheapest bug-prevention in pandas. Use it every time.

2.7 Data dictionaries and provenance

A **data dictionary** records, for each column: its name, type, meaning, units, permitted values, missingness, and known quirks. **Provenance** records where the data came from, who collected it, when, under what consent or licence, and what has been done to it since.

Neither is bureaucratic overhead. Six weeks after you build a dataset you will not remember whether passed means >= 50 or >= 40, and the person who inherits your work never knew.

The data dictionary for this course's dataset ships with the data. Read it; it is the model for the one you will write.

Worked example 2.1 — The response rate that hides a bias

The university runs a mid-term satisfaction survey. Results come back with a mean of 3.5 out of 5, and the Vice-Chancellor's office wants to report "students are broadly satisfied."

Step 1 — Check the response rate.

```
n_total = len(outcomes)
n_resp  = outcomes.midterm_satisfaction.notna().sum()
print(f"{n_resp} of {n_total} responded ({n_resp/n_total:.1%})")
# 916 of 1200 responded (76.3%)
print("mean score:", outcomes.midterm_satisfaction.mean().round(2))   # 3.50
```

76.3% is a good response rate by survey standards. That is exactly why it is dangerous — it feels safe enough to skip the next step.

Step 2 — Ask who is missing. Non-response only matters if non-responders differ from responders. We cannot see their satisfaction, but we *can* see their engagement:

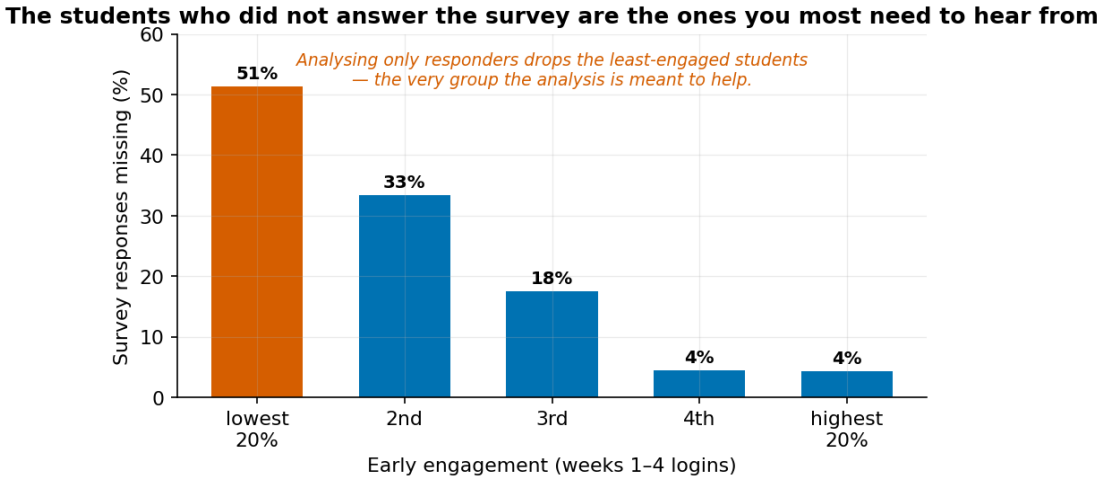
```
responded = outcomes.midterm_satisfaction.notna()
print(outcomes.groupby(responded).early_logins.mean().round(1))
# midterm_satisfaction
# False    11.9     <- non-responders
# True     21.1     <- responders
```

Non-responders logged in barely half as often. They are not a random 24%.

Step 3 — Quantify the gradient.

```
outcomes["q"] = pd.qcut(outcomes.early_logins, 5,
                        labels=["lowest 20%", "2nd", "3rd", "4th", "highest 20%"])
print(outcomes.groupby("q", observed=True)
          .midterm_satisfaction.apply(lambda s: s.isna().mean())
          .round(3))
# lowest 20%    0.514
# 2nd           0.333
# 3rd           0.175
# 4th           0.045
# highest 20%   0.043
```

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY



Missingness rises steeply as engagement falls.

More than half the least-engaged students are absent from the survey; almost none of the most-engaged are.

Step 4 — Bound the damage. Suppose non-responders would have averaged 2.5 rather than 3.5 — plausible given the engagement gradient. Then the true mean would be:

$$0.763 \times 3.50 + 0.237 \times 2.5 = 2.67 + 0.59 = 3.26$$

The headline drops from 3.50 to 3.26. If non-responders averaged 2.0, it drops to 3.14. This is a **sensitivity analysis**: we cannot know the true value, but we can show how much the conclusion depends on an assumption we cannot check.

The honest sentence:

> *Among the 76% of students who responded, mean satisfaction was 3.5 of 5. Non-responders logged in barely half as often as responders, so this figure is likely an overestimate of cohort satisfaction. Under plausible assumptions about non-responders the cohort mean falls between 3.1 and 3.5.*

Limitation: engagement is itself only a proxy for whatever drives both non-response and dissatisfaction. The bound above is illustrative, not a confidence interval, and should never be presented as one.

> **Faded variant.** Compute the same non-response comparison using early_submissions instead of early_logins. Does the gradient hold? Then answer the harder question: what would you need to observe to demonstrate that the missingness is *not* related to satisfaction itself?

Worked example 2.2 — Proxy versus concept

The retention team proposes flagging students whose minutes_on_platform falls below a threshold. Is that a good operational definition of disengagement?

Step 1 — Look at what the proxy actually correlates with.

```
tot = activity.groupby("student_id").agg(
      minutes=("minutes_on_platform", "sum"),
      logins=("logins", "sum"),
      subs=("submissions", "sum")).reset_index()
tot = tot.merge(outcomes[["student_id", "withdrew"]], on="student_id")

for col in ["minutes", "logins", "subs"]:
   r = tot[col].corr(tot.withdrew.astype(int))
   print(f"{col:8s} correlation with withdrawal: {r:+.3f}")
# minutes  correlation with withdrawal: -0.293
# logins   correlation with withdrawal: -0.290
# subs     correlation with withdrawal: -0.274
```

All three point the right way, and — importantly — all three are about equally strong. There is no predictive reason to prefer one. That makes the choice between them a question about *fairness and explicability*, not accuracy.

Step 2 — Ask how the proxy could fail. Construct the failure cases explicitly rather than hoping they are rare:

Student behaviour	minutes	Actually disengaged?
Downloads slides, works offline, submits everything	Low	No
Leaves the tab open during other work	High	Possibly yes
Uses the mobile app, which logs sessions differently	Low	No
Reads everything twice because they are struggling	High	Struggling, but engaged

Two of the four failure cases produce **false alarms for students who are doing fine**, and one produces a **missed case**. Crucially, the first and third rows are not random noise — they correlate with working patterns, which correlate with employment and with device access.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

Step 3 — Test that concern in the data.

```
tot = tot.merge(students[["student_id", "study_mode"]], on="student_id")
print(tot.groupby("study_mode")[["minutes", "logins", "subs"]].mean().round(1))
#             minutes  logins  subs
# Full-time    1655.6    63.6   5.5
# Part-time    1217.5    46.6   5.5
```

Part-time students show 26.5% fewer minutes and 26.8% fewer logins — but only 1.2% fewer submissions. A minutes-based or login-based threshold therefore flags part-time students at a sharply disproportionate rate, while a submissions-based threshold does not. Since all three predict withdrawal equally well, the minutes threshold buys no accuracy in exchange for that disparity.

Conclusion. minutes_on_platform is reliable — it will give the same number tomorrow — but weakly valid, and its errors fall unevenly across a group defined by circumstance rather than effort. submissions is the better operational definition: equally associated with the outcome, essentially undifferentiated by study mode, and easier to explain to a student who asks why they were flagged.

Notice what made the decision. The correlations were nearly identical, so no amount of predictive-performance comparison would have chosen between them. The disaggregation by study_mode did.

Responsible-use note. "Explainable to the person affected" is a real design criterion, not a courtesy. A student can contest "you submitted nothing in four weeks." Contesting "your minutes-on-platform metric was low" requires understanding a logging system they cannot see.

> **Faded variant.** Propose an operational definition of "struggling academically" using only columns available in week 4. Name two ways it could fail, and check in the data whether either failure falls unevenly across study_mode or first_generation.

Worked example 2.3 — A join that silently triples your data

You want mean grade by programme. Both facts live in different tables, so you merge.

```
bad = enrolments.merge(students, on="student_id")
print(bad.groupby("programme").final_grade.mean().round(2))
# Biology          58.19
# Computer Science   59.83
# Economics        58.04
# Psychology       58.72
```

The numbers look reasonable. They are also answering a question you did not ask.

What went wrong. bad has one row per *registration*. A student in four courses contributes four grades; a student in three contributes three. The "mean grade by programme" is therefore a mean over registrations, weighting students by how many courses they took.

Does it matter here? Test it rather than assume:

```
per_student = (enrolments.groupby("student_id").final_grade.mean()
        .rename("student_mean").reset_index()
        .merge(students[["student_id", "programme"]], on="student_id"))
print(per_student.groupby("programme").student_mean.mean().round(2))
# Biology          58.19
# Computer Science   59.84
```

```
# Economics      58.12
# Psychology     58.76
```

The difference is tiny — at most 0.08 of a mark — because 1,065 of the 1,200 students take exactly four courses and only 135 take three. **That is a property of this dataset, not a general reassurance.** Widen the variation in course counts and the gap grows immediately. You cannot know which case you are in without checking, and the check costs two lines.

The habit that prevents this. State the expected row count before merging, then assert it:

```
before = len(enrolments)
merged = enrolments.merge(students, on="student_id",
                          how="left", validate="many_to_one")
assert len(merged) == before, f"row count changed: {before} → {len(merged)}"
```

If students had contained a duplicate student_id, validate="many_to_one" would have raised immediately. Without it, the merge would have silently duplicated every affected registration, and every downstream mean would be quietly wrong.

Limitation: an assertion on row count catches duplication, not omission. An inner join that drops 200 unmatched students preserves nothing to assert on. Check both directions:

```
print("unmatched:", (~enrolments.student_id.isin(students.student_id)).sum())   # 0
```

> **Faded variant.** Merge activity.csv onto students.csv and compute mean entry_score by programme from the result. Predict *before running* whether it will match the value computed from students.csv alone, and by how much it could differ in the worst case. Then explain why the direction of the merge matters.

Common mistakes

Mistake	Consequence	Prevention
Reporting a mean without a response rate	Overstates a self-selected group	Always report n, N, and who is missing
Merging without validate=	Silent row duplication	Use validate= on every merge
Reading IDs as integers	Lost leading zeros, meaningless means	dtype={"student_id": str}
Trusting a proxy because it correlates	Model learns the proxy, fails where they diverge	Write out the failure cases before using it
No data dictionary	Nobody can reuse or audit the work	Write it while you still remember

Chapter summary

Data are produced, not found. The population, the sampling frame, and the sample are three different things, and coverage error between them does not shrink with sample size. Operational definitions turn concepts into columns and deserve explicit justification; proxies deserve explicit failure analysis. Joins between tables with different units are the most common route to silent corruption, and validate= catches most of it. Provenance and a data dictionary are what make an analysis reusable by anyone, including you.

Check your understanding

1. Distinguish target population, sampling frame, and sample using the mid-term survey.

2. Why can a large convenience sample be worse than a small random one?

3. What is a proxy, and what should you do before using one as a model feature?

4. You merge two tables and the row count grows from 1,200 to 1,340. What are the two most likely causes, and how do you distinguish them?

> **Model answers are in the Reference volume, Part 1.** Write your own answer first — comparing a finished attempt against a model answer teaches far more than reading the model answer cold.

Lab work — Lab 1: Sources, provenance, and a data dictionary

> The full two-hour version of this lab, with timings, is **Lab 1** in the Lab Manual (shared with Chapter 1).

Deliverable: notebook plus a one-page data dictionary.

1. Load students.csv, enrolments.csv, and activity.csv with explicit dtypes.
2. For each pair of tables, state and verify the cardinality of the relationship.
3. Merge all three into a student-level table. Assert the row count at every step.
4. Call one public API of your choice, save the raw response, and load it into a DataFrame. Record the endpoint, the date, and the terms of use.
5. Write a data dictionary for your merged table: every column, its type, meaning, units, permitted values, and missingness.
6. Write a 150-word provenance note covering source, licence, collection method, and every transformation you applied.

Chapter 3 — Describing Data: Centre, Spread, and Shape

> *A single number is a summary, and every summary is a decision about what to discard.*

After this chapter you can: compute and interpret measures of centre and spread; choose between them by question and distribution shape; standardise values; and recognise when a summary conceals more than it reveals.

This chapter appeared as Chapter 7 in the first edition. It is here because exploratory analysis (Chapter 6) and feature scaling (Chapter 5) both require it.

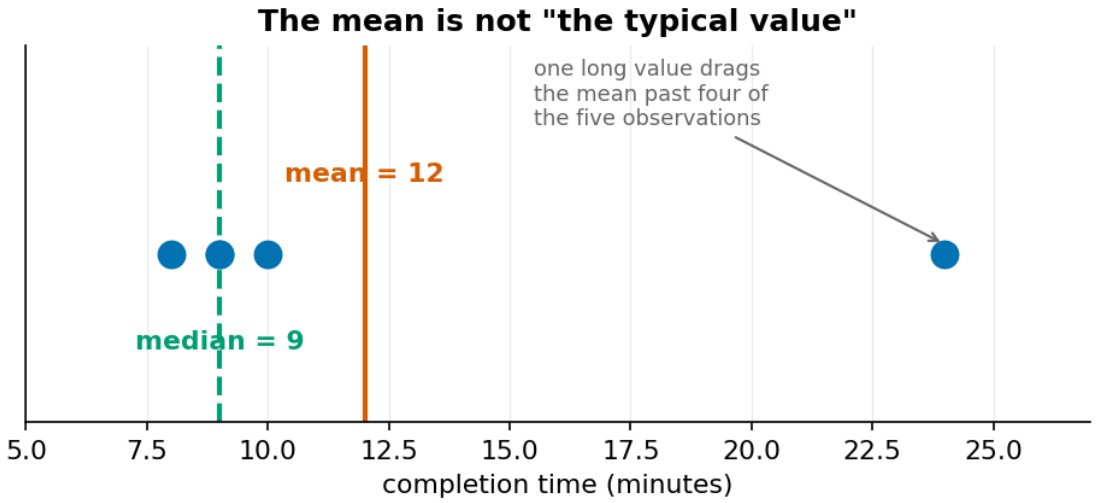
3.1 Statistics as compression

A summary statistic compresses many numbers into one. Compression is lossy by definition. The skill is not computing the mean — software does that — but knowing what the compression threw away and whether the discarded part mattered.

3.2 Centre

Consider five completion times in minutes: 8, 9, 9, 10, 24.

- **Mean:** $(8+9+9+10+24)/5 = 60/5 = 12$
- **Median:** the middle ordered value = 9
- **Mode:** the most frequent value = 9



The mean sits above four of the five observations.

The mean uses every value and is pulled by the 24. The median reports position and ignores magnitude. Neither is superior; they answer different questions. If you are budgeting total staff time, the mean is what you want, because total = mean × n. If you are describing what a typical student experiences, the median is closer.

3.3 Spread

Range and interquartile range describe position. Variance and standard deviation describe typical deviation from the mean.

Subtract the mean from each value: $-4, -3, -3, -2, +12$. These sum to zero — always, by construction, which is why we cannot simply average them. Squaring prevents cancellation:

$$s^2 = \frac{(-4)^2 + (-3)^2 + (-3)^2 + (-2)^2 + 12^2}{5-1} = \frac{16+9+9+4+144}{4} = \frac{182}{4} = 45.5$$

$$s = \sqrt{45.5} \approx 6.75 \text{ minutes}$$

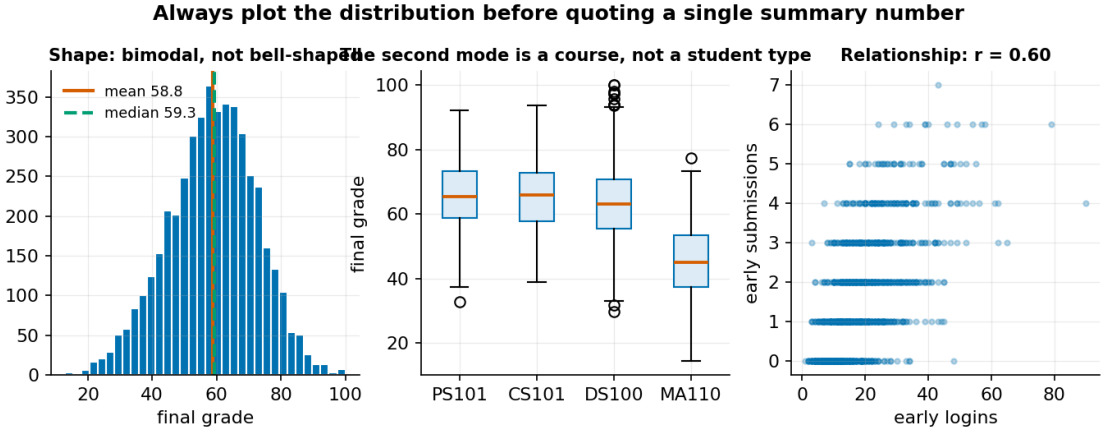
We divide by $n-1$, not n, because the deviations were taken from the *sample* mean rather than the unknown population mean, which makes them slightly too small. Dividing by $n-1$ corrects for that. With large n the distinction barely matters; with n = 5 it changes the answer by 12%.

The **interquartile range** (Q3 − Q1) is the robust alternative: it ignores the tails entirely, so a single extreme value cannot move it.

3.4 Shape

Four properties are worth checking on any numerical variable, in this order:

1. **Modality** — one peak or several? A bimodal distribution usually means two populations are mixed.
2. **Skew** — is one tail longer? Right skew pulls the mean above the median.
3. **Spread** — how wide, in units you can interpret?
4. **Outliers** — extreme values, and whether they are errors or real.



Final grades are bimodal, and the second mode turns out to be a course, not a student type.

The left panel shows why "mean grade = 58.8" is nearly useless here. The distribution has two peaks, and the middle panel identifies the cause: MA110, CH110, and ST110 are hard quantitative service courses with much lower grades. The mean sits in the valley between the modes, describing almost nobody.

A single summary number should not be reported before its distribution has been looked at. This is the most frequently violated rule in applied statistics.

3.5 Standardisation

A **z-score** reports how many standard deviations a value lies from the mean:

$$z = \frac{x - \acute{x}}{s}$$

If $\acute{x} = 70$, $s = 10$, and $x = 85$, then $z = 1.5$: the value is one and a half standard deviations above average.

Standardisation makes variables on different scales comparable, and it matters enormously for distance-based methods (Chapters 5 and 13). Two things it does *not* do: it does not make a skewed distribution normal, and it does not turn an outlier into a non-outlier. A z-score of 6 is still a z-score of 6.

3.6 Choosing a summary

Situation	Use	Because
Roughly symmetric, no extremes	Mean and SD	Uses all the information
Skewed or has extremes	Median and IQR	Robust to the tail
Need a total	Mean	Total = mean × n
Comparing across scales	z-scores	Removes units
Bimodal	**None of the above**	Report the groups separately

Worked example 3.1 — When the mean and the median disagree

Compute both for final_grade, and decide which to report.

```
g = enrolments.final_grade
print(f"mean   {g.mean():.2f}")    # mean   58.78
print(f"median {g.median():.2f}")  # median 59.30
print(f"sd     {g.std():.2f}")     # sd     13.72
print(f"IQR    {g.quantile(.75) - g.quantile(.25):.2f}")  # IQR    18.40
```

Step 1 — Read the sign of the gap. The mean sits 0.52 marks *below* the median. Mean < median indicates **left skew**: a tail of low values pulling the mean down. The gap is small, which tells us the skew is mild — but a mild skew in a pooled distribution can still conceal a large one in the parts.

Step 2 — Find the tail.

```
print((g < 40).mean().round(3))        # 0.093 — 9.3% below 40
print((g > 90).mean().round(3))        # 0.008 — 0.8% above 90
```

Over nine per cent of grades fall below 40, against under one per cent above 90 — an eleven-to-one asymmetry. The distribution is not symmetric and the left tail is heavy.

Step 3 — Ask whether the tail is one thing or many.

```
hard = ["MA110", "CH110", "ST110"]
print(enrolments.groupby(enrolments.course_code.isin(hard)).final_grade
    .agg(["count", "mean", "median", "std"]).round(2))
#       count   mean  median   std
# False  3600  63.13   63.00  11.22
# True   1065  44.08   43.80  10.94
```

The tail is a *course effect*, not a scattering of struggling students. Within the hard courses, mean and median agree almost exactly (44.08 vs 43.80); within the others they also agree (63.13 vs 63.00). Each subgroup is close to symmetric. Only the mixture is skewed — and the two subgroup means are 19 marks apart, which is nearly one and a half pooled standard deviations.

Step 4 — Decide what to report. Neither the mean nor the median of the pooled data is a good summary, because the pooled data are two populations. The honest report is:

> *Mean final grade was 63.1 in standard courses (n = 3,600) and 44.1 in the three quantitative service courses (n = 1,065). A single cohort-wide figure of 58.8 sits between the two group means and describes neither.*

Limitation: "hard courses" is a category we imposed after seeing the data. That is legitimate for description and dangerous for inference — a subgroup discovered by looking is not the same as one specified in advance. Chapter 9 returns to this.

> **Faded variant.** Compute mean and median early_logins. Predict the direction of the skew *before* computing, using what you know about how counts behave near zero. Then check whether splitting by study_mode explains it, as splitting by course explained the grade skew.

Worked example 3.2 — Standard deviation by hand, and why n−1

Five students score 52, 58, 63, 71, 76. Compute the sample standard deviation without software, then check.

Step 1 — Mean.

$$\acute{x} = \frac{52 + 58 + 63 + 71 + 76}{5} = \frac{320}{5} = 64$$

Step 2 — Deviations. Subtract 64 from each:

$$-12, -6, -1, +7, +12$$

Sum: $-12 - 6 - 1 + 7 + 12 = 0$. It always will. This is a useful arithmetic check — if your deviations do not sum to zero, the mean is wrong.

Step 3 — Square them.

$$144, 36, 1, 49, 144 \quad \text{sum} = 374$$

Step 4 — Divide. Sample variance divides by $n - 1 = 4$:

$$s^2 = \frac{374}{4} = 93.5$$

Step 5 — Square root, returning to the original units.

$$s = \sqrt{93.5} \approx 9.67 \text{ marks}$$

Step 6 — Check against software, and inspect the default.

```
import numpy as np, pandas as pd
x = np.array([52, 58, 63, 71, 76])

print(pd.Series(x).std())     # 9.669...  pandas uses ddof=1 by default
print(np.std(x))              # 8.648...  numpy uses ddof=0 by default
print(np.std(x, ddof=1))      # 9.669...  now they agree
```

This discrepancy is a real source of bugs. pandas defaults to the sample standard deviation ($n - 1$); NumPy defaults to the population one (n). Mixing the two libraries in one analysis and comparing the results will produce a difference you cannot explain until you find this.

Why $n - 1$? The deviations were measured from $\acute{x}$, which was itself computed from the same five numbers. The sample mean is, by construction, the value that makes the sum of squared deviations as small as possible — so those deviations are systematically too small as estimates of deviation from the *true* population mean. Dividing by $n - 1$ instead of n inflates the result by exactly the right amount on average. Here it is the difference between 8.65 and 9.67, about 12%. At n = 500 it is 0.1%.

Interpretation: a standard deviation of 9.67 marks means typical scores sit within roughly 10 marks of 64. It does *not* mean 68% fall within one SD — that is a property of the normal distribution, and five observations tell you nothing about whether this one is normal.

> **Faded variant.** For the values 2, 4, 6: explain why the deviations must sum to zero before computing anything, then compute s by hand and verify with both pandas and numpy. Predict which will give the larger answer, and by what factor.

Worked example 3.3 — A z-score that changes a decision

Two students apply for a scholarship. Amara scored 78 on the Economics entry test; Bilal scored 71 on the Computer Science one. The tests are different, so the raw scores are not comparable.

Step 1 — Get the reference distributions.

```
ref = students.groupby("programme").entry_score.agg(["mean", "std"]).round(2)
print(ref)
#                    mean    std
# Biology            66.53  10.53
# Computer Science   67.48  10.86
# Economics          64.46  10.18
# Psychology         64.23  11.92
```

Step 2 — Standardise each score against its own distribution.

Amara, Economics:

$$z = \frac{78 - 64.46}{10.18} = \frac{13.54}{10.18} \approx 1.33$$

Bilal, Computer Science:

$$z = \frac{71 - 67.48}{10.86} = \frac{3.52}{10.86} \approx 0.32$$

Step 3 — Read the answer. Amara stands 1.33 standard deviations above her cohort; Bilal 0.32 above his. Relative to peers, Amara's performance is substantially stronger, and the gap in standardised terms (1.01 SD) is far larger than the raw 7-mark gap suggests.

```
def z(score, programme):
    row = ref.loc[programme]
    return (score - row["mean"]) / row["std"]

print(f'Amara: {z(78, "Economics"):.2f}")          # 1.33
print(f"Bilal: {z(71, "Computer Science"):.2f}")   # 0.32
```

Step 4 — Say what this does not establish. The z-score answers "how unusual is this score within its own group." It does not answer "who will do better", and it does not make the two tests measure the same thing. If the Economics test is easier, a high z-score on it reflects the cohort, not the difficulty.

Worse, standardising *within programme* silently accepts each programme's cohort as the right comparison group. If one programme admits a stronger cohort, its students are penalised by this method — a strong student in a strong cohort gets a lower z than an equally strong student in a weak one.

Responsible-use note. This is a live fairness question, not a technical footnote. Whether to standardise within group or across the whole population changes who receives the scholarship, and there is no purely statistical answer. The choice must be stated in the policy, justified, and open to challenge.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

Faded variant. A third applicant scores 74 in Psychology. Compute her z-score within programme (you should get +0.82). Then recompute all three against the *pooled* distribution across all programmes, where the answers are +1.11, +0.47, and +0.74. Identify whose ranking changes and explain why the pooled and within-programme methods disagree. Write two sentences on which comparison you would defend to an appeals panel.

In code

```
import pandas as pd, numpy as np

g = enrolments.final_grade

# Centre and spread, together — never one without the other
summary = pd.Series({
   "n":       len(g),
   "mean":    g.mean(),
   "median":  g.median(),
   "sd":      g.std(),           # ddof=1
   "IQR":     g.quantile(.75) - g.quantile(.25),
   "min":     g.min(),
   "max":     g.max(),
   "skew":    g.skew(),
})
print(summary.round(2).to_string())

# Robust vs non-robust, demonstrated
contaminated = pd.concat([g, pd.Series([0, 0, 0])])
print("mean moved by  ", round(contaminated.mean()   - g.mean(), 3))    # -0.038
print("median moved by", round(contaminated.median() - g.median(), 3))  #  0.000  <- unmoved

# Standardise
students["entry_z"] = (students.entry_score - students.entry_score.mean()) / students.entry_score.std()
print(students.entry_z.agg(["mean", "std"]).round(3))   # 0.0, 1.0 by construction

# Within-group standardisation
students["entry_z_prog"] = students.groupby("programme").entry_score.transform(
   lambda s: (s - s.mean()) / s.std())
```

Common mistakes

Mistake	Why it matters	Prevention
Reporting a mean for bimodal data	Describes nobody	Always plot the distribution first
Mixing numpy and pandas SD	Silent 1–12% discrepancy	Set ddof explicitly, every time
"68% fall within one SD"	Only true for normal distributions	Check the shape before quoting the rule
Standardising after splitting train/test	Leakage — see Chapter 5	Fit the scaler on training data only
Treating a z-score as a fairness fix	It embeds the comparison group as a choice	State and defend the reference group

Chapter summary

Every summary discards information, and the discarded part is sometimes the finding. Mean and median answer different questions and their disagreement is diagnostic: mean above median means right skew, below means left. Spread needs units, a denominator, and a distribution to be interpreted. Quantiles, IQR fences, groupwise summaries, and z-scores make the decision behind a summary inspectable. The extended field guide below turns these tools into a reporting contract: name the unit, the reference population, n and missingness, the shape, and any rule used to flag unusual values.

Check your understanding

1. For a right-skewed income distribution, which is larger, the mean or the median, and why?

2. Why does sample variance divide by n − 1?

3. A colleague reports "mean grade 58.8, SD 15.4" for the whole cohort. What is your objection?

4. A student has a z-score of 1.5 on the entry test. What does this tell you about their percentile rank?

> **Model answers are in the Reference volume, Part 1.** Write your own answer first — comparing a finished attempt against a model answer teaches far more than reading the model answer cold.

Lab work — Lab 2: Summaries by hand and in code

> The full two-hour version of this lab, with timings, is **Lab 2** in the Lab Manual.

Deliverable: notebook with a written comparison.

1. Take any 8 values from entry_score. Compute mean, median, variance, SD, and IQR **by hand**, showing each step. Verify with pandas and explain any disagreement.
2. Set ddof=0 and ddof=1 on the same column. Quantify the difference at n = 8, n = 100, and n = 1,200, and plot how it shrinks.
3. Produce a summary table of final_grade by course_code: n, mean, median, SD, IQR. Identify every course where mean and median differ by more than 2 marks and explain each.
4. Standardise entry_score two ways: against the full cohort, and within programme. Find the student whose rank changes most between the two, and write three sentences on which you would use for a scholarship decision.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

3.7 The summary contract: unit, population, n, and missingness

Before calculating a centre or spread, write a one-line contract for the summary. It states what one observation represents, who is eligible to be counted, how many values entered the calculation, and who is missing. This takes less than a minute and prevents the common error of treating a result from a partial response set as a result for the full cohort.

Question	FDS teaching-data answer	Why it changes the claim
What is one row?	For students.csv, one row represents one enrolled student.	Rows can instead be events, attempts, or course enrolments; counting them as people then overstates n.
Who is the population?	The enrolled cohort contains N = 1,200 students.	A result for a course, programme, or survey respondents is not automatically a result for all students.
How many values were observed?	midterm_satisfaction is observed for n = 916 students.	The calculation uses the observed denominator, not the full cohort denominator.
Who is missing?	284 of 1,200 values are missing (23.7%).	If non-responders differ from responders, the observed summary may not describe the whole cohort.

> A defensible opening sentence is: "Among 916 of 1,200 enrolled students who reported a midterm satisfaction score, …" The wording does not fix missing data, but it makes the limit visible rather than letting a reader accidentally assume 1,200 responses.

Do not hide the denominator in a table caption. Put n beside every groupwise result, especially when groups have different response rates. A mean based on n = 8 and a mean based on n = 800 deserve different confidence, even if the numbers happen to match.

3.8 Quantiles, the five-number summary, and boxplots

Quantiles answer positional questions. After sorting a numerical column, the median is the 50th percentile; Q1 is the 25th percentile and Q3 is the 75th percentile. Together with the minimum and maximum, they form the five-number summary. It gives more structure than a mean and standard deviation without pretending that a distribution has one typical shape.

Minimum	Q1 (25th)	Median (50th)	Q3 (75th)	Maximum
35.0	58.3	65.9	73.6	99.0

For entry_score in students.csv, the middle half of observed scores lies from 58.3 to 73.6. Its interquartile range is IQR = 73.6 − 58.3 = 15.3 marks. Notice the different statement

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

being made: the median describes a middle position; the IQR describes the spread of the middle half; neither claims that every student is close to 65.9.

A boxplot is a compact drawing of this summary. The box runs from Q1 to Q3, the line inside is the median, and the whiskers extend to the most extreme values not flagged by the chosen rule. It is useful for comparing many groups because the same scale is repeated. It is not a substitute for a histogram: boxplots hide modality, gaps, and fine detail in the tails.

> Quantile conventions differ slightly across software, particularly for small samples or values lying between two observations. For course work, use the stated pandas method consistently. For a regulated or externally audited report, name the software and quantile convention in the methods note.

3.9 IQR fences: flag unusual values; do not silently delete them

A familiar exploratory rule flags values beyond 1.5 IQR from the box. The lower and upper fences are Q1 − 1.5 × IQR and Q3 + 1.5 × IQR. The word is flag, not remove. A fence is a reproducible way to ask “what is happening here?”, not a verdict that the observation is erroneous.

```
minutes = activity.groupby("student_id")["minutes_on_platform"].sum()
q1, q3 = minutes.quantile([.25, .75])
iqr = q3 - q1
upper_fence = q3 + 1.5 * iqr
flagged = minutes[minutes > upper_fence]
```

In the teaching data, total activity minutes have Q1 = 821.00, Q3 = 2,157.25, and IQR = 1,336.25. The upper fence is 4,161.63 minutes. Twenty-three students sit beyond it, and the maximum is 6,717 minutes. That is a prompt to inspect the records and the measurement process, not permission to make the distribution look tidier.

What you find	Appropriate response	What to record
A physically impossible value, such as age = 305	Check source and units; correct only with evidence, otherwise set aside with a documented reason.	Original value, evidence, replacement or exclusion rule.
A plausible but extreme total, such as 6,717 activity minutes	Keep it unless it is demonstrably a duplicate, unit error, or logging failure.	Why it was retained and whether the summary is sensitive to it.
A subgroup creates the tail	Split the report by the meaningful group rather than trimming the group away.	Grouping variable, group n, and comparison summary.

> In the teaching data, excluding the 23 flagged totals would lower the median from 1,426 to 1,405 minutes and the mean from 1,592.42 to 1,525.86 minutes. That sensitivity is itself a result: report the before-and-after values, document the

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

operational decision, and do not let a deletion rule hide a conclusion-changing choice.

3.10 When one distribution is actually several groups

A pooled histogram can be multimodal because different populations have been mixed. The combined result may be mathematically correct yet descriptively misleading. In the enrolment data, the cohort-wide mean final grade is 58.78, while DS100 averages 63.11 and MA110 averages 45.39. A single middle value makes those courses look more alike than they are.

```
course_summary = (enrolments.groupby("course_code").final_grade
   .agg(n="count", mean="mean", median="median", sd="std"))
print(course_summary.round(2))
```

Use the sequence: plot first, identify a substantive grouping variable, then report groupwise n, centre, and spread. Do not create dozens of groups just to hunt for a striking difference. If a grouping was discovered after looking at the data, label it exploratory and avoid treating it as a pre-specified test.

A histogram's appearance also depends on bin width. Fewer bins can conceal two peaks; too many can turn random noise into a jagged story. Try a small range of sensible bin widths, use the same bins when comparing groups, and state the scale in the figure caption. A stable feature that survives reasonable choices is more credible than a feature visible in only one plot.

3.11 Z-scores, empirical percentile, and the reference-group decision

A z-score is not an intrinsic property of a score. It is a comparison with a chosen reference distribution: z = (x − mean) / SD. For an entry score of 99, the z-score is +3.03 against the whole cohort, +2.90 against Computer Science, and +3.39 against Economics. The raw score did not move; the reference group did.

That choice can be a fairness decision. Standardising within programme asks "how unusual is this score among programme peers?" Standardising across the cohort asks a different question. Neither choice is automatically neutral, so a report must justify the reference group before any ranking or allocation is made.

```
x = 99
z_whole = (x - students.entry_score.mean()) / students.entry_score.std()
empirical_percentile = 100 * (students.entry_score <= x).mean()
print(z_whole, empirical_percentile)
```

Do not translate a z-score mechanically into a percentile. The familiar 68–95 rule is a normal-curve approximation; it is most useful for distributions that are roughly unimodal and symmetric. An empirical percentile counts the observed distribution directly. Use it when rank is what the decision needs, and report ties or the chosen ranking convention when they matter.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

3.12 A minimal, defensible descriptive report

The final step is prose. A numerical table is not self-interpreting: it needs a sentence that links the summary to the decision, names what was excluded, and alerts the reader to the shape or grouping that could change the conclusion.

Include	Question the reader should not have to ask
Unit and population	What does one row represent, and who was eligible to appear?
n, N, and missingness	How many records were used, out of how many possible records?
Centre and spread	What is typical, and how much do values vary in the original units?
Shape or group structure	Could skew, outliers, gaps, or mixed groups make the single number misleading?
Decision and limitation	What action does the number inform, and what does it not establish?

> Template: "Among [n] observed [unit] from [N] eligible [population], [variable] had [median or mean] of [value] and [IQR or SD] of [value]. The distribution was [shape / group pattern]; [missingness or outlier decision]. Therefore [decision-relevant interpretation], but this does not establish [limit]."

Example: "Among 1,200 enrolled students, entry_score had a median of 65.9 marks and an IQR of 15.3; the mean was 65.86 and the distribution was close to symmetric. These values describe the incoming cohort, not a causal effect of programme choice." This is compact, reproducible, and harder to over-read than a bare mean.

Extended chapter recap

Use the smallest summary that answers the decision question, then state what it leaves out. Always show n and the unit; use medians and IQRs when tails matter; plot before claiming a distribution has one centre; treat an outlier rule as an investigation trigger; split meaningful groups; and name the reference population for every z-score. These habits make descriptive analysis useful before a model is ever fitted.

Further reading

NIST/SEMATECH e-Handbook of Statistical Methods: Histograms, Box Plots, and Detection of Outliers. These short reference pages explain the graphical and exploratory conventions used in this field guide.

Chapter 4 — Data Quality and Cleaning

> *Cleaning is not tidying. It is a sequence of decisions, each of which changes what the data can tell you.*

After this chapter you can: assess fitness for purpose; diagnose missingness mechanisms; resolve duplicates and type errors; distinguish outliers from errors; and keep a cleaning log that makes every decision auditable.

4.1 Quality is fitness for purpose

There is no absolute standard of clean data. A dataset with 5% missing ages is unusable for an age-stratified analysis and perfectly fine for a question that never touches age. Quality is a relationship between a dataset and a question, which is why cleaning cannot sensibly begin before the question is written.

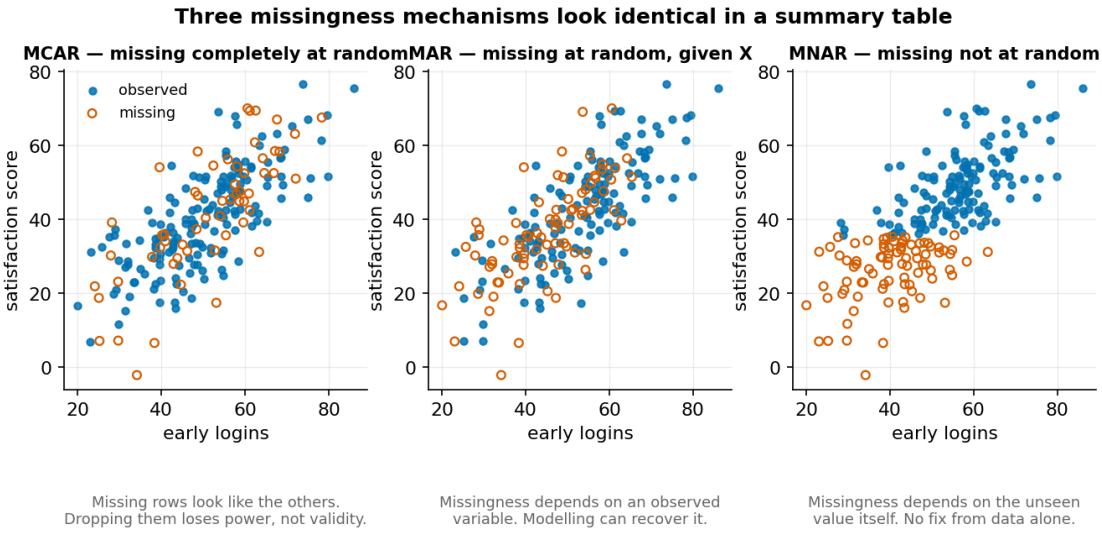
Six dimensions are worth checking explicitly:

Dimension	Question	Check
Completeness	What is missing, and why?	df.isna().sum() plus a missingness *pattern* check
Validity	Are values in permitted ranges?	Range and set-membership assertions
Consistency	Do related fields agree?	Cross-field rules (registered_on before term start)
Uniqueness	Is each entity represented once?	duplicated() on the key, not the whole row
Accuracy	Do values match reality?	Usually unanswerable from the data alone — flag it
Timeliness	Is the snapshot current enough?	Provenance, not the file

Accuracy is the one you cannot check internally, and it is the one people assume. A perfectly complete, valid, consistent, unique dataset can be entirely wrong.

4.2 Missingness has mechanisms

Missing values are not a uniform problem. Three mechanisms behave differently and demand different responses.



The same summary table is produced by all three mechanisms. Only the pattern distinguishes them.

MCAR — missing completely at random. The probability of being missing is unrelated to anything. Dropping those rows loses statistical power but does not bias the result. This is the benign case, and it is rare.

MAR — missing at random given observed variables. Missingness depends on something you *can* see. Older students skip a question more often, and you have age. Modelling can recover the structure, because the thing driving missingness is in your data.

MNAR — missing not at random. Missingness depends on the unobserved value itself. Students who are dissatisfied do not answer the satisfaction survey. **No purely statistical fix exists**, because the information needed to correct the bias is precisely the information you do not have.

You cannot test MAR against MNAR from the data. You can only reason about the mechanism, look for observable gradients, and report what you assumed.

4.3 The missingness in our data

Chapter 2 established that midterm_satisfaction is missing for 23.7% of students and that the missingness rises steeply as engagement falls: 51.4% of the least-engaged quintile is absent, against 4.3% of the most engaged.

Is that MAR or MNAR? It depends on what you condition on. Given early_logins, which we observe, it looks like MAR — and we could impute using engagement. But if dissatisfaction *itself* drives both disengagement and non-response, then conditioning on logins does not close the gap and the data are MNAR. Both stories fit the observed pattern equally well.

The professional response is not to pick one. It is to do the analysis under both assumptions and report how much the answer moves — the sensitivity analysis from Worked Example 2.1.

4.4 Duplicates and entity resolution

Three different things get called duplicates:

- **Exact duplicates:** every field identical. Usually a loading error. Safe to drop.
- **Key duplicates with identical content:** same student_id, same values. Same as above.
- **Key duplicates with *different* content:** same student_id, different entry_score. **These are not duplicates.** They are either two records of the same entity at different times, or a data-entry conflict. Dropping one at random silently picks a winner.

The third case is the dangerous one because drop_duplicates() handles it silently, keeping whichever row came first.

4.5 Outliers are a question, not a category

An outlier is a value far from the others. That is a *description*, not a diagnosis. Four different things produce them:

1. **Impossible values** — age_at_entry = 220. Definitionally an error. Fix or remove, and record it.
2. **Possible but wrong** — age_at_entry = 2. Could be a typo for 20 or 21. You usually cannot tell.
3. **Real and rare** — a 55-year-old part-time student. Removing this is not cleaning; it is deleting the population you were asked about.
4. **Real and influential** — a legitimate value that dominates a model. Keep it and report its influence.

The default should be to keep, flag, and analyse sensitivity — not to remove. Every removal narrows what your conclusion is about.

4.6 The cleaning log

Every cleaning decision changes the dataset and therefore the result. A **cleaning log** records, for each decision: what you found, how many rows it affected, what you did, and why. It is the difference between an analysis someone can audit and one they must simply trust.

Issue	Rows	Action	Justification
programme case/whitespace variants	150	.str.strip().str.title()	16 distinct values collapse to the 4 documented ones
entry_score stored as "67.3%"	44	Strip %, cast to float	Format artefact of the export, not a different measurement

Issue	Rows	Action	Justification
age_at_entry outside 16–70	9	Set to missing	Values of 0, 1, 2, 178, 199, 220, 305 are impossible
Exact duplicate rows	2	Drop	Identical in every field
student_id duplicated, differing values	3	Keep most recent registered_on	Later record supersedes; both retained in the audit file
entry_score missing	55	Retain as missing	Do not impute before knowing the mechanism

Worked example 4.1 — A cleaning log for `students_dirty.csv`

The registry sends an export. Before analysing anything, inventory the damage.

Step 1 — Never trust the shape.

```
dirty = pd.read_csv("data/students_dirty.csv")
print(dirty.shape)              # (1205, 8)
print("expected 1200 students")
```

Five rows too many. Something is duplicated.

Step 2 — Separate the three kinds of duplicate.

```
print("exact duplicate rows:", dirty.duplicated().sum())          # 2
print("duplicated student_id:", dirty.student_id.duplicated().sum())  # 5
```

Two rows are identical in every field — safe to drop. But five IDs repeat, so three of them repeat with *different content*. Look at them:

```
dupe_ids = dirty.student_id[dirty.student_id.duplicated(keep=False)]
conflicts = dirty[dirty.student_id.isin(dupe_ids)].sort_values("student_id")
print(conflicts[["student_id", "entry_score", "registered_on"]].to_string())
```

Three students appear twice with different entry_score and a later registered_on. These are **updates**, not duplicates. drop_duplicates() would keep whichever row happened to be first in the file — which is to say, at random.

```
# Wrong: silently picks a winner
clean = dirty.drop_duplicates(subset="student_id")

# Right: state the rule
dirty["reg_date"] = pd.to_datetime(dirty.registered_on, format="mixed", dayfirst=True)
clean = (dirty.sort_values("reg_date")
```

```
        .drop_duplicates(subset="student_id", keep="last"))
print(len(clean))      # 1200
```

Step 3 — Categorical variants.

```
print(dirty.programme.nunique())       # 16
print(sorted(dirty.programme.unique())[:6])
# [' Biology  ', ' Computer Science  ', ' Economics  ', ' Psychology  ',
#  ' biology  ', ' computer science  ']
```

Sixteen values for four programmes: leading/trailing whitespace and lowercase variants, in every combination. A groupby("programme") on this produces sixteen groups and a nonsense table.

```
clean["programme"] = clean.programme.str.strip().str.title()
print(clean.programme.nunique())       # 4
```

Step 4 — Types masquerading as values.

```
print(clean.entry_score.dtype)                    # object
print(clean.entry_score.astype(str).str.contains("%").sum())    # 44
```

Forty-four scores exported as "67.3%". The column is text, so any arithmetic on it fails or — worse — silently concatenates.

```
clean["entry_score"] = pd.to_numeric(
    clean.entry_score.astype(str).str.strip("%"), errors="coerce")
```

errors="coerce" turns anything still unparseable into NaN rather than raising. That is the right choice *only if you then check how many it created*:

```
print("newly missing after coercion:", clean.entry_score.isna().sum() - 55)   # 0
```

Zero. Had it been non-zero, we would have silently destroyed data.

Step 5 — Impossible values.

```
clean["age_at_entry"] = pd.to_numeric(clean.age_at_entry, errors="coerce")
bad = (clean.age_at_entry < 16) | (clean.age_at_entry > 70)
print(sorted(clean.loc[bad, "age_at_entry"].dropna()))
# [0.0, 0.0, 1.0, 1.0, 2.0, 178.0, 199.0, 220.0, 305.0]
clean.loc[bad, "age_at_entry"] = pd.NA
```

Note what we did *not* do: guess. 2 might be a truncated 21, and 178 might be 17.8 — but we do not know, and inventing a value is worse than admitting ignorance. Setting them missing is honest and preserves the row's other fields.

Step 6 — Assert, so the next run fails loudly.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

```
assert clean.student_id.is_unique
assert set(clean.programme) == {"Biology", "Computer Science", "Economics",
"Psychology"}
assert clean.entry_score.dropna().between(0, 100).all()
assert clean.age_at_entry.dropna().between(16, 70).all()
print("all checks passed —", len(clean), "rows")     # 1200 rows
```

Assertions turn silent corruption into a loud failure. Write them once and every future re-run is checked for free.

Limitation: none of this addresses *accuracy*. Every value could now be well-formed, in range, unique, and still wrong — a mistyped 67 that should be 76 passes every check above. Internal validation cannot detect that; only comparison against an external source can.

> **Faded variant.** The registered_on column arrives in three formats: 2025-03-14, 14/03/2025, and Mar 14, 2025. Parse it correctly and prove your parse is right. Specifically: identify which format is ambiguous, construct a date where a wrong parse changes the *year* or *month*, and show what dayfirst=True versus dayfirst=False does to it.

Worked example 4.2 — Missing for a reason

You want mean satisfaction by study mode. The naive route:

```
d = outcomes.merge(students, on="student_id")
print(d.groupby("study_mode").midterm_satisfaction.mean().round(2))
# Full-time    3.51
# Part-time    3.44
```

Nearly identical. Report it?

Step 1 — Report n alongside every mean. Always.

```
print(d.groupby("study_mode").midterm_satisfaction
    .agg(["count", "mean"]).round(2))
#             count  mean
# Full-time     789  3.51
# Part-time     127  3.44
print(d.study_mode.value_counts().to_dict())
# {'Full-time': 1027, 'Part-time': 173}
```

Response rates: 789/1027 = 76.8% full-time, 127/173 = 73.4% part-time. Similar, which is mildly reassuring — but only about the *rates*.

Step 2 — Check whether the missingness gradient differs by group.

```
d["responded"] = d.midterm_satisfaction.notna()
print(d.groupby(["study_mode", "responded"]).early_logins.mean().round(1))
# Full-time  False   12.4
#            True    22.1
# Part-time  False   10.3
#            True    16.7
```

In both groups, non-responders are far less engaged. The gradient is real and it is present in both.

Step 3 — Bound the effect on the comparison. Suppose non-responders in each group would have scored 1.0 below their group's responders:

$$\text{Full-time: } 0.768 \times 3.51 + 0.232 \times 2.51 = 2.70 + 0.58 = 3.28$$

$$\text{Part-time: } 0.734 \times 3.44 + 0.266 \times 2.44 = 2.53 + 0.65 = 3.18$$

The gap moves from 0.07 to 0.10. The comparison is fairly robust to this assumption, because both groups have similar response rates.

Step 4 — Find the assumption it is *not* robust to. Suppose part-time non-responders are more dissatisfied than full-time ones — plausible, since a part-time student who disengages may have work conflicts the university never hears about. Set full-time non-responders at 2.51 and part-time at 2.00:

$$\text{Part-time: } 0.734 \times 3.44 + 0.266 \times 2.00 = 2.53 + 0.53 = 3.06$$

Now the gap is 0.22 — three times the observed value. The conclusion "study modes report similar satisfaction" survives one assumption and not the other.

The honest report:

> *Among respondents, mean satisfaction was 3.51 (full-time, n = 789) and 3.44 (part-time, n = 127). Response rates were similar (76.8% and 73.4%), so the comparison is not obviously distorted by differential non-response. However, both groups' non-responders were substantially less engaged than their responders, and if part-time non-responders are more dissatisfied than full-time ones the true gap could be three times larger. This comparison should not be used to conclude that the modes are equivalent.*

Never impute before this analysis. Filling the missing values with the group mean would have produced an apparently complete dataset, a tighter confidence interval, and the same bias — now invisible.

> **Faded variant.** Impute midterm_satisfaction three ways: overall mean, group mean, and a regression on early_logins. Recompute the study-mode gap under each. Rank the three by how much they understate uncertainty, and explain why the regression imputation is not simply the best one.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

Worked example 4.3 — Outlier or population?

The age_at_entry column, after the impossible values are removed, still has a long right tail.

```
print(clean.age_at_entry.describe().round(1))
# mean   21.0   std   4.6   min  17.0   25%  19.0
# 50%    19.0   75%  20.0   max  47.0
```

A common reflex is to trim values beyond three standard deviations: $21.0 + 3(4.6) = 34.8$.

Step 1 — Count what that would remove and who they are.

```
cut = clean.age_at_entry.mean() + 3 * clean.age_at_entry.std()
trimmed = clean[clean.age_at_entry > cut]
print(len(trimmed), "students would be removed")
print(trimmed.study_mode.value_counts().to_dict())
```

Nearly all of them are part-time students. This is not a scattering of anomalies; it is a **subpopulation**.

Step 2 — Ask what the rule actually encodes. "Remove students over 35" is, in this dataset, approximately "remove mature part-time students." If the analysis is about widening participation, the rule deletes the entire object of study. If the analysis is about first-year attainment across the whole cohort, it silently redefines "cohort".

Step 3 — Notice the circularity of the rule itself. The mean and SD used to define the cut were computed *including* the values being tested. A few extreme values inflate the SD, which widens the cut, which lets them survive — or, with a tighter distribution, a single extreme value can drag the threshold enough to change which points are flagged. Robust alternatives (median ± 3 × MAD, or the IQR rule) avoid this, but they do not solve the real problem, which is that the question of whether these students belong is not statistical.

Step 4 — Do the sensitivity analysis instead of choosing.

```
for label, subset in [("all students", clean),
                      ("age <= 35", clean[clean.age_at_entry <= 35])]:
    m = subset.merge(outcomes, on="student_id")
    print(f"{label:16s} n={len(m):5d}  withdrawal rate={m.withdrew.mean():.3f}")
```

If the two rates are close, the exclusion does not matter and you should keep the students anyway. If they differ, the exclusion is doing real work and must be justified in the report, not buried in a preprocessing cell.

The rule to carry forward: a value is an error if it is *impossible*, not if it is *unusual*. Age 220 is impossible. Age 47 is a person.

> **Faded variant.** Apply the IQR rule (below Q1 − 1.5·IQR or above Q3 + 1.5·IQR) to early_logins. How many students are flagged? Cross-tabulate them against withdrew. Then answer the question that matters: would removing them make

the withdrawal model better, or would it remove most of the students the model exists to find?

Common mistakes

Mistake	Consequence	Prevention
drop_duplicates() on conflicting records	Silently keeps a random version	Sort by a timestamp and use keep="last"
Imputing before diagnosing the mechanism	Hides bias behind a complete-looking table	Diagnose first, impute later or not at all
Removing outliers by SD rule	Can delete an entire subpopulation	Ask “impossible or unusual?”
errors="coerce" without a follow-up count	Silently destroys unparseable data	Always count new NaNs after coercion
Cleaning in a notebook without a log	Nobody can audit or reproduce it	Write the log as you go

Chapter summary

Quality is fitness for a specific purpose, so cleaning follows the question rather than preceding it. Missingness has mechanisms — MCAR, MAR, MNAR — that cannot be distinguished from the data alone, so the response is sensitivity analysis rather than a chosen fix. Key duplicates with differing content are updates, not duplicates, and need an explicit rule. Outliers are impossible or merely unusual, and only the first is an error. Every cleaning decision belongs in a log with a count and a justification, and assertions turn future silent corruption into loud failure.

Check your understanding

1. Give an example of MNAR missingness in the course dataset and explain why no statistical fix exists.

2. Why is drop_duplicates() dangerous on records sharing a key but differing in content?

3. A value is 4 standard deviations from the mean. Should you remove it?

4. You coerce a text column to numeric and the row count is unchanged. Is the operation safe?

> **Model answers are in the Reference volume, Part 1.** Write your own answer first — comparing a finished attempt against a model answer teaches far more than reading the model answer cold.

Lab work — Lab 3: Clean the registry export

The full two-hour version of this lab, with timings, is **Lab 3** in the Lab Manual.

Deliverable: cleaned CSV, notebook, and a cleaning log table.

1. Load students_dirty.csv. Produce a full damage inventory before changing anything.
2. Fix each issue in a separate, commented cell. For every fix, print the number of rows affected.
3. Handle the duplicate-with-update records with an explicit, stated rule.
4. Parse registered_on from its three formats. Prove your parse is correct on the ambiguous cases.
5. Do **not** impute missing values. Instead, produce a missingness diagnosis for each column with missing data.
6. Write assertions covering uniqueness, permitted categories, and value ranges. Confirm they pass.
7. Submit the cleaning log as a table: issue, rows affected, action, justification.

Chapter 5 — Transformation, Features, and Leakage

> *Representation determines what an algorithm can see. Leakage determines whether your evaluation means anything.*

After this chapter you can: scale and transform numerical variables; encode categorical ones; engineer features from dates; recognise and prevent leakage; and build a scikit-learn pipeline that keeps evaluation honest.

5.1 Why representation matters

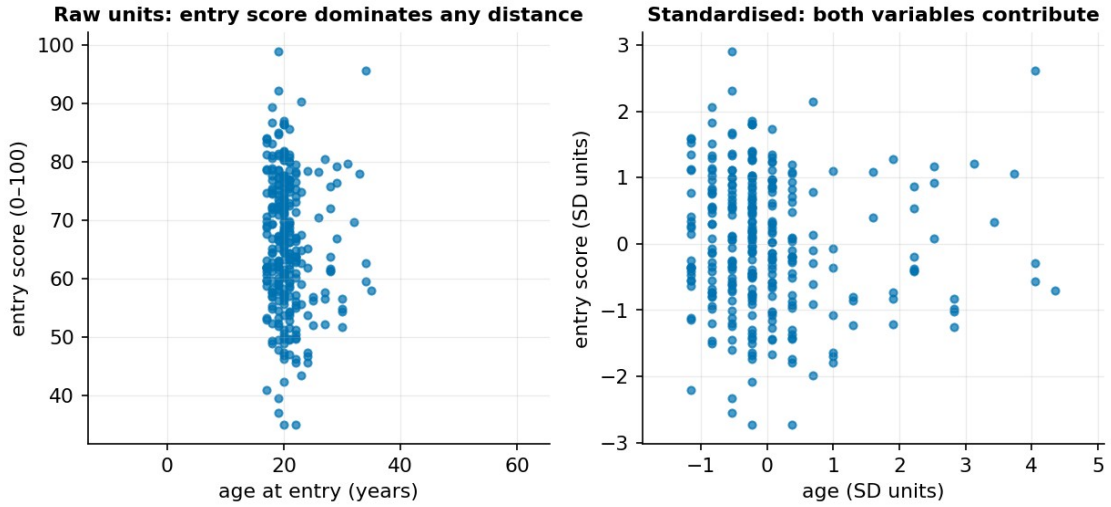
A model does not see “a student”. It sees a vector of numbers. Every choice about how to turn a column into numbers changes what patterns are learnable. A date stored as a string is invisible to a linear model; the same date split into month, day-of-week, and days-since-term-start may carry most of the signal.

5.2 Numerical transformations

Standardisation subtracts the mean and divides by the standard deviation, producing z-scores. Use it for distance-based methods (k-NN, k-means, SVM) and regularised linear models.

Min–max scaling maps observed limits to a chosen interval, usually [0, 1]. Sensitive to future values outside the training range.

Log transformation compresses right skew and turns multiplicative relationships into additive ones. Requires positive values; use log1p when zeros are present.



A k-means or k-NN model on the left is effectively a model of entry score alone. Fit the scaler on training data only.

k-means on the left is effectively a model of entry score alone.

Tree-based models do not need scaling — they split on thresholds, and a monotone transformation of a feature leaves the split structure unchanged. Distance-based models need it badly. This is not a stylistic preference: in the left panel above, entry_score spans about 60 units and age_at_entry about 30, so Euclidean distance is dominated by the former and the clusters are entry-score bands.

5.3 Categorical encoding

One-hot encoding creates one binary column per category. Safe, interpretable, and expands dimensionality. For our four programmes it costs four columns.

Ordinal encoding maps categories to integers. Correct only when the order is real *and* the spacing is meaningful. Encoding programme as 0–3 tells a linear model that Psychology is three times Biology, which is nonsense.

Target encoding replaces a category with the mean outcome for that category. Powerful and a notorious leakage source: computed on the full dataset, it lets each row's own outcome influence its own feature.

```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(sparse_output=False, handle_unknown="ignore")
programmes = enc.fit_transform(students[["programme"]])
print(enc.get_feature_names_out())
# ['programme_Biology' 'programme_Computer Science'
#  'programme_Economics' 'programme_Psychology']
```

handle_unknown="ignore" matters at deployment. Without it, a category unseen during training raises an error in production.

5.4 Features from time

```
activity["week_of_term"] = activity.iso_week
activity["is_early"] = activity.iso_week <= 4

# Lag and rolling features — one row's history, not its future
activity = activity.sort_values(["student_id", "iso_week"])
activity["logins_prev"] = activity.groupby("student_id").logins.shift(1)
activity["logins_roll3"] = (activity.groupby("student_id")
                            .logins.transform(lambda s: s.rolling(3).mean()))
```

Note shift(1) and the ordering. A rolling mean computed without sorting, or centred rather than trailing, uses future values — which brings us to the chapter's real subject.

5.5 Leakage

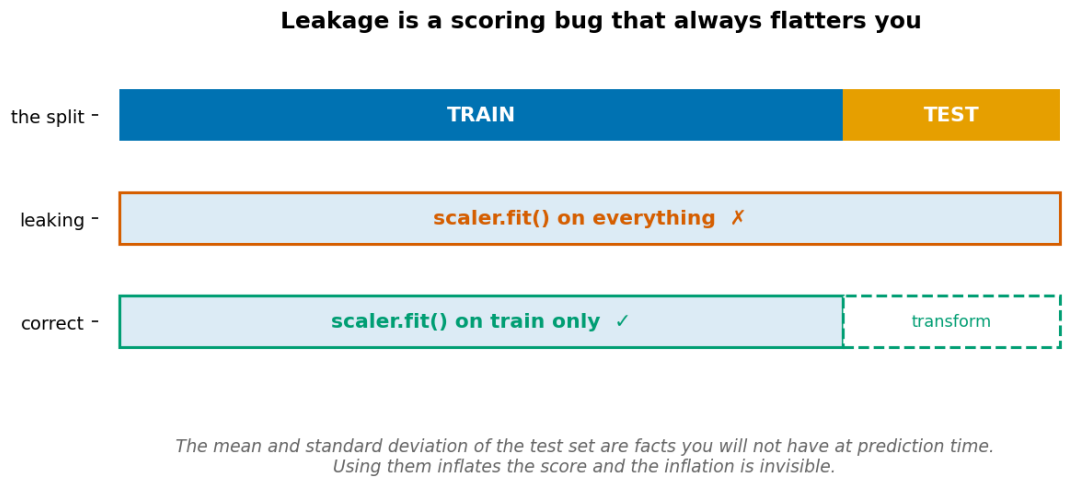
Leakage is any situation where information unavailable at prediction time influences training or evaluation. It is the single most common reason a model that scored well in development fails in deployment, and it is dangerous precisely because it produces *better* numbers.

Three families:

Target leakage — a feature is a consequence of the outcome. Predicting withdrawal using full-term activity: a withdrawn student has no activity in week 10 *because* they withdrew.

Train–test contamination — a preprocessing step sees the test set. Fitting a scaler, imputer, or feature selector on all the data before splitting.

Temporal leakage — training on data from after the prediction point. Covered in Chapter 14.



Fit on train, transform both. Never fit on everything.

5.6 Pipelines make leakage structurally difficult

A Pipeline binds preprocessing to the model so that fit and transform respect the split automatically. It is not a convenience — it is a correctness mechanism.

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression

numeric = ["early_logins", "early_submissions", "entry_score", "age_at_entry"]
categorical = ["programme", "study_mode"]

pre = ColumnTransformer([
   ("num", Pipeline([("impute", SimpleImputer(strategy="median")),
                     ("scale", StandardScaler())]), numeric),
   ("cat", OneHotEncoder(handle_unknown="ignore", sparse_output=False), categorical),
])

model = Pipeline([("pre", pre),
                  ("clf", LogisticRegression(max_iter=1000))])
```

Every step inside model now learns its parameters from training folds only, whether you call fit, cross_val_score, or GridSearchCV. The median used for imputation, the mean and SD used for scaling, and the category list used for encoding are all fitted per fold.

Worked example 5.1 — Scaling by hand, and why the model cares

Three students: ages 18, 20, 22 and entry scores 55, 70, 85.

Step 1 — Standardise age. Mean $= (18+20+22)/3 = 20$. Deviations: $-2, 0, +2$. Squared: $4, 0, 4$; sum 8; variance $= 8/2 = 4$; SD $= 2$.

$$z_{\text{age}} = -1, 0, +1$$

Step 2 — Standardise entry score. Mean $= (55+70+85)/3 = 70$. Deviations: $-15, 0, +15$. Squared: $225, 0, 225$; sum 450; variance $= 225$; SD $= 15$.

$$z_{\text{score}} = -1, 0, +1$$

Both variables now have mean 0 and SD 1 — which is the whole point.

Step 3 — See what changed for a distance calculation.

Raw distance between student 1 (18, 55) and student 3 (22, 85):

$$\sqrt{(22-18)^2+(85-55)^2}=\sqrt{16+900}=\sqrt{916}\approx 30.3$$

Of that squared distance, age contributes 16 and score contributes 900 — **98.3% of the distance comes from one variable**, purely because marks are measured on a wider scale than years.

Standardised:

$$\sqrt{(1-(-1))^2+(1-(-1))^2}=\sqrt{4+4}=\sqrt{8}\approx 2.83$$

Now each contributes exactly half.

Step 4 — Confirm in code.

```
import numpy as np
from sklearn.preprocessing import StandardScaler

X = np.array([[18, 55], [20, 70], [22, 85]], dtype=float)
Z = StandardScaler().fit_transform(X)
print(Z.round(3))
# [[-1.225 -1.225]
#  [ 0.     0.   ]
#  [ 1.225  1.225]]
```

Why ±1.225 and not ±1? StandardScaler divides by the *population* standard deviation (ddof=0), not the sample one. With $n=3$: $\sigma=\sqrt{8/3}=1.633$, so $2/1.633=1.225$. Our hand calculation used s with $n-1$. Both are correct; they answer slightly different questions, and the ratio is $\sqrt{n/(n-1)}=\sqrt{1.5}=1.225$.

This does not affect any downstream model — every value is scaled by the same constant, so distances and coefficients scale uniformly. It *will* confuse you when hand-checking, which is exactly the Chapter 3 ddof trap in a new costume.

> **Faded variant.** Add a fourth student aged 45 with entry score 60. Recompute both standardisations by hand. Predict which variable's z-scores change more, and by how much, before calculating. Then explain why min–max scaling would be affected far more severely by this addition than standardisation is.

Worked example 5.2 — Encoding that lies to the model

Encode programme for a linear model.

The wrong way:

```
mapping = {"Biology": 0, "Computer Science": 1, "Economics": 2, "Psychology": 3}
students["prog_code"] = students.programme.map(mapping)
```

This runs, produces a numeric column, and fits without complaint. It also asserts three things that are false:

1. The programmes have an **order** (Biology < Computer Science < Economics < Psychology).
2. The **gaps are equal** — Biology to CS equals Economics to Psychology.
3. Psychology is **three times** Biology in whatever the coefficient measures.

A linear model fits one coefficient β and predicts $\beta \times$ code. That forces the four programme effects onto a straight line through the codes. If the true effects are Biology +2, CS −1, Economics +3, Psychology 0, no single β can represent them — and the model will not tell you it failed. It will report a coefficient and a p-value as usual.

The right way:

```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(sparse_output=False, drop="first", handle_unknown="ignore")
X = enc.fit_transform(students[["programme"]])
print(enc.get_feature_names_out())
# ['programme_Computer Science' 'programme_Economics' 'programme_Psychology']
```

Four categories become three columns. Biology, the dropped one, becomes the **reference level**: its effect is folded into the intercept, and each remaining coefficient reads as "difference from Biology."

Why drop one? With all four columns present, they sum to 1 for every row — perfectly collinear with the intercept. The design matrix is singular and coefficients are not uniquely determined. This is the *dummy variable trap*. Regularised models and trees tolerate it; ordinary least squares does not.

When ordinal encoding is correct: when the order is real and the spacing is defensible. midterm_satisfaction on 1–5 has a genuine order. Whether the gaps are equal is contested — the distance from "very dissatisfied" to "dissatisfied" need not equal "satisfied" to "very satisfied" — which is why treating Likert scales as continuous is a modelling assumption, not a fact.

Step — Verify the difference costs something.

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

y = students.merge(
      enrolments.groupby("student_id").final_grade.mean(), on="student_id"
   ).final_grade

ordinal = students[["prog_code"]]
onehot  = pd.get_dummies(students.programme, drop_first=True)
```

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

```
for name, Xf in [("ordinal", ordinal), ("one-hot", onehot)]:
    r2 = cross_val_score(LinearRegression(), Xf, y, cv=5, scoring="r2").mean()
    print(f"{name:8s} CV R² = {r2:.4f}")
```

The one-hot version can never do worse, because it can represent everything the ordinal version can and more. If they score the same, the programme effects happened to be near-linear in your arbitrary alphabetical ordering — luck, not validation.

> **Faded variant.** One-hot encode study_mode (2 categories) and entry_cohort (3 categories). How many columns should each produce with drop="first"? Then explain what handle_unknown="ignore" does when a 2027 cohort appears at prediction time, and why the alternative ("error") might sometimes be what you want.

Worked example 5.3 — Leaking the future

Build a model to flag students at risk of withdrawal, so advisers can offer support in week 5.

The tempting version. More data is better, so use all fourteen weeks of activity:

```
full = (activity.groupby("student_id")
        .agg(all_logins=("logins", "sum"), all_subs=("submissions", "sum"))
        .reset_index())
d = outcomes.merge(students, on="student_id").merge(full, on="student_id")
y = d.withdrew.astype(int)

leaky = d[["all_logins", "all_subs", "entry_score"]]
honest = d[["early_logins", "early_submissions", "entry_score"]]

for name, X in [("honest (weeks 1–4)", honest), ("leaky (weeks 1–14)", leaky)]:
    Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=.3,
                                          random_state=0, stratify=y)
    p = make_pipeline(StandardScaler(), LogisticRegression(max_iter=1000)).fit(Xtr, ytr)
    pr = p.predict_proba(Xte)[:, 1]
    print(f"{name:22s} ROC-AUC {roc_auc_score(yte, pr):.3f}")
# honest (weeks 1–4)    ROC-AUC 0.701
# leaky (weeks 1–14)    ROC-AUC 0.775
```

The full-term model is better by 0.074 AUC. In a report, that is the model you would ship.

Why it is worthless. Look at the feature:

```
print(d.groupby("withdrew").all_logins.mean().round(1))
# False   65.3
# True    29.7
```

Withdrawn students have less than half the total logins — but that is *because they left*. Their weeks 8–14 contain no activity for the simple reason that they were no longer enrolled. The model has learned "students who stop appearing have withdrawn", which is true, useless, and unavailable in week 5.

The test that catches it. For every feature, ask: *would I have this value, with this meaning, at the moment I need the prediction?* In week 5, all_logins does not exist. It is not merely unknown — it is not yet defined.

Why 0.074 rather than 0.30. This is the important part. Leakage here is *subtle*. The features are not literally the target; they merely absorb some of it. A modest, plausible improvement is exactly what survives code review — a model that suddenly hit AUC 0.99 would be investigated, and this one would not.

A second, harder-to-spot case. Feature selection outside the pipeline:

```
from sklearn.feature_selection import SelectKBest, f_classif

# WRONG — selection sees every row, including test folds
sel = SelectKBest(f_classif, k=10).fit(X_all, y_all)
scores = cross_val_score(LogisticRegression(), sel.transform(X_all), y_all, cv=5)

# RIGHT — selection is refitted inside each fold
pipe = make_pipeline(SelectKBest(f_classif, k=10), LogisticRegression())
scores = cross_val_score(pipe, X_all, y_all, cv=5)
```

With many noise features the first version can report strong accuracy on pure noise, because the selector found the columns that happen to correlate with y *in this specific dataset*, and cross-validation then never gets a chance to disagree.

Responsible-use note. A leaked model deployed to flag students would perform far worse than advertised, and the failure falls on the students it fails to flag. "The validation looked fine" is not a defence when the validation was structurally incapable of detecting the problem.

> **Faded variant.** Build a model predicting final_grade in DS100 that uses passed as a feature. Report its R^2. Then explain in one sentence why the number is meaningless, and identify the exact moment in time at which passed becomes known.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

Common mistakes

Mistake	Consequence	Prevention
Fitting a scaler before splitting	Optimistic scores	Put every transformer in a Pipeline
Ordinal-encoding a nominal variable	Model asserts a false order	One-hot, unless order is real
Feature selection outside cross-validation	Can score well on pure noise	make_pipeline(SelectKBest(...), model)
Using post-outcome features	Excellent scores, useless model	Ask: available at prediction time?
Centred rolling windows on time data	Uses the future	Trailing windows and shift(1)

Chapter summary

Representation determines what a model can learn. Standardisation matters for distance-based methods and not for trees; one-hot encoding avoids asserting a false ordering; the dummy-variable trap is why one level is dropped. Leakage is information unavailable at prediction time influencing training or evaluation, and it always flatters. Its most dangerous form is not dramatic — a plausible improvement of a few points survives review in a way that an implausible one would not. Pipelines are the structural defence, because they make the correct thing the default.

Check your understanding

1. Why does a decision tree not require feature scaling, while k-nearest-neighbours does?

2. What is the dummy variable trap, and which models does it affect?

3. Give an example of target leakage from the course dataset.

4. You fit SimpleImputer on the full dataset, then split. What exactly has leaked, and would it show up as an error?

> **Model answers are in the Reference volume, Part 1.** Write your own answer first — comparing a finished attempt against a model answer teaches far more than reading the model answer cold.

Lab work — Lab 4: Pipelines and a leakage demonstration

> The full two-hour version of this lab, with timings, is **Lab 4** in the Lab Manual.

Deliverable: notebook with a written comparison.

1. Build a ColumnTransformer handling numeric (impute + scale) and categorical (one-hot) columns from students.csv and outcomes.csv.
2. Wrap it with a LogisticRegression in a Pipeline. Cross-validate. Record the score.

3. Now deliberately leak: fit the scaler and imputer on the full dataset before splitting. Report the difference. Explain why it is small here and name a preprocessing step where it would be large.
4. Build the target-leakage model from Worked Example 5.3. Report both AUCs and write two sentences on why the leaky one would pass code review.
5. Add 400 pure-noise columns. Run SelectKBest inside and outside the pipeline. Report both cross-validated accuracies and explain the gap.
6. For every feature in your final model, state the moment in time at which its value becomes known.

Chapter 6 — Exploratory Data Analysis

> *Exploration is structured, not aimless. The discipline is what separates finding something from finding anything.*

After this chapter you can: run a systematic EDA sequence; read distributions and relationships; detect confounding and Simpson's paradox; and convert a finding into a defensible claim.

6.1 A disciplined EDA sequence

Aimless plotting produces a folder of charts and no conclusions. A sequence produces findings:

1. **Structure** — shape, dtypes, unit of observation, keys.
2. **Univariate** — one variable at a time: modality, skew, spread, outliers.
3. **Missingness** — how much, and patterned how.
4. **Bivariate** — relationships suggested by the question, not by curiosity.
5. **Grouped** — does the relationship hold within meaningful subgroups?
6. **Claim** — write the sentence, with its limitation.

Steps 5 and 6 are the ones people skip, and they are where the errors live.

6.2 Reading a distribution

Four properties, in order: modality, skew, spread, outliers. Chapter 3 established the vocabulary; here it becomes a habit.

The single most useful EDA reflex: **before quoting any summary number, look at the distribution it summarises.** Our final_grade mean of 58.8 is arithmetically correct and describes neither of the two populations that produce it.

6.3 Relationships and confounding

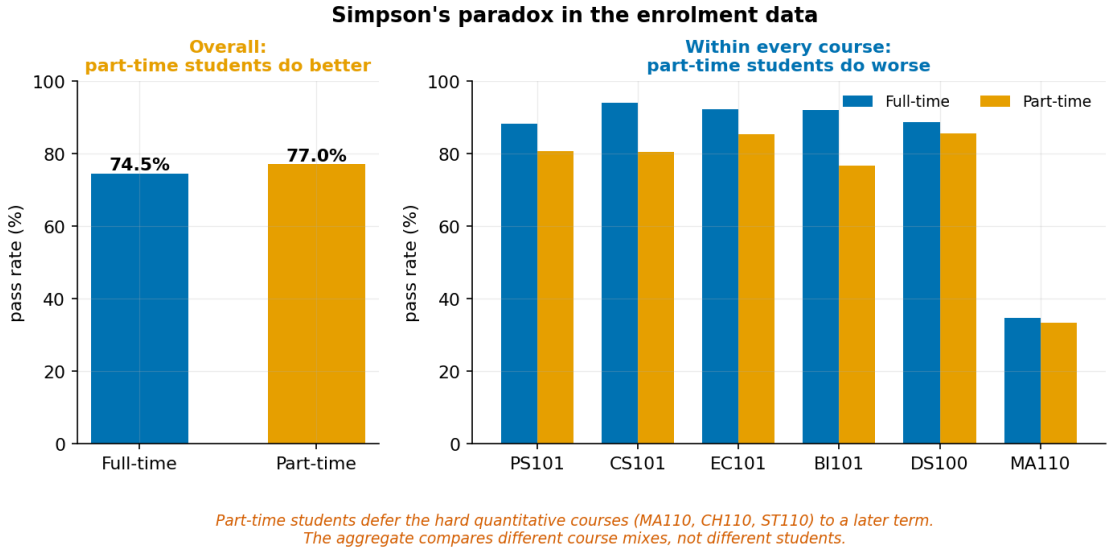
A **confounder** is a variable associated with both the supposed cause and the outcome, producing an association between them that is not a direct relationship.

In our data, entry_score is associated with both total_logins (r = 0.40) and mean_grade (r = 0.62). The observed login–grade correlation of 0.68 therefore cannot be read as the effect of logging in.

The remedy in exploration is **stratification** — look at the relationship within levels of the suspected confounder. If it persists at similar strength within each stratum, the confounder does not explain it. If it weakens or reverses, it does.

6.4 Simpson's paradox

The most dramatic form of confounding: an association present in aggregate reverses within every subgroup.



Part-time students pass at a higher rate overall and a lower rate in eleven of twelve courses.

This is not a curiosity. It is a structural feature of any comparison across groups with different compositions, and it means **an aggregate comparison is not a safe default**.

6.5 From finding to claim

A defensible finding has three parts, always in this order:

1. **Observation** — what the data show, with numbers and n.
2. **Interpretation** — what it plausibly means, with the assumptions named.
3. **Limitation** — what it cannot establish.

A number without a sentence is incomplete. A sentence claiming more than the design supports is wrong.

Worked example 6.1 — The average that hides two populations

Report typical performance in the DS100 course.

```
ds = enrolments[enrolments.course_code == "DS100"]
print(f"n = {len(ds)}, mean = {ds.final_grade.mean():.1f}")
# n = 1200, mean = 63.1
```

Step 1 — Plot before believing.

```
import matplotlib.pyplot as plt
ds.final_grade.hist(bins=30)
plt.xlabel("final grade"); plt.show()
```

Roughly unimodal here — so for DS100 alone the mean is defensible. Now do the same for the whole enrolment table:

```
print(f"all courses: mean = {enrolments.final_grade.mean():.1f}")   # 58.8
enrolments.final_grade.hist(bins=40); plt.show()
```

Two peaks. The pooled mean of 58.8 falls between them.

Step 2 — Find the grouping variable that explains the modes.

```
by_course = (enrolments.groupby("course_code").final_grade
             .agg(["count", "mean", "std"]).round(1)
             .sort_values("mean"))
print(by_course.to_string())
#           count  mean   std
# CH110       260  42.5  10.6
# ST110       229  42.6  10.6
# MA110       576  45.4  11.0
# EC102       274  60.8  10.3
# ...
# PS101       256  65.4  11.4
# CS101       374  65.5  10.7
```

Three courses sit far below the rest. They are the quantitative service courses.

Step 3 — Quantify the split.

```
hard = ["MA110", "CH110", "ST110"]
split = enrolments.groupby(enrolments.course_code.isin(hard)).final_grade.agg(
   ["count", "mean", "median", "std"]).round(2)
print(split.to_string())
#       count   mean  median   std
# False  3600  63.13   63.00  11.22
# True   1065  44.08   43.80  10.94
```

A 19-mark gap between group means, with within-group SDs of about 11. The gap is 1.7 within-group standard deviations — these are substantially separated populations.

Step 4 — Check that the split is not itself confounded. Are weaker students concentrated in the hard courses?

```
m = enrolments.merge(students[["student_id", "entry_score"]], on="student_id")
m["is_hard"] = m.course_code.isin(hard)
print(m.groupby("is_hard").entry_score.mean().round(2))
# False    65.86
# True     66.15
```

Entry scores differ by 0.29 of a mark — indistinguishable. The 19-mark grade gap is a property of the *courses*, not of who takes them. That strengthens the interpretation considerably.

The claim, in three parts:

> **Observation.** Mean final grade was 63.1 across nine standard courses (n = 3,600) and 44.1 across the three quantitative service courses (n = 1,065), a gap of 19.1 marks against within-group standard deviations of about 11.
>
> **Interpretation.** The two groups have comparable mean entry scores (65.9 vs 66.2), so the difference is unlikely to reflect student intake and more likely reflects course difficulty, assessment design, or teaching.
>
> **Limitation.** This is observational and cannot distinguish those three explanations. It also cannot rule out that students self-select into service courses in a way entry score does not capture. A cohort-wide mean of 58.8 should not be reported, as it describes neither group.
>
> **Faded variant.** Repeat the analysis for early_logins, which has skew 1.35. Is it bimodal or merely skewed? These require different responses — identify which, and state what you would report instead of a mean if it turns out to be the latter.

Worked example 6.2 — Simpson's paradox, worked

In Chapter 1 you were asked to fix the sentence *"Part-time students pass at a higher rate, so part-time study improves outcomes."* Here is why it is worse than overstated.

Step 1 — Verify the aggregate.

```
m = enrolments.merge(students[["student_id", "study_mode"]], on="student_id")
print(m.groupby("study_mode").passed.mean().round(4))
# Full-time    0.7446
# Part-time    0.7702
```

Part-time students pass 2.6 percentage points more often. The number is correct.

Step 2 — Disaggregate.

```
print(m.groupby(["course_code", "study_mode"]).passed.mean().unstack().round(3))
#             Full-time  Part-time
# BI101           0.920      0.766
# BI102           0.884      0.723
# CH110           0.257      0.000
# CS101           0.940      0.804
# CS102           0.868      0.732
# DS100           0.887      0.855
# EC101           0.921      0.853
# EC102           0.862      0.882
# MA110           0.346      0.333
# PS101           0.882      0.806
# PS102           0.895      0.833
# ST110           0.218      0.111
```

Part-time students pass at a **lower** rate in eleven of the twelve courses. The single exception, EC102, has only 34 part-time students — a cell small enough that the reversal is well within sampling noise. (Compute a confidence interval on that cell and you will find it comfortably contains the full-time rate.)

Step 3 — Find the mechanism. A paradox always has one, and it is always composition.

```
m["is_hard"] = m.course_code.isin(["MA110", "CH110", "ST110"])
print(m.groupby("study_mode").is_hard.mean().round(3))
# Full-time    0.250
# Part-time    0.068
```

Twenty-five per cent of full-time registrations are in the hard quantitative courses, against 6.8% of part-time ones. Part-time students defer those courses to a later term.

Step 4 — Reconstruct the aggregate arithmetically to prove the mechanism accounts for it. Using overall pass rates of roughly 0.79 in easy courses and 0.28 in hard ones:

$$\text{Full-time: } 0.750 \times 0.79 + 0.250 \times 0.28 = 0.593 + 0.070 = 0.663$$

$$\text{Part-time: } 0.932 \times 0.79 + 0.068 \times 0.28 = 0.736 + 0.019 = 0.755$$

The composition difference alone produces a 9-point gap favouring part-time students — more than enough to swamp the 8-to-15-point *deficit* they show within each course. The aggregate is not measuring student performance at all. It is measuring which courses each group registered for.

Step 5 — Say which comparison answers the question. Neither is universally right; it depends on what you are asking.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

- *"Which group has a higher pass rate this term?"* — the aggregate is the correct answer, and it is 77.0% vs 74.5%.
- *"Does studying part-time make a student less likely to pass a given course?"* — the within-course comparison is correct, and it says yes.
- *"Should we intervene?"* — neither, on its own. The within-course gap suggests part-time students need support; the composition difference suggests their course sequencing is already adaptive.

The claim:

> **Observation.** Part-time students passed 77.0% of registrations against 74.5% for full-time students. Within individual courses, part-time pass rates were lower in eleven of twelve courses, typically by 8–15 percentage points. The single reversal (EC102) has n = 34 and is not distinguishable from noise.
>
> **Interpretation.** The aggregate reverses because course mix differs sharply: 25% of full-time registrations are in the three low-pass-rate quantitative courses against 6.8% of part-time ones. The aggregate comparison is confounded by course difficulty.
>
> **Limitation.** These are observational data. The within-course gap does not establish that part-time study causes lower performance — part-time students differ in employment, caring responsibilities, and available study time, none of which are measured here.
>
> **Faded variant.** Test whether the same paradox appears for programme instead of study_mode. Compute the aggregate pass rate by programme, then the within-course rates. If no reversal appears, explain what would have to be true about programme course-mix for one to occur.

Worked example 6.3 — A relationship that survives stratification, and one that does not

Two candidate findings. One holds up; one dissolves.

Finding A: logins and grades.

```
per_student = (activity.groupby("student_id").logins.sum()
        .rename("total_logins").reset_index()
        .merge(enrolments.groupby("student_id").final_grade.mean()
            .rename("mean_grade"), on="student_id")
        .merge(students[["student_id", "entry_score", "study_mode"]],
            on="student_id"))

print(per_student[["total_logins", "mean_grade", "entry_score"]].corr().round(3))
#              total_logins  mean_grade  entry_score
# total_logins        1.000       0.681        0.402
```

```
# mean_grade        0.681     1.000     0.622
# entry_score       0.402     0.622     1.000
```

r = 0.68 between logins and grades. Is it explained by entry score?

Stratify by entry score.

```
per_student["entry_band"] = pd.qcut(per_student.entry_score, 4,
                            labels=["Q1", "Q2", "Q3", "Q4"])
print(per_student.groupby("entry_band", observed=True)
    .apply(lambda g: g.total_logins.corr(g.mean_grade), include_groups=False)
    .round(3))
```

If the correlation remains around 0.5–0.6 within every band, entry score does not explain it — the relationship survives. The association is real and not merely a reflection of prior preparation.

Finding B: study mode and pass rate. Worked Example 6.2 showed this one reverses under stratification by course. It does not survive.

The difference matters more than either result. Both findings looked identical in aggregate: a clean number from a correct calculation. Only stratification distinguished them, and stratification is a step you have to choose to take.

What to stratify by. You cannot stratify by everything — with enough subgroups every relationship dissolves into noise. Choose variables that are (a) plausibly associated with both sides, and (b) *nameable in advance*. Here, entry score is an obvious candidate for grades and logins; course is an obvious candidate for pass rates. Both were predictable before looking.

Limitation on the whole procedure. Stratification handles confounders you thought of. It does nothing about the ones you did not, and observational data can never rule those out. This is the honest ceiling of exploratory analysis, and it is why the causal rung needs a different design rather than a better computation.

> **Faded variant.** Test whether the relationship between early_submissions and withdrew survives stratification by programme. Predict first whether it will, and say what you would conclude in each case.

In code: a reusable EDA opening

```
def eda_report(df, name="dataset"):
    """The first ten minutes with any table."""
    print(f"=== {name} ===")
    print(f"shape: {df.shape}")
    print("\ndtypes:"); print(df.dtypes.to_string())
```

```
    miss = df.isna().sum()
    if miss.any():
        print("\nmissing:")
        print((miss[miss > 0].to_frame("n")
               .assign(pct=lambda d: (d.n / len(df) * 100).round(1))).to_string())

    num = df.select_dtypes("number")
    if len(num.columns):
        print("\nnumeric:")
        print(num.agg(["count", "mean", "median", "std", "min", "max", "skew"])
                .T.round(2).to_string())

    cat = df.select_dtypes(["object", "bool", "category"])
    for c in cat.columns:
        n = df[c].nunique()
        print(f"\n{c}: {n} distinct")
        if n <= 12:
            print(df[c].value_counts().to_string())

eda_report(students, "students")
```

Note the skew column. A skew above about 1 in absolute value is a signal to plot before summarising — early_logins has skew 1.35.

Common mistakes

Mistake	Consequence	Prevention
Quoting a mean without plotting	Describes a population that does not exist	Histogram first, always
Comparing aggregates across groups	Simpson's paradox	Disaggregate by any composition-differing variable
Stratifying by everything	Every finding dissolves into noise	Name candidate confounders before looking
Reporting a correlation as an effect	Overclaims by two rungs	Write observation, interpretation, limitation separately
Treating a small-cell reversal as a finding	Chases noise	Check n and the interval before interpreting

Chapter summary

EDA is a sequence, not a mood. Distributions come before summaries; missingness patterns come before imputation; stratification comes before any comparison across groups. Simpson's paradox is the extreme case of a general problem: aggregates compare group compositions as much as group members. A finding becomes a claim only when

observation, interpretation, and limitation are stated separately, and the limitation is the part that makes the other two trustworthy.

Check your understanding

1. What four properties should you inspect in a numerical distribution?

2. How can a missingness pattern create an apparent relationship?

3. Explain Simpson's paradox in your own words, using the study-mode example.

4. A correlation of 0.68 survives stratification by entry score. What does that establish, and what does it not?

> **Model answers are in the Reference volume, Part 1.** Write your own answer first — comparing a finished attempt against a model answer teaches far more than reading the model answer cold.

Lab work — Lab 5: A defensible exploratory report

> The full two-hour version of this lab, with timings, is **Lab 5** in the Lab Manual.

Deliverable: notebook with six purposeful figures and a one-page findings brief.

1. Write three analytical questions **before** plotting anything. Label each descriptive, comparative, or predictive.
2. Run eda_report on all four tables. Note anything surprising.
3. Produce univariate summaries for the variables your questions need — no others.
4. Investigate at least three relationships with suitable charts.
5. For each relationship, stratify by one plausible confounder named in advance. Report whether it survives.
6. Find the Simpson's paradox in this dataset independently, and document the composition difference that causes it.
7. Write each finding as observation / interpretation / limitation. A finding without all three does not count.

Chapter 7 — Visualization and Data Storytelling

> *A chart is an argument made visible. Its design can clarify evidence or quietly distort it.*

After this chapter you can: choose chart form by question and variable type; design accessible and honest graphics; show uncertainty; and build an evidence-to-action narrative.

7.1 Match form to question

Question	Form	Avoid
How is one variable	Histogram, density,	Pie chart

Question	Form	Avoid
distributed?	box	
How do a few categories compare?	Bar chart, zero baseline	3-D bar, donut
How do two numerics relate?	Scatter, optionally with fit	Dual y-axes
How does something change over time?	Line	Bar for continuous time
How does a distribution differ by group?	Faceted histogram, violin, box	Overlaid semi-transparent histograms with many groups
What is the composition of a whole?	Stacked bar, treemap	Pie beyond about four slices

The chart form is a consequence of the question. If you cannot name the question, no chart form is correct.

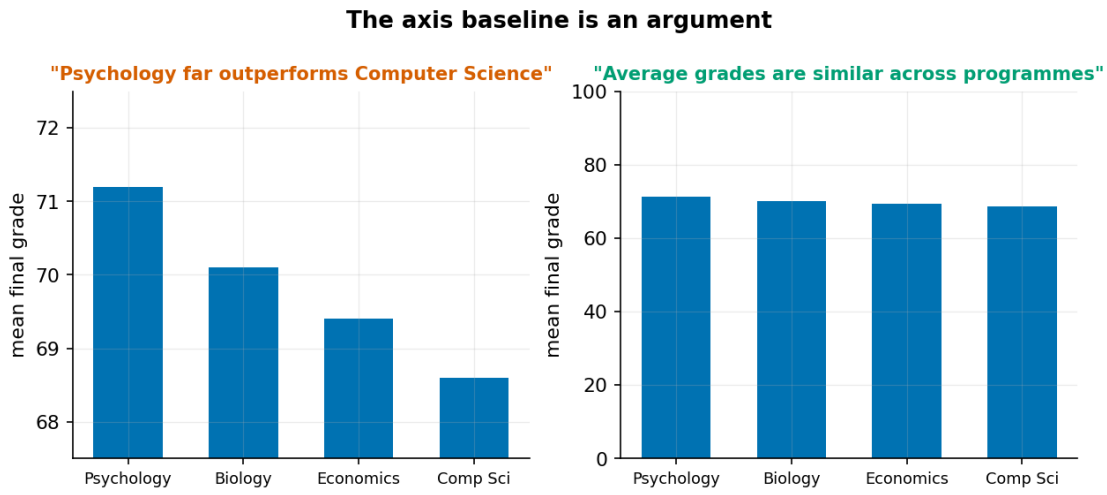
7.2 Visual encoding

Human perception judges some channels far more accurately than others. Roughly, in decreasing accuracy: **position** > **length** > **angle** > **area** > **colour intensity**.

This ordering is why pie charts underperform bar charts — angle is a weaker channel than length — and why bubble charts mislead: doubling a radius quadruples the area, so a reader comparing area sees a fourfold difference where the data show twofold.

Encode your most important comparison on position or length. Reserve colour for categories, not quantities you want read precisely.

7.3 Baselines and honest scales



Same four numbers. The left chart is not mislabelled or miscalculated — only the baseline moved. A truncated axis is a rhetorical choice, so make it deliberately and disclose it.

The same four numbers. Only the baseline moved.

For **bar charts the baseline must be zero**, because the bar's length encodes the value, and a truncated bar's length encodes nothing at all. For **line charts** a non-zero baseline is often legitimate — the line encodes change, not magnitude — but it must be labelled clearly.

A truncated axis is not automatically dishonest. It is a rhetorical choice, and the standard is that you make it deliberately and disclose it, rather than defaulting into it because the plotting library chose the limits.

7.4 Colour and accessibility

About 8% of men and 0.5% of women have some form of colour-vision deficiency, most commonly red–green. Design decisions that follow:

- Use a colour-blind-safe palette. The Okabe–Ito palette used throughout this book is one.
- **Never let colour be the only channel.** Add shape, line style, direct labels, or position.
- Check contrast. Light grey on white fails for many readers and for anyone printing in monochrome.
- Limit categorical colours to about six. Beyond that, facet instead.

```
OKABE_ITO = ["#0072B2", "#E69F00", "#009E73", "#D55E00",
             "#CC79A7", "#56B4E9", "#F0E442", "#000000"]
```

Test any figure by converting it to greyscale. If the message survives, colour is decorative rather than load-bearing — which is where you want it.

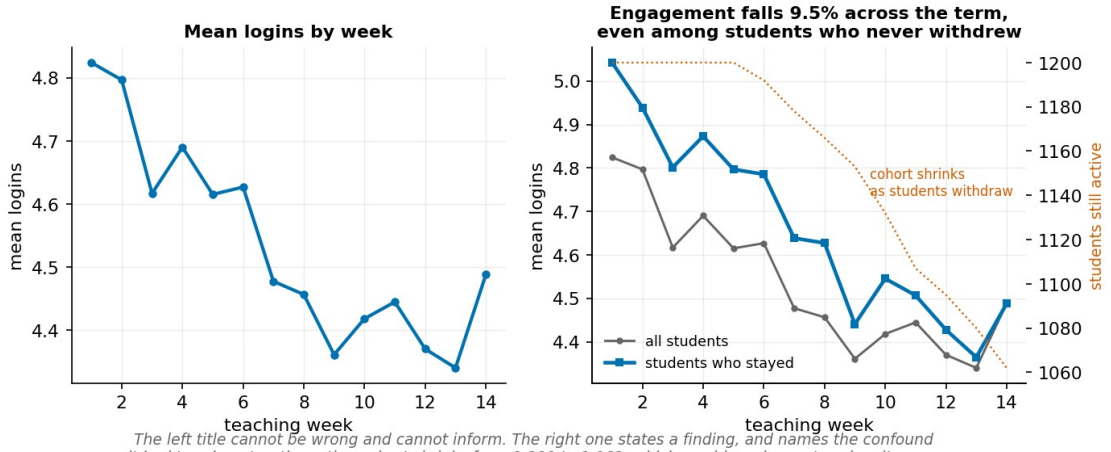
7.5 Showing uncertainty

A point estimate without uncertainty invites over-reading. Error bars, confidence bands, or a plotted distribution all communicate that the number could have come out differently.

Always label what the interval represents. A bar with error bars is ambiguous between standard deviation, standard error, and a 95% confidence interval — three quantities that differ by large factors and mean entirely different things.

7.6 Message titles

A **descriptive title** names the chart's contents: "Submissions by week." A **message title** states the finding: "Submission rates fall by a third after week 8."



The left title cannot be wrong and cannot inform. The right one states a finding, and names the confound it had to rule out — the active cohort shrinks from 1,200 to 1,062, which would produce a trend on its own.

A message title states the finding and can be argued with.

Message titles are more useful and more honest, because a claim in a title is a claim someone can challenge. "Submissions by week" cannot be wrong; it also cannot be informative.

Worked example 7.1 — Reading a deceptive axis

A draft slide reports mean grades by programme with the headline *"Psychology substantially outperforms Computer Science."*

Step 1 — Get the actual numbers.

```
per_student = (enrolments.groupby("student_id").final_grade.mean()
               .rename("mean_grade").reset_index()
               .merge(students[["student_id", "programme"]], on="student_id"))
print(per_student.groupby("programme").mean_grade.mean().round(1))
# Biology           58.2
# Computer Science  59.8
```

```
# Economics      58.1
# Psychology     58.8
```

The headline is false in *direction*: Computer Science is highest at 59.8, Psychology third at 58.8.

Step 2 — Work out what chart could have produced the claim. With a y-axis from 58 to 60, the Psychology bar and the Economics bar differ by 0.7 of the visible height — the visual gap is roughly 35 times the proportional difference in the data. Any two programmes can be made to look dramatically different by choosing limits tightly enough.

Step 3 — Quantify how small the real spread is.

$$\text{range} = 59.8 - 58.1 = 1.7 \text{ marks}$$

Against a within-programme standard deviation of about 11 marks, the spread across programmes is 0.15 SD. Practically, a student's programme tells you almost nothing about their expected grade.

Step 4 — Check whether it is distinguishable from noise at all.

```
from scipy import stats
groups = [g.mean_grade.values for _, g in per_student.groupby("programme")]
print(stats.f_oneway(*groups))
```

Step 5 — Redraw honestly. Zero baseline, and a title that states what is actually true:

```
import matplotlib.pyplot as plt
means = per_student.groupby("programme").mean_grade.mean().sort_values()
fig, ax = plt.subplots(figsize=(7, 3.5))
ax.bar(means.index, means.values, color="#0072B2")
ax.set_ylim(0, 100)
ax.set_ylabel("mean final grade")
ax.set_title("Mean grades differ by under 2 marks across programmes")
plt.show()
```

The lesson is not "always use a zero baseline." It is that the axis choice is part of the argument. Here the honest chart is boring, and the boring chart is the finding: programme does not predict grade. A chart that made this look dramatic would be manufacturing a result.

> **Faded variant.** Take the daily series in platform_daily.csv and produce two line charts of the same data: one supporting "platform use is collapsing" and one supporting "platform use is stable." Use only axis limits and date range — no other changes. Then write the caption you would need to make each defensible.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

Worked example 7.2 — Colour overload and the greyscale test

A dashboard plots mean weekly logins for all twelve courses as twelve coloured lines on one axis.

Step 1 — Count the channels being asked of colour. Twelve categories, one channel. A reader must hold twelve colour-to-course mappings in working memory while tracing lines — and working memory holds about four items. The chart is unreadable by construction, before any accessibility issue.

Step 2 — Apply the colour-vision check. With twelve colours from a default palette, several pairs become indistinguishable under red–green deficiency. A reader with deuteranopia does not see a slightly harder chart; they see two lines merge into one.

Step 3 — Apply the greyscale test. Save the figure and convert:

```
from PIL import Image
Image.open("twelve_lines.png").convert("L").save("twelve_lines_grey.png")
```

If the message vanishes, colour was carrying the entire load. It usually is.

Step 4 — Fix by changing the encoding, not the palette. Three options, in order of preference:

```
# Option A — small multiples. Position does the work; colour does none.
fig, axes = plt.subplots(3, 4, figsize=(12, 7), sharex=True, sharey=True)
for ax, (course, g) in zip(axes.flat, weekly.groupby("course_code")):
    ax.plot(g.iso_week, g.logins, color="#0072B2")
    ax.set_title(course, fontsize=9)

# Option B — highlight one, grey the rest. Answers "how does DS100 compare?"
for course, g in weekly.groupby("course_code"):
    focus = course == "DS100"
    ax.plot(g.iso_week, g.logins,
            color="#0072B2" if focus else "#CCCCCC",
            lw=2.2 if focus else 0.9, zorder=3 if focus else 1)

# Option C — aggregate to the groups that matter, then use 2–3 colours
weekly["group"] = np.where(weekly.course_code.isin(hard), "Quantitative", "Other")
```

Step 5 — Choose using the question. Option A is right for “how does each course behave?” Option B is right for “how does DS100 compare?” Option C is right for “do quantitative courses differ?” The chart cannot be fixed without knowing which was being asked — which is the real diagnosis. Twelve undifferentiated lines is what a chart looks like when nobody decided on the question.

> **Faded variant.** Build a scatter plot of entry_score against mean_grade with points coloured by programme and shaped by study_mode. Convert to greyscale

and identify precisely which comparison is lost. Then redesign so the greyscale version carries the same message as the colour one.

Worked example 7.3 — Writing the message title

You have plotted mean weekly logins and the line slopes downward.

Step 1 — State what the chart shows, precisely. Never eyeball a trend.

```
w = activity.groupby("iso_week").logins.mean()
print(w.round(3).to_string())

early, late = w.loc[1:4].mean(), w.loc[11:14].mean()
print(f"weeks 1–4: {early:.2f}, weeks 11–14: {late:.2f}, "
    f"change: {(late/early - 1):+.1%}")
# weeks 1–4: 4.73, weeks 11–14: 4.41, change: -6.8%
```

Now check a second variable before building any story on the first:

```
s = activity.groupby("iso_week").submissions.mean()
print(f"submissions change: {(s.loc[11:14].mean()/s.loc[1:4].mean() - 1):+.1%}")
# submissions change: +5.0%
```

Logins fall; submissions do not. Whatever is happening, it is not a general withdrawal of effort.

Step 2 — Draft three titles at increasing strength.

Title	Type	Problem
“Mean logins by week”	Descriptive	Cannot be wrong; cannot be informative
“Logins fall 6.8% across the term”	Message	Defensible — *if* the confound below is handled
“Students disengage in the second half of term”	Interpretation	Contradicted by flat submissions

Step 3 — Find the confound before defending even the second title. The denominator is not constant. Students withdraw during the term, so the set of people averaged in week 13 is not the set averaged in week 1:

```
print(activity.groupby("iso_week").student_id.count().to_string())
# 1   1200
# ...
# 14  1062
```

The active cohort shrinks by 138 students. If the students who leave were *above* average, their departure would create an apparent decline with no behaviour change at all — a survivorship effect. So the raw trend confounds two things.

Step 4 — Separate them. Restrict to students who never withdrew:

```
stay = outcomes.loc[~outcomes.withdrew, "student_id"]
w_adj = (activity[activity.student_id.isin(stay)]
         .groupby("iso_week").logins.mean())
print(f"change among stayers: "
      f"{(w_adj.loc[11:14].mean()/w_adj.loc[1:4].mean() - 1):+.1%}")
# change among stayers: -9.5%
```

The decline is *larger*, not smaller — 9.5% against 6.8% in the raw series. Withdrawing students were below average, so their departure was masking part of the drop. The finding survives, and it survived a check that could have destroyed it.

Step 5 — Choose the strongest title the evidence supports.

> **"Logins fall 9.5% across the term among students who never withdrew"**

The qualifying clause is not hedging. It is the confound you ruled out, stated where the reader can see it — and it is what makes the claim survive the first person who checks. Note also what the title does *not* say: not "disengage", because submissions held steady, and a student logging in less while submitting the same amount may simply have got more efficient.

Step 6 — Annotate rather than caption. Put the evidence for the title inside the figure: plot both the raw and the stayers-only series, show the shrinking cohort on a secondary axis, and label the two endpoint means directly. A reader who sees the claim and its support in the same visual field does not have to trust you.

The rule. A message title is a claim, and every claim on the ladder needs evidence at its rung. A title that says "fall" is descriptive and needs a number *and* a check that the denominator held still. A title that says "disengage" is an interpretation and needs the proxy defended. A title that says "because of X" is causal and almost always needs a design you do not have.

> **Faded variant.** Write message titles for three figures from your Lab 6 report. For each, identify the rung it claims and name the specific evidence in the figure that supports it. Rewrite any title whose rung exceeds its evidence.

In code: a reusable honest plot

```
import matplotlib.pyplot as plt

OKABE_ITO = ["#0072B2", "#E69F00", "#009E73", "#D55E00", "#CC79A7", "#56B4E9"]
```

```
plt.rcParams.update({
   "figure.dpi": 150, "font.size": 10.5,
   "axes.titlesize": 12, "axes.titleweight": "bold",
   "axes.spines.top": False, "axes.spines.right": False,
   "axes.grid": True, "grid.alpha": 0.25,
})

def bar_honest(series, ylabel, title, ymax=None):
   """Bar chart with an enforced zero baseline and a message title."""
   fig, ax = plt.subplots(figsize=(7, 3.6))
   ax.bar(series.index.astype(str), series.values, color=OKABE_ITO[0], width=0.6)
   ax.set_ylim(0, ymax or series.max() * 1.15)      # zero baseline, always
   ax.set_ylabel(ylabel)
   ax.set_title(title)
   for x, v in zip(range(len(series)), series.values):
      ax.text(x, v * 1.02, f"{v:.1f}", ha="center", fontsize=9)
   return fig, ax

means = per_student.groupby("programme").mean_grade.mean().sort_values()
bar_honest(means, "mean final grade",
          "Mean grades differ by under 2 marks across programmes", ymax=100)
plt.show()
```

Common mistakes

Mistake	Consequence	Prevention
Truncated bar baseline	Visual difference unrelated to data	ax.set_ylim(0, ...) on every bar chart
Colour as the only channel	Chart fails for ~8% of male readers	Greyscale test before shipping
Unlabelled error bars	Ambiguous by a factor of ~10	State SD, SE, or CI in the caption
Dual y-axes	Any correlation can be manufactured	Use two panels
Descriptive titles	Reader must find the point themselves	State the finding, at the right rung

Chapter summary

Charts are arguments. Form follows question; encoding follows perception, with position and length beating angle and area. A truncated baseline is a rhetorical choice requiring disclosure, and for bars it is simply wrong. Colour must never be the sole channel, and the greyscale test settles the matter in seconds. Uncertainty belongs in the figure, labelled.

Message titles make findings contestable, which is the point — but a title's claim must sit on a rung the evidence reaches.

Check your understanding

1. Why must bar charts start at zero when line charts often need not?

2. Name two problems with encoding twelve categories by colour.

3. What is a message title, and when is one inappropriate?

4. A chart shows means with error bars and no caption. Why can you not interpret it?

> **Model answers are in the Reference volume, Part 1.** Write your own answer first — comparing a finished attempt against a model answer teaches far more than reading the model answer cold.

Lab work — Lab 6: Chart critique and redesign

> The full two-hour version of this lab, with timings, is **Lab 6** in the Lab Manual.

Deliverable: notebook containing three critiques and three redesigns.

1. Find three misleading charts — from news, marketing, or a published report. Save each.
2. For each, name the specific technique (truncated baseline, area encoding, dual axes, colour-only encoding, cherry-picked range) and estimate how much it exaggerates.
3. Redesign each honestly using the course data or the original where available.
4. Apply the greyscale test to all three redesigns. Fix any that fail.
5. Write a message title for each redesign, and state which rung it claims and what evidence supports it.
6. Produce one figure showing uncertainty explicitly, with a caption stating exactly what the interval represents.

Chapter 8 — Probability and Bayesian Reasoning

> *Probability is how we reason about what we do not know, and the direction of a conditional probability is easy to reverse by accident.*

After this chapter you can: apply the rules of probability; distinguish $P(A|B)$ from $P(B|A)$; use Bayes' rule with natural frequencies; and simulate to check an analytical answer.

8.1 Probability as a model of uncertainty

A probability is a number between 0 and 1 describing how strongly we expect an event. Two readings coexist: the **frequentist** one — the long-run proportion in repeated trials — and the **subjective** one — a degree of belief. Both obey the same rules, and for our purposes the rules matter more than the interpretation.

8.2 The rules you need

For events A and B:

- **Complement:** $P(\text{not } A) = 1 - P(A)$
- **Addition:** $P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B)$
- **Multiplication:** $P(A \text{ and } B) = P(A) \times P(B \mid A)$
- **Independence:** A and B are independent iff $P(A \mid B) = P(A)$, in which case $P(A \text{ and } B) = P(A)P(B)$

The multiplication rule is the one to internalise, because it is the general case. Independence is a special case, and assuming it when it does not hold is a common and consequential error.

8.3 Conditional probability has a direction

$P(A \mid B)$ and $P(B \mid A)$ are different numbers. Confusing them has a name — the **base-rate fallacy** — and it is the single most consequential probabilistic error in applied work.

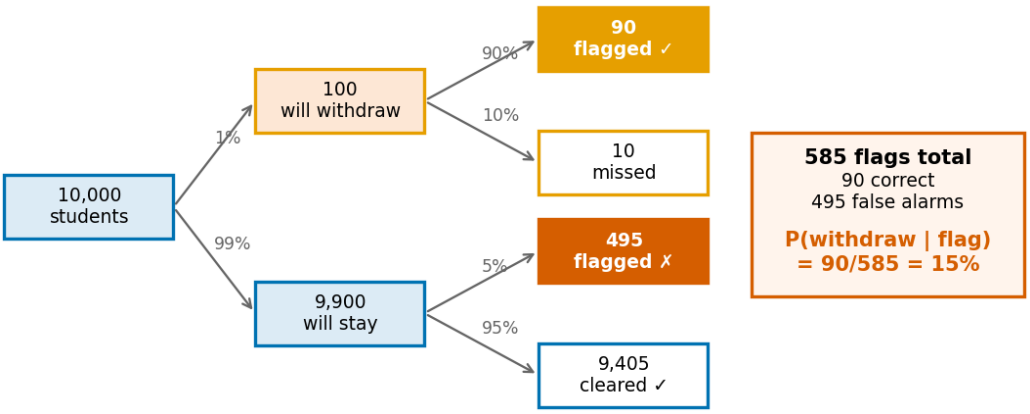
$$P(\text{positive test} \mid \text{disease}) \neq P(\text{disease} \mid \text{positive test})$$

The first is a property of the test. The second is what the patient wants to know, and it depends on how common the disease is.

8.4 Bayes' rule

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

The formula is correct and hard to reason with. **Natural frequencies are easier and less error-prone**: take a concrete population, split it by the true condition, apply the conditional rates within each branch, and count.



Work with counts, not probabilities.

8.5 Simulation

When an analytical answer is hard or you want to check one, simulate.

```
import numpy as np
rng = np.random.default_rng(42)

# P(at least one six in four rolls)
rolls = rng.integers(1, 7, size=(200_000, 4))
print(((rolls == 6).any(axis=1)).mean())        # 0.5177

# Analytical: 1 - (5/6)^4
print(1 - (5/6)**4)                             # 0.5177
```

Agreement confirms the arithmetic. It does not confirm that the model is appropriate — a wrong model simulated faithfully gives a wrong answer with great precision. Always set a seed.

Worked example 8.1 — Why a 90%-accurate flag is wrong five times in six

The university proposes an early-warning flag. Historically 1% of students withdraw *in the first month*. The flag catches 90% of students who will withdraw (sensitivity 0.90) and correctly clears 95% of those who will not (specificity 0.95).

An adviser sees a flag. What is the probability this student will withdraw?

Most people answer “about 90%.” Work it with counts.

Step 1 — Take a concrete population. 10,000 students.

Step 2 — Split by what actually happens.

- Will withdraw: $1\% \times 10,000 = 100$
- Will stay: $9,900$

Step 3 — Apply the flag within each branch.

- Of the 100 who withdraw, the flag catches 90%: $0.90 \times 100 = 90$ true positives
- Of the 9,900 who stay, it wrongly fires on 5%: $0.05 \times 9,900 = 495$ false positives

	Flagged	Not flagged	Total
Will withdraw	90	10	100
Will stay	495	9,405	9,900
Total	**585**	**9,415**	**10,000**

Step 4 — Answer the question that was asked.

$$P(\text{withdraws} \mid \text{flagged}) = \frac{90}{90+495} = \frac{90}{585} = 0.154$$

Step 5 — Confirm with Bayes' rule.

$$P(W \mid F) = \frac{P(F \mid W)P(W)}{P(F \mid W)P(W) + P(F \mid \neg W)P(\neg W)} = \frac{0.90 \times 0.01}{0.90 \times 0.01 + 0.05 \times 0.99}$$

$$= \frac{0.009}{0.009 + 0.0495} = \frac{0.009}{0.0585} = 0.154$$

```
prevalence, sens, spec = 0.01, 0.90, 0.95
tp = sens * prevalence
fp = (1 - spec) * (1 - prevalence)
print(f"P(withdraws | flagged) = {tp / (tp + fp):.3f}")    # 0.154
```

Step 6 — Interpret. About 15% of flagged students will actually withdraw. Five flags in six are false alarms — not because the test is bad, but because withdrawal is rare. There are 99 times more students who stay than leave, so a 5% error rate on the large group swamps a 90% hit rate on the small one.

Why the intuition fails. 0.90 is $P(\text{flag} \mid \text{withdraws})$. The adviser needs $P(\text{withdraws} \mid \text{flag})$. Different quantities; Bayes connects them.

What follows for design. At 15% precision, this flag cannot justify any automatic consequence. It *can* justify an offer of support: 585 outreach emails to reach 90 of 100 at-risk students is a defensible use of adviser time, and the 495 contacted unnecessarily receive an offer they can ignore. The same 585 flags used to deny course registration would harm 495 students who were going to be fine.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

Limitation. All three inputs are estimates from past cohorts. In our actual dataset the full-term withdrawal rate is 11.5%, not 1%. Recompute at that base rate:

$$P(W \mid F) = \frac{0.90 \times 0.115}{0.90 \times 0.115 + 0.05 \times 0.885} = \frac{0.1035}{0.1035 + 0.0443} = 0.700$$

Precision jumps from 15% to 70% — the *same test*, on a less rare outcome. Base rate is not a detail; it is most of the answer.

> **Faded variant.** Recompute at prevalence 5% with the same sensitivity and specificity. Predict *before calculating* whether precision will be closer to 15% or 70%, and roughly where. Then find the prevalence at which precision reaches exactly 50%, and state what that means for anyone proposing a "majority-confidence" threshold.

Worked example 8.2 — Conditional direction in a sentence

A journalist writes: *"90% of students who withdrew had low week-1 engagement. Low engagement is therefore a strong warning sign."*

Step 1 — Identify which conditional was measured. "Of students who withdrew, 90% had low engagement" is

$$P(\text{low engagement} \mid \text{withdrew}) = 0.90$$

Step 2 — Identify which the conclusion needs. "A strong warning sign" means that seeing low engagement should substantially raise concern:

$$P(\text{withdrew} \mid \text{low engagement}) = ?$$

These are different, and the second is not derivable from the first alone.

Step 3 — Get the missing ingredients from the data.

```
d = outcomes.merge(students, on="student_id")
d["low_engagement"] = d.early_logins <= d.early_logins.quantile(0.25)

print("P(withdrew):", d.withdrew.mean().round(3))
print("P(low engagement):", d.low_engagement.mean().round(3))
print("P(low | withdrew):",
    d[d.withdrew].low_engagement.mean().round(3))
print("P(withdrew | low):",
    d[d.low_engagement].withdrew.mean().round(3))
```

The data give:

```
P(withdrew):          0.115
P(low engagement):    0.256
```

P(low | withdrew): 0.464
P(withdrew | low): 0.208

Step 4 — Do the arithmetic explicitly. Bayes converts one direction to the other:

$$P(\text{withdrew} \mid \text{low}) = \frac{P(\text{low} \mid \text{withdrew})\,P(\text{withdrew})}{P(\text{low})} = \frac{0.464 \times 0.115}{0.256} = \frac{0.0534}{0.256} = 0.208$$

So about 21%. Even taking the journalist's framing at face value — suppose the figure really were 90% rather than the 46% in our data — the conversion would give $0.90 \times 0.115 / 0.256 = 0.404$, still well under half. The sentence invites the reader to carry 90% across, and 90% is not the number for anything the reader cares about.

Step 5 — Note the asymmetry's source. The gap comes entirely from the base rates. Low engagement is common (25.6% of students) and withdrawal is rare (11.5%). A common symptom of a rare outcome is necessarily a weak individual predictor, no matter how reliably the outcome produces the symptom. Flagging every low-engagement student would identify 21% correctly and misidentify 79%.

The general form. Whenever you read "X% of [outcome group] had [feature]," ask two questions before believing anything follows: how common is the outcome, and how common is the feature? Without both, the sentence constrains nothing.

Responsible-use note. This error is not neutral in its effects. A 90%-sounding warning sign attached to a 41%-real risk will be over-trusted by whoever acts on it, and the students wrongly flagged are disproportionately those whose circumstances — employment, caring responsibilities, illness — reduce early engagement for reasons unrelated to withdrawal.

> **Faded variant.** A report states: *"Two-thirds of students who failed MA110 had not submitted the week-3 assignment."* Compute what you would need to know to convert this into a usable warning sign, then compute it from the data. State the answer as a probability with its direction written out in words.

Worked example 8.3 — Checking an answer by simulation

Four students independently have a 20% chance of missing a deadline. What is the probability that at least two miss it?

Step 1 — Analytically. Let $X \sim \text{Binomial}(n = 4, p = 0.2)$. We want $P(X \geq 2) = 1 - P(X = 0) - P(X = 1)$.

$$P(X = 0) = \binom{4}{0}(0.2)^0(0.8)^4 = 1 \times 1 \times 0.4096 = 0.4096$$

$$P(X = 1) = \binom{4}{1}(0.2)^1(0.8)^3 = 4 \times 0.2 \times 0.512 = 0.4096$$

$$P(X \geq 2) = 1 - 0.4096 - 0.4096 = 0.1808$$

Step 2 — By simulation.

```
rng = np.random.default_rng(7)
trials = rng.random((200_000, 4)) < 0.20
print((trials.sum(axis=1) >= 2).mean())       # 0.18098
```

Step 3 — Confirm with the exact distribution.

```
from scipy import stats
print(1 - stats.binom.cdf(1, n=4, p=0.2))     # 0.1808
```

Three methods, one answer. The agreement rules out arithmetic error.

Step 4 — Now break the assumption that matters. "Independently" is doing enormous work. Suppose the four students are in a study group, so a shared cause — a clashing deadline, an unclear brief — makes them miss together. Model it as a shared shock:

```
rng = np.random.default_rng(7)
n = 200_000
shared = rng.random(n) < 0.10                 # group-level disruption
p_ind = np.where(shared, 0.60, 0.156)         # keeps marginal p ≈ 0.20
trials = rng.random((n, 4)) < p_ind[:, None]

print("marginal P(miss):", trials.mean().round(3))      # 0.200
print("P(at least 2):", (trials.sum(axis=1) >= 2).mean())  # 0.239
```

Each student still has a 20% individual chance. But $P(X \geq 2)$ rises from 0.181 to 0.239 — a 32% relative increase — purely from correlation. **Independence was not a technicality; it was most of the answer.**

Why this matters beyond dice. The same structure appears whenever risks are assumed independent and are not: correlated defaults in a loan portfolio, correlated failures in redundant systems, correlated non-response in a survey. In each case the marginal rates look fine and the joint behaviour is far worse than modelled.

Limitation. Simulation checks arithmetic against a model. It cannot tell you the model is right. Both simulations above are internally correct and they disagree, because they encode different assumptions about the world. Choosing between them requires knowing whether the students talk to each other — a question no amount of computation answers.

> **Faded variant.** Simulate the flag from Worked Example 8.1 rather than computing it: generate 100,000 students, assign withdrawal at 1% prevalence, apply the flag with the stated sensitivity and specificity, and estimate P(withdraws | flagged). Confirm it matches 0.154. Then raise the number of simulated students to 10,000,000 and describe how the estimate's *precision* changes, and how its *accuracy* does not.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

Common mistakes

Mistake	Consequence	Prevention
Reversing a conditional	Base-rate fallacy	Write both directions out in words
Ignoring the base rate	Massively overestimates precision on rare events	Always use natural frequencies
Assuming independence	Understates joint risk, often severely	Ask what could make events co-occur
Unseeded simulation	Irreproducible results	default_rng(seed), always
Trusting a simulation's precision	Precision is not accuracy	Vary the model, not just n

Chapter summary

Probability obeys a small set of rules, of which the multiplication rule is the general case and independence a special one. Conditional probabilities have direction, and reversing them is the base-rate fallacy — the most consequential probabilistic error in applied work. Natural frequencies make Bayes' rule tractable: take a population, split by condition, apply rates, count. A rare outcome makes even an accurate test mostly wrong, which is a fact about the base rate rather than the test. Simulation checks arithmetic; it cannot check assumptions.

Check your understanding

1. A test is 99% accurate for a disease affecting 1 in 10,000. You test positive. Roughly what is the probability you have it?

2. Why is $P(A \mid B) \neq P(B \mid A)$?

3. When does $P(A \text{ and } B) = P(A)P(B)$?

4. Your simulation of 10 million trials gives 0.2387. How confident should you be in that number?

> **Model answers are in the Reference volume, Part 1.** Write your own answer first — comparing a finished attempt against a model answer teaches far more than reading the model answer cold.

Lab work — Lab 7: Simulation and Bayesian reasoning

The full two-hour version of this lab, with timings, is **Lab 7** in the Lab Manual.

Deliverable: notebook with analytical and simulated answers side by side.

1. Build the screening table from Worked Example 8.1 as a function of prevalence, sensitivity, and specificity.

2. Plot precision against prevalence from 0.1% to 50%, holding sensitivity and specificity fixed. Mark the point where precision reaches 50%.
3. Compute the empirical withdrawal base rate from outcomes.csv and evaluate the flag at that rate.
4. Verify every analytical answer by simulation with a fixed seed. Report both.
5. Take one result and break its independence assumption. Quantify how much the answer moves.
6. Write a short note advising the retention team what threshold could justify an automatic action, if any, and why.

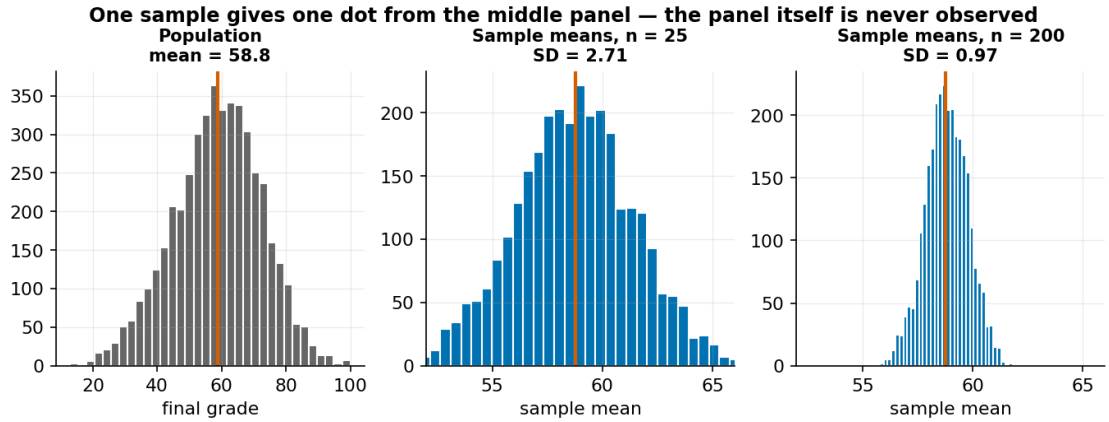
Chapter 9 — Statistical Inference and Experiments

> *An interval says what the data can support. A p-value says less than most people think it does.*

After this chapter you can: explain sampling variability; construct and interpret confidence intervals; run and read hypothesis tests; report effect sizes; and design a simple A/B test.

9.1 From sample to population

Any statistic computed from a sample would have come out differently with a different sample. The distribution of a statistic across hypothetical repeated samples is the **sampling distribution**, and all of inference is reasoning about it from the one sample you actually have.



Quadrupling the sample size halves the spread of the sample mean. That √n relationship is what a standard error measures.

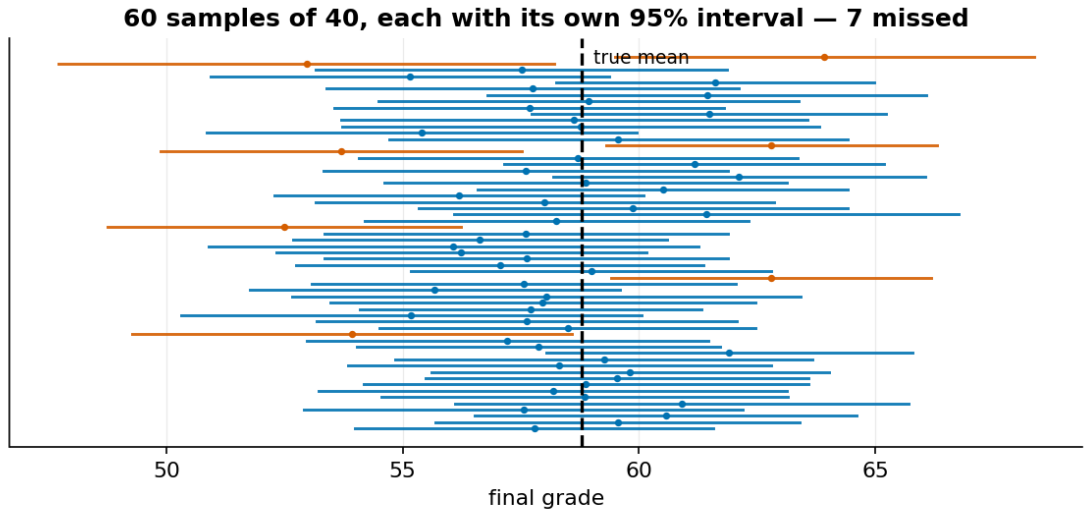
One sample gives one dot from the middle panel. The panel itself is never observed.

The **standard error** is the standard deviation of the sampling distribution. For a mean it is $s / \sqrt{n}$ — which is why quadrupling the sample halves the standard error, and why the returns to more data diminish.

9.2 Confidence intervals

A 95% confidence interval is constructed by a procedure that captures the true parameter in 95% of repeated samples.

$$\acute{x} \pm t^* \frac{s}{\sqrt{n}}$$



"95% confident" describes the procedure across repeated samples, not the probability that this one interval is right.

Sixty samples, sixty intervals. The failures are the point.

The "95%" describes the **procedure across repetitions**, not this particular interval. Your one interval either contains the true value or does not. What you can say is that it was produced by a method that works 95% of the time.

Report the interval before the test. It contains everything the p-value does, plus the magnitude and the units.

9.3 Hypothesis tests without ritual

A test asks: if the null hypothesis were true, how surprising would this data be?

- **Null hypothesis** (H_0): the effect is zero.
- **p-value:** $P(\text{data at least this extreme} \mid H_0 \text{ true})$.
- A small p-value means the data are unlikely under H_0.

What a p-value is not: the probability that H_0 is true; the probability the result is due to chance; a measure of effect size; or a measure of importance. It is a conditional probability of the data given a hypothesis, and the direction — as in Chapter 8 — is the thing people reverse.

9.4 Effect size, power, and multiple testing

Effect size measures magnitude independent of sample size. Cohen's *d* for two means:

$$d = \frac{\acute{x}_1 - \acute{x}_2}{s_{\text{pooled}}}$$

Conventionally 0.2 is small, 0.5 medium, 0.8 large — though these thresholds are arbitrary and field-dependent.

Power is the probability of detecting an effect that exists. Underpowered studies produce two failure modes, and the second is less well known: they miss real effects, *and* the effects they do detect are systematically overestimated, because only unusually large sample effects clear the significance threshold.

Multiple testing. Twenty independent tests at $\alpha = 0.05$ give roughly a 64% chance of at least one false positive: $1 - 0.95^{20} = 0.64$. Correct for it (Bonferroni, Benjamini–Hochberg) or pre-register which comparison is primary.

9.5 Experiments and A/B tests

Randomisation is what licenses causal claims. It makes treatment groups comparable *in expectation on every variable*, including the ones you never measured — which is something no amount of statistical adjustment on observational data can achieve.

A minimum A/B design specifies, **before collecting data**: the unit of randomisation, the primary outcome (one), the minimum effect worth detecting, the sample size implied by that effect and desired power, the analysis method, and the stopping rule.

The stopping rule matters more than people expect. Checking results repeatedly and stopping when $p < 0.05$ inflates the false-positive rate dramatically — with continuous monitoring it approaches 100%.

Worked example 9.1 — Interval first, test second

Do full-time and part-time students differ in mean grade?

Step 1 — Get the estimates.

```
per_student = (enrolments.groupby("student_id").final_grade.mean()
               .rename("mean_grade").reset_index()
               .merge(students[["student_id", "study_mode"]], on="student_id"))

A = per_student[per_student.study_mode == "Full-time"].mean_grade
B = per_student[per_student.study_mode == "Part-time"].mean_grade
print(f"Full-time: mean {A.mean():.2f}, sd {A.std():.2f}, n {len(A)}")
print(f"Part-time: mean {B.mean():.2f}, sd {B.std():.2f}, n {len(B)}")
```

 # Full-time: mean 58.87, sd 7.87, n 1027
 # Part-time: mean 58.45, sd 8.20, n 173

Difference: 0.42 marks.

Step 2 — Build the interval for the difference.

$$SE = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} = \sqrt{\frac{7.87^2}{1027} + \frac{8.20^2}{173}} = \sqrt{0.0603 + 0.3887} = \sqrt{0.4490} \approx 0.670$$

$$0.42 \pm 1.96 \times 0.670 = 0.42 \pm 1.31 = [-0.89, 1.74]$$

```
from scipy import stats
res = stats.ttest_ind(A, B, equal_var=False)
diff = A.mean() - B.mean()
se = np.sqrt(A.var(ddof=1)/len(A) + B.var(ddof=1)/len(B))
print(f"difference {diff:.2f}, 95% CI [{diff-1.96*se:.2f}, {diff+1.96*se:.2f}]")
print(f"t = {res.statistic:.3f}, p = {res.pvalue:.4f}")
# difference 0.42, 95% CI [-0.89, 1.74]
# t = 0.633, p = 0.5275
```

Step 3 — Read the interval first. It runs from −0.89 to 1.74 marks. It contains zero, so the data are consistent with no difference. More usefully, **both endpoints are small**: even the most favourable value in the interval is under two marks, on a 100-point scale, against a within-group SD of about 8.

Step 4 — Effect size.

$$s_{\text{pooled}} = \sqrt{\frac{7.87^2 + 8.20^2}{2}} = \sqrt{\frac{61.9 + 67.2}{2}} = \sqrt{64.6} \approx 8.04$$

$$d = \frac{0.42}{8.04} \approx 0.05$$

Cohen's *d* of 0.05 is well below the conventional "small" threshold of 0.2.

Step 5 — Now, and only now, the p-value. p = 0.53. Unsurprising given the interval, and it adds nothing the interval had not already said.

The claim:

> **Observation.** Mean grade was 58.87 for full-time students (n = 1,027) and 58.45 for part-time (n = 173). The difference of 0.42 marks has a 95% confidence interval of [−0.89, 1.74] and Cohen's *d* = 0.05.
>
> **Interpretation.** The data are consistent with no difference in mean grade by study mode, and rule out any difference larger than about 1.8 marks — a quantity too small to matter for any decision the university would make.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

Limitation. This is a comparison of averages across all courses and therefore inherits the composition problem of Chapter 6. It also does not address pass rates, where the within-course picture differs sharply.

Why the interval beats the p-value here. "p = 0.53, not significant" is compatible with two very different situations: a genuinely null effect, or a study too small to detect anything. The interval distinguishes them. It is narrow, so this is the first case. Had n been 30 per group, the same difference would have given an interval like [−6, 7] — equally "not significant" and meaning something entirely different.

Faded variant. Compute the same interval for entry_score by first_generation. Before calculating, predict whether the interval will be wider or narrower than the one above, using only the group sizes. Then check.

Worked example 9.2 — Statistically significant and practically irrelevant

The platform team A/B tests a new dashboard layout. With 40,000 users per arm, mean session time rises from 24.0 to 24.3 minutes, p = 0.002.

Step 1 — The test is correct. With $s \approx 12$ minutes and $n = 40,000$ per arm:

$$SE = \sqrt{\frac{12^2}{40000} + \frac{12^2}{40000}} = \sqrt{0.0036 + 0.0036} = \sqrt{0.0072} \approx 0.0849$$

$$t = \frac{0.3}{0.0849} \approx 3.53 \; p \approx 0.0004$$

Genuinely significant. Nothing is wrong with the statistics.

Step 2 — Effect size.

$$d = \frac{0.3}{12} = 0.025$$

That is 1/8th of the conventional "small" threshold. In practical terms: 18 seconds.

Step 3 — See what n did. Hold the effect fixed and vary the sample:

```
import numpy as np
from scipy import stats

diff, sd = 0.3, 12.0
for n in [100, 1_000, 10_000, 40_000, 100_000]:
    se = np.sqrt(2 * sd**2 / n)
    t = diff / se
    p = 2 * (1 - stats.norm.cdf(abs(t)))
    print(f"n={n:7,}  SE={se:.4f}  t={t:5.2f}  p={p:.4f}")
# n=    100  SE=1.6971  t= 0.18  p=0.8597
```

```
# n=  1,000  SE=0.5367  t= 0.56  p=0.5762
# n= 10,000  SE=0.1697  t= 1.77  p=0.0770
# n= 40,000  SE=0.0849  t= 3.53  p=0.0004
# n=100,000  SE=0.0537  t= 5.59  p=0.0000
```

The effect never changed. Only the sample size did. With enough data, any non-zero difference becomes statistically significant — which is why "significant" is not a synonym for "important", and why a p-value alone cannot support a decision.

Step 4 — Ask the decision question. Is 18 seconds worth a redesign? That depends on engineering cost, on whether session time is even the right outcome, and on whether 18 more seconds on a dashboard benefits anyone. **None of these are statistical questions**, and the p-value speaks to none of them.

Step 5 — Define the threshold before the test, not after. The discipline that prevents this is stating a **minimum detectable effect of interest** in advance. If the team had written "we will adopt the new layout if it increases session time by at least 2 minutes," the result would read: observed 0.3 minutes, 95% CI [0.13, 0.47], entirely below the 2-minute threshold — a clear *negative* result, despite p = 0.0004.

Responsible-use note. Session time is a proxy for engagement, and a poor one. A layout that increases time-on-page by confusing users would score identically to one that increases it by being useful. Optimising a metric that is only loosely coupled to the goal is how systems end up working against the people they serve.

> **Faded variant.** The same test with n = 200 per arm gives p = 0.81. Compute the confidence interval and state precisely what it rules out. Then explain why "no significant difference" would be a misleading summary of that result, and what you would write instead.

Worked example 9.3 — Designing the A/B test properly

The retention team wants to test whether a personalised week-3 email reduces withdrawal. Design it before collecting anything.

Step 1 — Unit of randomisation. The student. Randomising by tutor group would risk contamination — students talk — and would require accounting for clustering, which reduces effective sample size.

Step 2 — One primary outcome. Withdrawal before end of term, a binary. Everything else (open rate, logins, satisfaction) is secondary and will not be used to declare success.

Step 3 — Minimum effect worth detecting. Baseline withdrawal is 11.5%. The team judges that a reduction to 9.5% — two percentage points, about a sixth of withdrawals — would justify running the programme. That is the MDE.

Step 4 — Sample size. For two proportions at 80% power and $\alpha = 0.05$:

$$n \text{ per arm} \approx \frac{\left(z_{\alpha/2} + z_{\beta}\right)^2 \left[p_1\left(1 - p_1\right) + p_2\left(1 - p_2\right)\right]}{\left(p_1 - p_2\right)^2}$$

With $z_{0.025} = 1.96$, $z_{0.20} = 0.84$, $p_1 = 0.115$, $p_2 = 0.095$:

$$n \approx \frac{(2.80)^2\left[0.115(0.885) + 0.095(0.905)\right]}{(0.02)^2} = \frac{7.84 \times \left[0.1018 + 0.0860\right]}{0.0004} = \frac{7.84 \times 0.1878}{0.0004} \approx 3,681$$

```
from statsmodels.stats.power import NormalIndPower
from statsmodels.stats.proportion import proportion_effectsize

es = proportion_effectsize(0.115, 0.095)
n = NormalIndPower().solve_power(effect_size=es, alpha=0.05, power=0.80,
                                 alternative="two-sided")
print(f"n per arm: {np.ceil(n):.0f}")    # ~3,700
```

Step 5 — Confront the result. About 3,700 per arm, so 7,400 students. **We have 1,200.** The study as designed cannot be run.

This is the single most valuable moment in the whole process, and it happens before any data are collected. The options are:

- **Accept a larger MDE.** With n = 600 per arm, the detectable effect is roughly 11.5% → 6.5% — a halving of withdrawals. Is that plausible from one email? Almost certainly not.
- **Run across multiple cohorts** to accumulate sample.
- **Choose a more sensitive outcome.** Continuous outcomes need far smaller samples than rare binary ones. Week 5–8 engagement might be measurable at n = 1,200.
- **Do not run it.** An underpowered test of a real effect will probably return "no significant difference," and that result will be misread as evidence the email does not work.

Step 6 — Stopping rule, fixed in advance. Analyse once, at end of term, at the pre-specified n. No peeking. If interim analysis is genuinely needed, use a sequential design with an alpha-spending function — not repeated naive tests.

Step 7 — Ethics review before randomisation. Half the students are being withheld a support intervention the university believes might help. That requires justification: genuine equipoise (we do not know it works), no withdrawal of existing support, and a plan to roll out to everyone if it succeeds.

> **Faded variant.** Recompute the required sample size for an MDE of 3 percentage points (11.5% → 8.5%) and for 90% power. Then find the MDE that n = 600 per arm can detect at 80% power, and write two sentences advising the team whether to proceed.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

Common mistakes

Mistake	Consequence	Prevention
Reporting p without an interval	Loses magnitude and units	Interval first, always
"Not significant" = "no effect"	Confuses absence of evidence with evidence of absence	Report what the interval rules out
Significance = importance	Any effect is significant at large n	State an MDE before testing
Peeking and stopping at $p < 0.05$	False-positive rate approaches 1	Fix n and the stopping rule in advance
Many tests, no correction	~64% false positive across 20 tests	Pre-register the primary outcome

Chapter summary

Inference reasons about a sampling distribution that is never observed. A confidence interval describes the long-run behaviour of a procedure, and it carries magnitude and units that a p-value discards — so report it first. A p-value is the probability of the data given the null, not the probability of the null given the data. Significance and importance are independent: at large n every non-zero effect is significant, and at small n a real effect may not be. Randomisation is what licenses causal claims, and a power calculation done before data collection often reveals that the study cannot answer its question — which is worth knowing then rather than afterwards.

Check your understanding

1. Interpret a 95% confidence interval of [−0.89, 1.74] for a difference in means.

2. Why can a tiny effect be highly significant?

3. What does a p-value of 0.03 mean, precisely?

4. A team runs 20 comparisons and reports the two with $p < 0.05$. What is wrong?

> **Model answers are in the Reference volume, Part 1.** Write your own answer first — comparing a finished attempt against a model answer teaches far more than reading the model answer cold.

Lab work — Lab 8: Bootstrap, permutation, and power

> The full two-hour version of this lab, with timings, is **Lab 8** in the Lab Manual.

Deliverable: notebook comparing analytical and resampling methods.

1. Compute a 95% CI for mean final_grade three ways: t-interval, bootstrap percentile, and bootstrap BCa. Compare and explain any differences.

2. Test the study-mode grade difference with a permutation test written from scratch — shuffle labels 10,000 times and locate the observed difference in the null distribution.
3. Compare your permutation p-value to scipy.stats.ttest_ind. Explain what assumptions each makes.
4. Reproduce the coverage figure: draw 200 samples of n = 40, build an interval for each, and count how many contain the population mean.
5. Run a power analysis for the week-3 email A/B test. Produce a curve of detectable effect against sample size and mark where n = 1,200 falls.
6. Write a one-page memo advising the retention team, including a recommendation on whether to run the test at all.

Chapter 10 — Regression and the Machine-Learning Workflow

> *A model is only good relative to a baseline, and only trustworthy relative to data it has never seen.*

After this chapter you can: build a baseline; fit and interpret linear regression; choose evaluation metrics; diagnose residuals; and use cross-validation to estimate generalisation.

10.1 Prediction is a different task

Chapters 3 to 9 described and explained. Prediction asks a different question: given features available *now*, what will the outcome be? The success criterion changes accordingly. A model can have every coefficient interpretable and predict badly, or predict well and be uninterpretable. Decide which you need before you start.

10.2 Always start with a baseline

A model that beats nothing has not been shown to be useful. The baseline for regression is predicting the training mean for every observation.

```
from sklearn.dummy import DummyRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error

ds = enrolments[enrolments.course_code == "DS100"].merge(students, on="student_id")
X, y = ds[["entry_score"]], ds.final_grade
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.3, random_state=0)

base = DummyRegressor(strategy="mean").fit(X_tr, y_tr)
print(f"baseline MAE: {mean_absolute_error(y_te, base.predict(X_te)):.2f}")
# baseline MAE: 8.88
```

Any model reported without this number is uninterpretable. “MAE of 7.98” means nothing until you know the baseline is 8.88.

10.3 Linear regression

$$\hat{y} = \beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k$$

Ordinary least squares chooses the β minimising the sum of squared residuals. Squaring makes the problem solvable in closed form and makes large errors count disproportionately — a modelling choice with consequences, not a neutral default.

```
from sklearn.linear_model import LinearRegression

lm = LinearRegression().fit(X_tr, y_tr)
print(f"intercept {lm.intercept_:.2f}, coefficient {lm.coef_[0]:.3f}")
# intercept 29.22, coefficient 0.517
print(f"model MAE: {mean_absolute_error(y_te, lm.predict(X_te)):.2f}")   # 7.98
```

Interpreting the coefficient. 0.517 means: comparing two students whose entry scores differ by one mark, the model predicts a difference of 0.517 in final grade. It does **not** mean raising a student's entry score by one mark would raise their final grade by 0.517 — that is the causal rung, and this is observational data.

10.4 Metrics

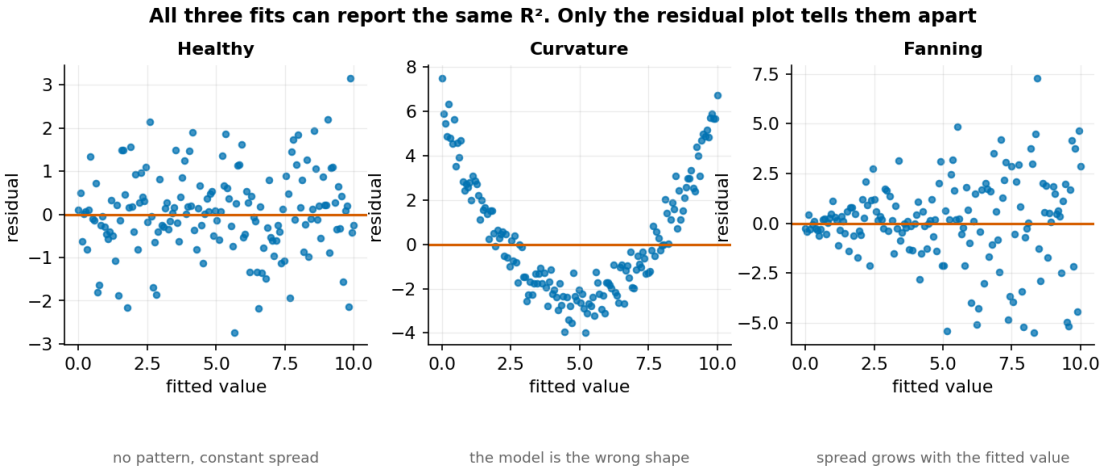
Metric	Formula	Units	Behaviour
MAE	mean|y − ŷ|	Original	Robust; treats all errors proportionally
RMSE	√mean(y − ŷ)²	Original	Penalises large errors heavily
R^2	1 − SS_res/SS_tot	None	Proportion of variance explained

RMSE ≥ MAE always, with equality only when every error is identical. The ratio is a quick diagnostic: a large gap means a few big errors dominate.

R^2 has a specific trap: it is **not** a measure of whether the model is correct or useful. A model can have $R^2 = 0.9$ and be badly misspecified, and $R^2 = 0.17$ while being genuinely informative for a decision.

10.5 Residual diagnosis

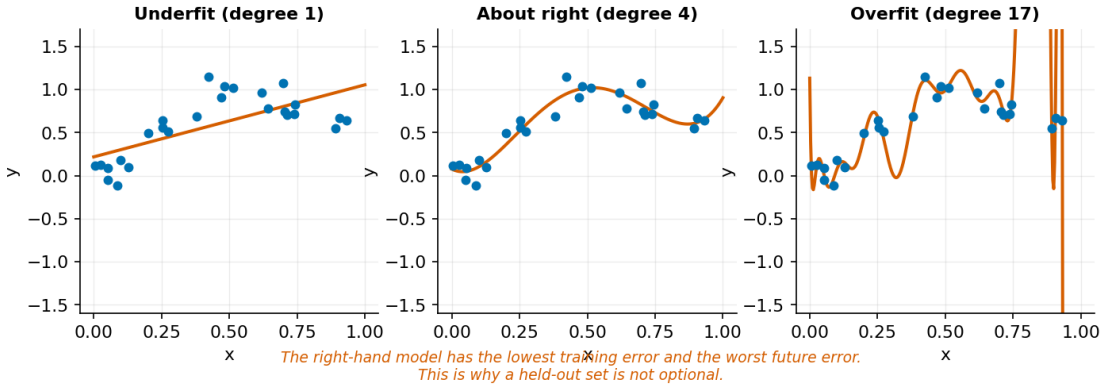
Residuals ($y - \hat{y}$) carry the information the model failed to capture. Plot them against fitted values.



All three fits can report the same R^2. Only the residual plot distinguishes them.

- **Healthy:** no pattern, constant spread.
- **Curvature:** the model's functional form is wrong. Add a term or change the model.
- **Fanning (heteroscedasticity):** spread grows with the fitted value. Coefficients remain unbiased but standard errors are wrong, so intervals and p-values mislead.

10.6 Generalisation and cross-validation



The lowest training error and the worst future error.

Overfitting is learning the training set's noise. It always produces excellent training performance, which is why training performance is not evidence of anything.

k-fold cross-validation splits the data into k parts, trains on k - 1 and tests on the remaining one, and rotates. It uses all data for both roles and gives a variance estimate as a bonus.

```
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

pipe = make_pipeline(StandardScaler(), LinearRegression())
```

```
scores = cross_val_score(pipe, X, y, cv=5, scoring="neg_mean_absolute_error")
print(f"CV MAE: {-scores.mean():.2f} (±{scores.std():.2f})")
```

Report the standard deviation. A model scoring 7.9 ± 0.2 is a different proposition from one scoring 7.9 ± 2.1, and the mean alone hides the difference.

Worked example 10.1 — Baseline first

Predict DS100 final grade from entry score. Do it in the order that keeps you honest.

Step 1 — Baseline. Predict the training mean for everyone.

```
y_tr.mean()        # 63.31
```

Errors are therefore $|y_i - 63.31|$, averaged: MAE 8.88.

Step 2 — Model.

$$\hat{y} = 29.22 + 0.517 \times \text{entry\_score}$$

For a student with entry score 70: $29.22 + 0.517\,|70| = 29.22 + 36.19 = 65.41$.

```
print(lm.predict([[70]]))     # [65.41]
```

Step 3 — Compare, and quantify the improvement honestly.

	MAE	R^2
Baseline (mean)	8.88	0.000
Linear regression	7.98	0.173

$$\text{improvement} = \frac{8.88 - 7.98}{8.88} = 10.1\,\%$$

Step 4 — Decide whether 10% is worth anything. This is the step that is usually skipped. The model reduces typical error from about 8.9 marks to 8.0. For a decision like "which students should be offered tutoring", a 0.9-mark improvement in predicted grade is unlikely to change who gets selected. For a decision like "how many marks of headroom should we build into the assessment", it might.

R^2 = 0.173 says entry score explains 17% of the variance in DS100 grades. **83% is something else** — teaching, effort, prior subject knowledge, circumstances, luck. That number is not a failure of the model; it is a finding about how weakly entry qualifications determine outcomes, and it is arguably more useful than the prediction.

Step 5 — Say what the model must not be used for. A prediction of 65.4 for a student with entry score 70 comes with a residual standard deviation of about 10 marks. An individual prediction interval spans roughly ±20 marks. Using this to tell a student what grade to expect would be indefensible; using it to estimate the *cohort's* mean is fine.

Faded variant. Add early_logins and early_submissions as predictors. Report baseline MAE, single-predictor MAE, and multi-predictor MAE. Compute the improvement over baseline for each. Then answer: at what point would you stop adding predictors, and on what evidence?

Worked example 10.2 — What a coefficient does and does not mean

Fit a two-predictor model and interpret it carefully.

```
X2 = ds[["entry_score", "early_logins"]]
lm2 = LinearRegression().fit(X2, ds.final_grade)
for name, c in zip(X2.columns, lm2.coef_):
    print(f"{name:18s} {c:+.4f}")
```

Step 1 — State the interpretation with all its conditions. The coefficient on entry_score means: among students *with the same early_logins*, a one-mark difference in entry score is associated with the stated difference in final grade.

The clause "with the same early_logins" is not decoration. It changes the quantity being estimated. A coefficient in a multiple regression is a **partial** association, holding the other included variables fixed.

Step 2 — Watch the coefficient move. Fit entry score alone, then with logins added:

```
solo = LinearRegression().fit(ds[["entry_score"]], ds.final_grade).coef_[0]
both = lm2.coef_[0]
print(f"entry_score alone: {solo:.4f}, with logins: {both:.4f}")
```

The coefficient shrinks. Nothing about the data changed; the *question* changed. The first asks "how do grades differ across entry scores?" The second asks "how do grades differ across entry scores among students who log in equally often?" Since entry score and logins are correlated (r = 0.40), part of what the first attributed to entry score is reassigned.

Step 3 — Recognise the trap this creates. Adding a variable that lies *on the causal path* removes exactly the effect you wanted to measure. If better-prepared students log in more, and logging in improves grades, then controlling for logins removes part of entry score's real influence. Whether to include a variable is a question about the causal structure, and no amount of model-fitting answers it.

Step 4 — Compare magnitudes correctly. Raw coefficients are in the units of their variables, so they are not comparable. Standardise first:

```
from sklearn.preprocessing import StandardScaler
Z = StandardScaler().fit_transform(X2)
lm_z = LinearRegression().fit(Z, ds.final_grade)
for name, c in zip(X2.columns, lm_z.coef_):
    print(f"{name:18s} {c:+.3f} per SD")
```

Now each coefficient reads as "change in grade per one-SD change in the predictor", and the two can be ranked.

Step 5 — Refuse the causal reading explicitly. In a report, write: *"Among students with similar early engagement, a 10-mark difference in entry score is associated with a X-mark difference in final grade."* Not: *"Raising entry score by 10 marks raises final grade by X."* The second sentence describes an intervention nobody performed.

> **Faded variant.** Add study_mode (one-hot encoded) to the model. Report how the entry_score coefficient changes. Then decide whether study_mode is a confounder, a mediator, or a collider for the entry-score-to-grade relationship, and say what that implies about including it.

Worked example 10.3 — MAE and RMSE disagree

Five test predictions:

Actual	Predicted	Error	|Error|	Error²
60	62	−2	2	4
55	57	−2	2	4
70	68	+2	2	4
65	66	−1	1	1
80	60	+20	20	400

Step 1 — MAE.

$$\text{MAE} = \frac{2+2+2+1+20}{5} = \frac{27}{5} = 5.4$$

Step 2 — RMSE.

$$\text{RMSE} = \sqrt{\frac{4+4+4+1+400}{5}} = \sqrt{\frac{413}{5}} = \sqrt{82.6} \approx 9.09$$

Step 3 — Read the gap. RMSE is 68% larger than MAE. That gap is diagnostic: it says a small number of large errors dominate. Remove the fifth row and the two converge:

$$\text{MAE} = \frac{7}{4} = 1.75 \quad \text{RMSE} = \sqrt{\frac{13}{4}} = 1.80$$

Almost identical. **The RMSE/MAE ratio is a cheap outlier detector for your predictions.**

```
import numpy as np
y  = np.array([60, 55, 70, 65, 80])
yh = np.array([62, 57, 68, 66, 60])
```

```
mae  = np.abs(y - yh).mean()
rmse = np.sqrt(((y - yh) ** 2).mean())
print(f"MAE {mae:.2f}  RMSE {rmse:.2f}  ratio {rmse/mae:.2f}")
# MAE 5.40  RMSE 9.09  ratio 1.68
```

Step 4 — Choose by the cost of errors, not by convention.

- If being wrong by 20 is exactly ten times as bad as being wrong by 2, **MAE** matches your loss.
- If being wrong by 20 is catastrophically worse — a student predicted to pass who fails badly, a bridge, a dose — **RMSE** matches.
- If you cannot say which, you have not yet defined the decision the model serves, and no metric is defensible.

Step 5 — Note what this implies about fitting. Least squares *minimises* squared error, so it is already optimising RMSE. If MAE is the right loss for your problem, ordinary regression is fitting the wrong objective, and quantile or robust regression is the correct tool. Choosing the evaluation metric after fitting is a common inconsistency.

> **Faded variant.** Compute MAE and RMSE for the DS100 model on the test set. Report the ratio and identify the three largest residuals. Then state whether those three are errors in the data, unusual students, or a signal the model is missing something — and how you would tell.

Common mistakes

Mistake	Consequence	Prevention
Reporting a score with no baseline	Number is uninterpretable	Always fit DummyRegressor first
Evaluating on training data	Overfitting invisible	Held-out set or cross-validation
Causal language for coefficients	Overclaims by two rungs	“Associated with”, and name what is held fixed
Choosing the metric after seeing results	Metric-shopping	State the metric before fitting
Reporting CV mean without SD	Hides instability	Always report the spread

Chapter summary

Prediction is a different task from explanation and needs a different success criterion. A baseline makes a score interpretable, and without one no number means anything. Regression coefficients are partial associations conditional on the other included variables, so adding a variable changes the question rather than improving the answer. MAE and RMSE encode different beliefs about the cost of large errors, and the gap between them is

diagnostic. Cross-validation estimates generalisation; the standard deviation across folds is part of the result, not an optional extra.

Check your understanding

1. Why fit a baseline before any real model?

2. Interpret the coefficient 0.517 on entry_score.

3. RMSE is much larger than MAE. What does that tell you?

4. Your model gets $R^2 = 0.95$ on training data and 0.31 on test data. Diagnose it.

> **Model answers are in the Reference volume, Part 1.** Write your own answer first — comparing a finished attempt against a model answer teaches far more than reading the model answer cold.

Lab work — Lab 9: The full regression workflow

> The full two-hour version of this lab, with timings, is **Lab 9** in the Lab Manual.

Deliverable: notebook with a model-selection record.

1. Predict DS100 final_grade. Split into train and test once, at the start, and do not touch the test set until step 6.
2. Fit DummyRegressor. Record baseline MAE and RMSE.
3. Fit linear regression with one, then three, then all available predictors. Cross-validate each. Report mean and SD.
4. Produce residual-versus-fitted plots for your best model. Diagnose any pattern.
5. Keep a model-selection record: every model tried, its CV score, and why you kept or rejected it.
6. Evaluate your single chosen model on the test set, once. Report the difference from CV and explain it.
7. Write three sentences on whether this model should be used for any decision about an individual student.

Chapter 11 — Classification I: Building a Classifier

> *A classifier outputs a probability. Turning that probability into a decision is a separate act, and it is where the ethics live.*

After this chapter you can: explain and fit logistic regression; interpret coefficients as log-odds and odds ratios; fit k-nearest-neighbours and a decision tree; and compare all three against a baseline.

This chapter is new in the second edition. The first edition evaluated classifiers it never showed you how to build.

11.1 Why linear regression fails for classification

Predict withdrew (0 or 1) with linear regression and two things break. The predictions are unbounded — you will get −0.3 and 1.4, which are not probabilities. And the errors are not remotely normal, so every interval and p-value is wrong.

We need a function mapping any real number into (0, 1).

11.2 The logistic function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

z	$\sigma(z)$
−4	0.018
−2	0.119
0	0.500
2	0.881
4	0.982

It is S-shaped, symmetric about $z = 0$, and asymptotes at 0 and 1 without reaching them.

Logistic regression fits a linear expression and passes it through σ:

$$P(y = 1 \mid x) = \sigma(\beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k)$$

The model is linear *in the log-odds*, not in the probability. Rearranging:

$$\log \frac{p}{1 - p} = \beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k$$

The left side is the **log-odds** or **logit**. This is what makes the coefficients interpretable.

11.3 Fitting

There is no closed-form solution. The coefficients are found by maximising the likelihood — equivalently, minimising **log loss**:

$$L = -\frac{1}{n} \sum_i \left[y_i \log \hat{p}_i + (1 - y_i) \log (1 - \hat{p}_i) \right]$$

Log loss punishes confident wrong predictions severely: predicting 0.99 when the truth is 0 contributes $-\log(0.01) = 4.6$, while predicting 0.6 wrongly contributes only 0.92. This is a feature — it forces the model to be calibrated rather than merely decisive.

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
```

```
from sklearn.model_selection import train_test_split

d = outcomes.merge(students, on="student_id")
features = ["early_logins", "early_submissions", "early_minutes",
            "entry_score", "age_at_entry"]
X, y = d[features], d.withdrew.astype(int)

X_tr, X_te, y_tr, y_te = train_test_split(
    X, y, test_size=0.3, random_state=0, stratify=y)

clf = make_pipeline(StandardScaler(),
                    LogisticRegression(max_iter=1000)).fit(X_tr, y_tr)
p = clf.predict_proba(X_te)[:, 1]
```

stratify=y matters when a class is rare. Without it, a random split can leave very different positive rates in train and test, and the comparison becomes noise.

11.4 Reading the coefficients

```
lr = clf.named_steps["logisticregression"]
for name, c in zip(features, lr.coef_[0]):
    print(f"{name:20s} {c:+.3f}   odds ratio {np.exp(c):.3f}")
# early_logins         -0.196   odds ratio 0.822
# early_submissions    -0.422   odds ratio 0.656
# early_minutes        -0.329   odds ratio 0.720
# entry_score          -0.461   odds ratio 0.631
# age_at_entry         +0.030   odds ratio 1.030
print("intercept:", round(lr.intercept_[0], 3))     # -2.414
```

Because we standardised, each coefficient is **per one standard deviation** of its feature.

Odds ratio = e^{β}. For early_submissions, $e^{-0.422} = 0.656$: a one-SD increase in early submissions multiplies the *odds* of withdrawal by 0.656, a 34% reduction in odds.

Odds are not probability. Odds = $p / (1 - p)$. At $p = 0.5$ the odds are 1; at $p = 0.9$ they are 9. A 34% reduction in odds is a smaller reduction in probability, and how much smaller depends on where you start.

The intercept of −2.414 is the log-odds at the mean of every standardised feature: $\sigma(-2.414) = 0.082$, close to the 11.5% base rate.

11.5 Two comparators

k-nearest-neighbours predicts from the k most similar training points. No training phase, no assumed functional form, and it requires scaling — distance is the entire model.

Decision trees split recursively on single features. Naturally handle interactions and non-linearity, need no scaling, and overfit enthusiastically unless depth is limited.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import cross_val_score

models = {
    "baseline (majority)": DummyClassifier(strategy="most_frequent"),
    "logistic regression": make_pipeline(StandardScaler(),
                                         LogisticRegression(max_iter=1000)),
    "k-NN (k=15)":         make_pipeline(StandardScaler(),
                                         KNeighborsClassifier(n_neighbors=15)),
    "decision tree (d=4)": DecisionTreeClassifier(max_depth=4, random_state=0),
}
for name, m in models.items():
    auc = cross_val_score(m, X, y, cv=5, scoring="roc_auc")
    print(f"{name:22s} ROC-AUC {auc.mean():.3f} (±{auc.std():.3f})")
```

Note that the baseline scores 0.500 on ROC-AUC by construction — a model with no discrimination. But on **accuracy** it scores 0.885, because 88.5% of students do not withdraw. Which is the subject of the next chapter.

Worked example 11.1 — Logistic regression by hand

A student has standardised features: early_logins = −1.5, early_submissions = −1.2, early_minutes = −1.0, entry_score = −0.8, age_at_entry = +0.5. What is their predicted withdrawal probability?

Step 1 — Compute the linear predictor.

$$z = -2.414 + (-0.196)(-1.5) + (-0.422)(-1.2) + (-0.329)(-1.0) + (-0.461)(-0.8) + (0.030)(0.5)$$

Term by term:

Feature	β	x	βx
intercept	—	—	−2.414
early_logins	−0.196	−1.5	+0.294
early_submissions	−0.422	−1.2	+0.506
early_minutes	−0.329	−1.0	+0.329
entry_score	−0.461	−0.8	+0.369

Feature	β	x	βx
age_at_entry	+0.030	+0.5	+0.015
		z =	**−0.901**

Step 2 — Apply the logistic function.

$$p = \frac{1}{1+e^{0.901}} = \frac{1}{1+2.462} = \frac{1}{3.462} = 0.289$$

Step 3 — Verify.

```
import numpy as np
z = -2.414 + np.dot([-0.196, -0.422, -0.329, -0.461, 0.030],
                    [-1.5, -1.2, -1.0, -0.8, 0.5])
print(f"z = {z:.3f}, p = {1/(1+np.exp(-z)):.3f}")   # z = -0.901, p = 0.289
```

Step 4 — Interpret against the base rate. 28.9% against a cohort base rate of 11.5% — this student's estimated risk is about 2.5 times average. Every negative feature is below average, and each pushes z upward.

Step 5 — Note what the model does *not* say. It does not say this student will withdraw; 71% of students with this profile will not. It does not say why. And it does not say what to do — that requires a threshold, which is Chapter 12.

Step 6 — Trace the largest contribution. early_submissions contributes +0.506, the biggest single term. Without it, $z = -1.407$ and $p = 0.197$. One feature moves the estimate by 9 percentage points, which tells you where to look if the prediction is challenged — and a student is entitled to challenge it.

> **Faded variant.** Compute p for a student with all standardised features at 0. Confirm it equals $\sigma(\text{intercept})$ and explain why. Then find the value of standardised early_submissions, holding all else at 0, that would push the predicted probability to exactly 0.50, and say whether such a student plausibly exists in the data.

Worked example 11.2 — Three models against a baseline

Which classifier should the retention team use?

Step 1 — Establish the baseline properly. There are two, and they answer different questions.

```
maj = DummyClassifier(strategy="most_frequent").fit(X_tr, y_tr)
print("majority-class accuracy:", (y_te == 0).mean().round(3))   # 0.886
```

Predicting "nobody withdraws" achieves **88.6% accuracy** and identifies zero at-risk students. Any model must beat this on accuracy to be worth anything — and beating it on accuracy is nearly impossible while also being useful. That tension is the whole problem.

Step 2 — Fit and compare on a discrimination metric.

```
for name, m in models.items():
    auc = cross_val_score(m, X, y, cv=5, scoring="roc_auc")
    ap  = cross_val_score(m, X, y, cv=5, scoring="average_precision")
    print(f"{name:22s} AUC {auc.mean():.3f}  PR-AUC {ap.mean():.3f}")
```

Cross-validated on the full dataset:

Model	ROC-AUC	PR-AUC
Baseline (majority)	0.500	0.115
Logistic regression	**0.727**	**0.284**
k-NN (k = 15)	0.632	0.177
Decision tree (depth 4)	0.652	0.224

Step 3 — Read the numbers honestly. ROC-AUC of 0.727 is *modest*. It means that given one student who withdraws and one who does not, the model ranks them correctly about 73% of the time — better than the 50% of a coin, well short of anything decisive.

PR-AUC of 0.284 against a baseline of 0.115 is roughly a 2.5-fold improvement. That is the more informative comparison when positives are rare, and it is the number to report. Note that logistic regression leads on both metrics, and that k-NN — the most flexible of the three in principle — performs worst.

Step 4 — Prefer the simpler model when scores are close. Logistic regression wins narrowly, and it wins decisively on grounds the AUC does not capture:

- **Explicability.** You can tell a student exactly which features drove their flag and by how much. A k-NN flag means "fifteen students similar to you did this", which is not a reason a person can act on or contest.
- **Calibration.** Logistic regression outputs probabilities that mean something. Trees at depth 4 produce a handful of distinct values; k-NN with k = 15 produces multiples of 1/15.
- **Stability.** Check it — refit with different random seeds and see which model's coefficients move.

Step 5 — Ask whether any of them should be deployed. At AUC 0.727, roughly three in ten pairs are ranked wrongly. Whether that is good enough depends entirely on what happens after a flag. For "send a supportive email offering a tutorial", plainly yes. For anything with a cost to the student, plainly no.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

Faded variant. Tune k for k-NN across 3, 5, 15, 30, 60 using cross-validation. Plot AUC against k. Identify where underfitting and overfitting begin, and explain why very large k converges toward the baseline.

Worked example 11.3 — Where the decision boundary sits

Simplify to two features so the boundary can be drawn.

Step 1 — Fit.

```
X2 = d[["early_submissions", "entry_score"]]
clf2 = make_pipeline(StandardScaler(),
                     LogisticRegression(max_iter=1000)).fit(X2, y)
lr2 = clf2.named_steps["logisticregression"]
b0 = lr2.intercept_[0]
b1, b2 = lr2.coef_[0]
```

Step 2 — Derive the boundary algebraically. The model predicts class 1 when $p \geq 0.5$, which happens exactly when $z \geq 0$:

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$$

Solve for x_2:

$$x_2 = -\frac{\beta_0 + \beta_1 x_1}{\beta_2}$$

A straight line. That is the defining property and the defining limitation of logistic regression: in the feature space, the boundary is always a hyperplane. Any curved separation is unreachable without adding transformed features.

Step 3 — Note where the line actually falls. With an intercept near −2.4 and a base rate of 11.5%, the $p = 0.5$ line sits far out in the corner of the feature space. Almost no student crosses it, which is why the threshold-0.5 confusion matrix in Chapter 12 is nearly empty on one side.

Step 4 — Move the threshold and watch the line translate. For threshold t, the boundary is where $z = \log \frac{t}{1-t}$:

$$x_2 = \frac{\log \frac{t}{1-t} - \beta_0 - \beta_1 x_1}{\beta_2}$$

Threshold t	$\log \frac{t}{1-t}$
0.10	−2.197

Threshold t	$\log \frac{t}{1-t}$
0.25	−1.099
0.50	0
0.75	+1.099

Lowering the threshold shifts the line parallel to itself, sweeping more students into the flagged region. **The slope never changes** — only the offset. So changing the threshold cannot change *who is ranked ahead of whom*; it changes only how far down the ranking you cut.

Step 5 — Draw the consequence. Because the ranking is fixed by the model and the cut is fixed by the threshold, the model and the decision are genuinely separable concerns. You can improve the model (better ranking) or change the policy (different cut) independently. Conflating them — "the model decided" — is both technically wrong and a way of avoiding responsibility for a choice someone made.

> **Faded variant.** Add a quadratic term entry_score^2 and refit. Plot the boundary in the original feature space. Explain why it is now curved even though logistic regression remains a linear model, and state what "linear" is actually referring to.

Common mistakes

Mistake	Consequence	Prevention
Linear regression for a binary target	Predictions outside [0,1]	Logistic regression
Reporting accuracy on imbalanced data	88.6% means nothing here	Report against the majority baseline
Interpreting β as a probability change	Off by a large factor	e^{β} is an odds ratio
k-NN without scaling	One feature dominates distance	Always pipeline with a scaler
Unstratified split on a rare class	Train and test differ	stratify=y

Chapter summary

Logistic regression maps a linear predictor through the logistic function to produce a probability, and is linear in the log-odds rather than the probability. Coefficients exponentiate to odds ratios, which are not probability changes. Log loss penalises confident errors and pushes toward calibration. k-NN and shallow trees are useful comparators with different failure modes. On this data all three achieve modest discrimination, and logistic regression is preferred less for its score than for being explicable to the person affected.

The decision boundary is a hyperplane whose position — but never whose orientation — the threshold controls.

Check your understanding

1. Why not use linear regression for a binary outcome?

2. A coefficient is −0.422 on a standardised feature. Interpret it.

3. Your classifier gets 88.6% accuracy. Is it good?

4. Why does k-NN require scaling when a decision tree does not?

> **Model answers are in the Reference volume, Part 1.** Write your own answer first — comparing a finished attempt against a model answer teaches far more than reading the model answer cold.

Lab work — Lab 10: Build three classifiers

> The full two-hour version of this lab, with timings, is **Lab 10** in the Lab Manual (shared with Chapter 12).

Deliverable: notebook with a model comparison table.

1. Build the feature matrix from outcomes.csv and students.csv. Split with stratify=y.
2. Fit DummyClassifier. Report both its accuracy and its ROC-AUC, and explain why they differ so much.
3. Fit logistic regression, k-NN, and a depth-limited tree, each inside a Pipeline.
4. Cross-validate all four on accuracy, ROC-AUC, and average precision. Present as one table.
5. Extract and interpret the logistic coefficients as odds ratios. Rank the features.
6. Hand-compute the predicted probability for one student and verify against predict_proba.
7. Tune k for k-NN and max_depth for the tree. Plot the validation curve for each.
8. Recommend one model in three sentences, including at least one reason that is not its score.

Chapter 12 — Classification II: Errors, Thresholds, and Fairness

> *There is no statistically correct threshold. There is only a defensible one, and defending it is your job, not the model's.*

After this chapter you can: read a confusion matrix; choose among precision, recall, and F1; interpret ROC and precision-recall curves; handle imbalance; assess calibration; and audit a classifier for disparate impact.

12.1 The confusion matrix

	Predicted negative	Predicted positive
Actually negative	True negative (TN)	False positive (FP)
Actually positive	False negative (FN)	True positive (TP)

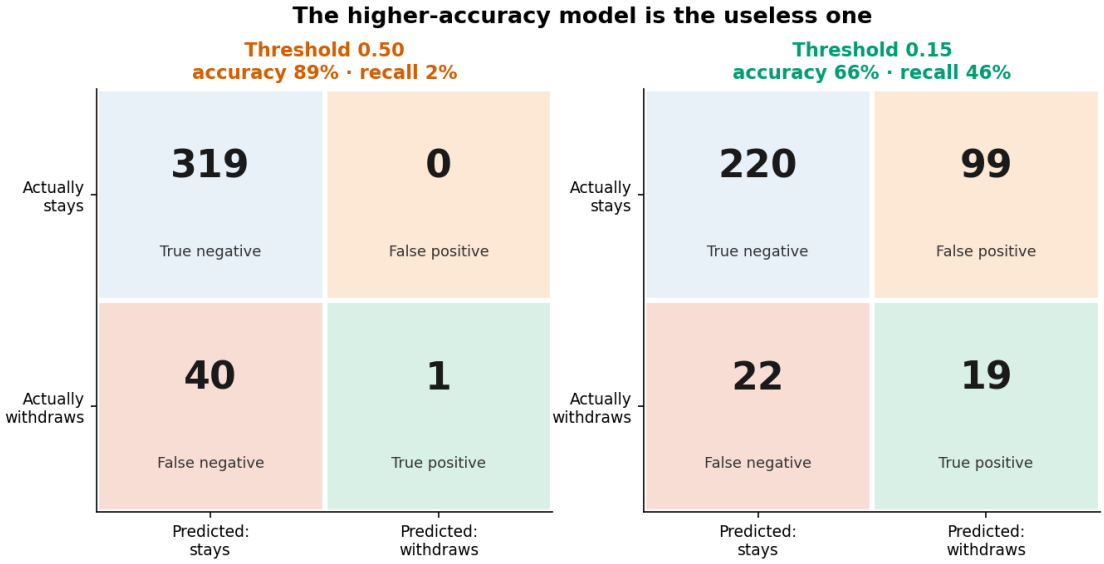
$$\text{Accuracy} = \frac{TP + TN}{\text{total}} \quad \text{Precision} = \frac{TP}{TP + FP} \quad \text{Recall} = \frac{TP}{TP + FN}$$

Precision answers: of those we flagged, how many were right? **Recall** answers: of those we should have found, how many did we find? They trade off, and which matters depends on the cost of each error.

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

F_1 is the harmonic mean, so it is close to the smaller of the two. It implicitly weights precision and recall equally — a choice that is rarely the right one and almost never examined.

12.2 The accuracy trap

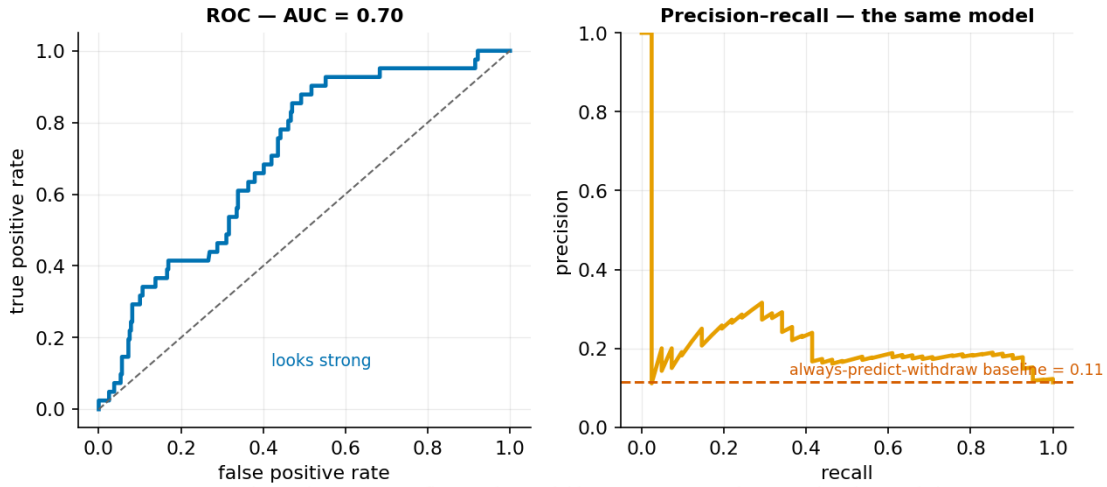


The higher-accuracy model is the useless one.

At threshold 0.50 the model reaches 88.9% accuracy — above the 88.6% majority baseline — by flagging one student out of 360. It finds 1 of 41 students who withdraw. At threshold 0.15 accuracy falls to 67.8% and recall rises to 51%.

The model with worse accuracy is the useful one. Accuracy is the wrong instrument when the positive class is rare, because it is dominated by the majority.

12.3 Threshold curves



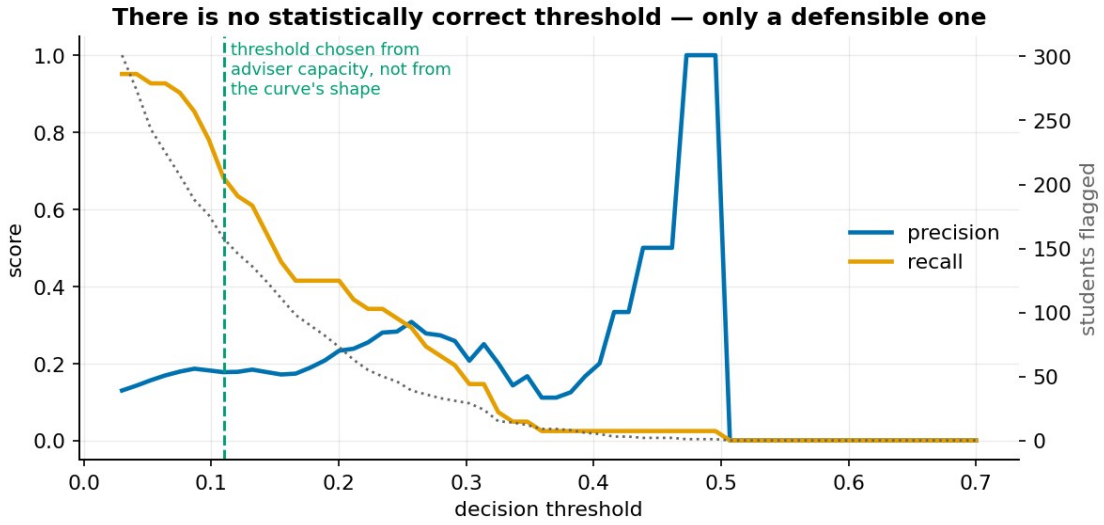
ROC flatters when negatives are abundant.

ROC plots true positive rate against false positive rate. Its area (AUC) is the probability that a randomly chosen positive is ranked above a randomly chosen negative. It is threshold-independent, which makes it good for comparing models and useless for choosing an operating point.

Precision–recall plots precision against recall. Its baseline is the positive rate — 0.115 here, not 0.5.

Use PR when positives are rare. ROC's false-positive rate has a huge denominator (all negatives), so hundreds of false positives barely move it. On the held-out test set this model gives ROC-AUC 0.708 and PR-AUC 0.236; the PR figure against a 0.115 baseline is the truer summary.

12.4 Choosing the threshold



The threshold comes from capacity and error costs, not from the curve.

The threshold is a **policy decision**. Setting it requires knowing:

1. What happens after a positive prediction?
2. What does a false positive cost, and to whom?
3. What does a false negative cost, and to whom?
4. What capacity exists to act on flags?

In our case: a flag triggers an offer of tutoring. A false positive costs an unnecessary email and a little adviser time. A false negative is a student who needed help and was not offered it. The costs are asymmetric, so recall should be favoured — bounded by how many students the advisers can actually contact.

12.5 Imbalance

Options, roughly in order of preference:

1. **Change the threshold.** Usually sufficient, and it changes nothing about the model.
2. **Class weights.** class_weight="balanced" reweights the loss. Affects calibration.
3. **Resampling** (SMOTE, undersampling). Must happen *inside* the cross-validation fold — resampling before splitting is leakage, and a common one.
4. **Collect more positives.** Usually best and usually impossible.

Start at 1. It is free, reversible, and leaves the probabilities interpretable.

12.6 Calibration

A model is **calibrated** if among cases it assigns probability 0.3, about 30% are positive. Discrimination and calibration are independent: a model can rank perfectly and be systematically overconfident.

Calibration matters whenever the probability itself is used — for expected-value calculations, for triage across a queue, or when a human reads "0.7" and acts on it. It does not matter if you only ever threshold.

```
from sklearn.calibration import calibration_curve
true_p, pred_p = calibration_curve(y_te, p, n_bins=8, strategy="quantile")
```

12.7 Fairness and contestability

Aggregate performance can conceal group-level failure. A model with 70% recall overall may have 80% for one group and 45% for another.

Several incompatible definitions of fairness exist, and it is a mathematical result — not a matter of effort — that they cannot generally be satisfied at once:

- **Demographic parity:** equal positive rates across groups.
- **Equal opportunity:** equal recall across groups.
- **Predictive parity:** equal precision across groups.

When base rates genuinely differ between groups, satisfying any two forces violation of the third. You must therefore choose, state the choice, and justify it.

Contestability is the practical safeguard: a person affected must be able to learn they were flagged, see the reason, and challenge it. A model too complex to explain is a model that cannot be contested.

Worked example 12.1 — The accuracy trap, quantified

The retention model reports 88.9% accuracy. Should the team ship it?

Step 1 — Compare against the trivial model.

```
print("majority-class accuracy:", (y_te == 0).mean().round(3))    # 0.886
```

The model beats "predict nobody withdraws" by 0.3 percentage points.

Step 2 — Look at the confusion matrix.

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_te, (p >= 0.50).astype(int)))
# [[319   0]
#  [ 40   1]]
```

	Predicted stays	Predicted withdraws
Actually stays	319	0
Actually withdraws	40	1

The model flagged **one student**. It found 1 of 41 who withdrew.

Step 3 — Compute the metrics that reveal it.

$$\text{Precision} = \frac{1}{1+0} = 1.000 \quad \text{Recall} = \frac{1}{1+40} = 0.024$$

$$F_1 = 2 \cdot \frac{1.000 \times 0.024}{1.000 + 0.024} = 0.047$$

Perfect precision, 2.4% recall. The model achieves its accuracy by almost never making a positive prediction, which is exactly what accuracy rewards on imbalanced data.

Step 4 — Lower the threshold and watch the trade.

```
for t in [0.50, 0.30, 0.15, 0.11]:
    pred = (p >= t).astype(int)
    tn, fp, fn, tp = confusion_matrix(y_te, pred).ravel()
    print(f"t={t:.2f}  flagged={pred.sum():3d}  acc={(tp+tn)/len(y_te):.3f}  "
          f"prec={tp/max(tp+fp,1):.3f}  rec={tp/(tp+fn):.3f}")
# t=0.50  flagged=  1  acc=0.889  prec=1.000  rec=0.024
# t=0.30  flagged= 30  acc=0.853  prec=0.300  rec=0.220
# t=0.15  flagged=117  acc=0.678  prec=0.179  rec=0.512
# t=0.11  flagged=158  acc=0.603  prec=0.177  rec=0.683
```

Step 5 — Read the table as a menu of policies. Accuracy falls monotonically as the model becomes more useful. At t = 0.15 the model finds half the at-risk students, at the cost of contacting 117 to reach 21.

Step 6 — Answer the shipping question. Ship it *with a stated threshold and a stated purpose*, never as "the model". Reporting 88.9% accuracy without the confusion matrix would be, at best, negligent. The number is true and creates a false impression, which is the most common way quantitative reporting misleads.

> **Faded variant.** Compute precision and recall at t = 0.05 and t = 0.40. At which threshold does the number flagged exceed a team of three advisers contacting 40 students each? State the recall achievable within that capacity.

Worked example 12.2 — Metric arithmetic

A confusion matrix from a different run:

	Predicted negative	Predicted positive
Actually negative	220	96
Actually positive	20	24

Compute everything by hand.

Step 1 — Name the cells. TN = 220, FP = 96, FN = 20, TP = 24. Total = 360.

Step 2 — Accuracy.

$$\frac{220+24}{360}=\frac{244}{360}=0.678$$

Step 3 — Precision. Of 120 flagged, how many were right?

$$\frac{24}{24+96}=\frac{24}{120}=0.200$$

Step 4 — Recall. Of 44 who withdrew, how many did we find?

$$\frac{24}{24+20}=\frac{24}{44}=0.545$$

Step 5 — F1.

$$F_1=2\cdot\frac{0.200\times0.545}{0.200+0.545}=2\cdot\frac{0.109}{0.745}=0.293$$

Note F1 = 0.293 sits much closer to precision (0.200) than to recall (0.545). The harmonic mean is dragged toward the smaller value — which is the point, and also why F1 hides which of the two is failing.

Step 6 — Specificity and false positive rate.

$$\text{Specificity}=\frac{220}{220+96}=0.696\quad \text{FPR}=1-0.696=0.304$$

Step 7 — Interpret in the actual decision. Five in six flags are false alarms; we catch just over half of at-risk students; and we contact 120 students to reach 24. Whether that is acceptable depends on the intervention. As an offer of tutoring, defensible — 96 students receive an offer they can decline. As a trigger for a mandatory meeting, indefensible.

Step 8 — Compute the number that matters to a flagged student. A student asks: "I got flagged. What does that mean?" The honest answer is precision: *"One in five students we contact goes on to withdraw. Four in five do not. This is an offer of help, not a prediction about you."*

> **Faded variant.** Suppose you double the flagged count to 240 by lowering the threshold, catching 36 of 44 at-risk students. Recompute precision, recall, F1, and accuracy. Then state which single number you would put in front of a committee, and why.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

Worked example 12.3 — Auditing for disparate impact

Before deploying, check whether the model performs equally across groups.

Step 1 — Choose the groups and justify the choice. study_mode and first_generation are plausible axes of disadvantage, and — critically — neither is a feature in the model. We are auditing outcomes, not using the attribute to decide. This is the distinction drawn in Chapter 1.

Step 2 — Compute per-group metrics at the operating threshold.

```
from sklearn.metrics import precision_score, recall_score

te = d.iloc[y_te.index].copy()
te["p"], te["flag"], te["y"] = p, (p >= 0.15).astype(int), y_te.values

for group in ["study_mode", "first_generation"]:
    print(f"\n--- {group} ---")
    for level, g in te.groupby(group):
        if g.y.sum() == 0:
            print(f"{str(level):16s} n={len(g):3d}  no positives — cannot compute recall")
            continue
        print(f"{str(level):16s} n={len(g):3d}  base={g.y.mean():.3f}  "
              f"flagged={g.flag.mean():.3f}  "
              f"prec={precision_score(g.y, g.flag, zero_division=0):.3f}  "
              f"rec={recall_score(g.y, g.flag):.3f}")
```

Step 3 — Read the three fairness criteria off the output.

- **Demographic parity** asks whether flagged rates are equal.
- **Equal opportunity** asks whether rec (recall) is equal.
- **Predictive parity** asks whether prec is equal.

Step 4 — Observe that they conflict. Suppose part-time students have a higher base rate of withdrawal. Then:

- Equalising *flag rates* under-flags part-time students relative to their risk, so recall is lower for them — they are under-served.
- Equalising *recall* means flagging part-time students at a higher rate, violating demographic parity.
- Equalising *precision* requires yet another threshold.

This is not a bug to be engineered away. It is a theorem: when base rates differ, no classifier can satisfy all three simultaneously except in degenerate cases.

Step 5 — Choose, and state the choice. For an *offer of support*, equal opportunity is the appropriate criterion: every at-risk student should have the same chance of being found, regardless of group. That means accepting unequal flag rates, and saying so.

If the same model gated access to something scarce, the analysis would run differently — and this is the point. **The right fairness criterion depends on what the flag does.** A model is not fair or unfair in isolation.

Step 6 — Check the small-cell problem before believing any of it. With few part-time students in the test set, per-group metrics have wide intervals. Compute them, or bootstrap:

```
n_pt = (te.study_mode == "Part-time").sum()
pos_pt = te.loc[te.study_mode == "Part-time", "y"].sum()
print(f"part-time test students: {n_pt}, of whom {pos_pt} withdrew")
```

If that positive count is in single figures, the group recall estimate is nearly uninformative and reporting it to three decimals would be false precision. Say so rather than reporting the number.

Step 7 — Write the contestability statement. Whatever the audit shows, deployment requires that a flagged student can find out they were flagged, see which features drove it (Worked Example 11.1 shows how), and challenge the inputs. Without that, the audit protects the institution and not the student.

> **Faded variant.** Run the same audit stratified by programme. Identify the group with lowest recall. Then estimate a per-group threshold that would equalise recall, and write three sentences on whether you would recommend deploying group-specific thresholds — including at least one argument against.

Common mistakes

Mistake	Consequence	Prevention
Reporting accuracy on imbalanced data	Hides a useless model	Confusion matrix and PR-AUC
Optimising F1 by default	Assumes precision and recall matter equally	Derive weights from error costs
ROC-AUC on rare positives	Flatters the model	Precision–recall curve
Resampling before splitting	Leakage; inflated scores	Resample inside the fold
Reporting only aggregate metrics	Group failure invisible	Always disaggregate
Per-group metrics on tiny cells	False precision	Report n and an interval, or decline to report

Chapter summary

The confusion matrix is the primary object; every scalar metric is a lossy summary of it. Accuracy is dominated by the majority class and is the wrong instrument for rare outcomes. Precision and recall trade off, and the threshold that sets the trade-off is a policy decision derived from error costs and capacity, not from any curve. ROC flatters when negatives are abundant; precision–recall does not. Aggregate metrics conceal group-level failure, and the several definitions of fairness are provably incompatible when base rates differ — so the criterion must be chosen, stated, and justified against what the flag actually does.

Check your understanding

1. Your model has 88.9% accuracy and 2.4% recall on a rare outcome. Explain how both are true.

2. When should you prefer a precision–recall curve to an ROC curve?

3. How should the threshold be chosen?

4. A model has equal accuracy across two groups. Is it fair?

> **Model answers are in the Reference volume, Part 1.** Write your own answer first — comparing a finished attempt against a model answer teaches far more than reading the model answer cold.

Lab work — Lab 10: Thresholds and a fairness audit

> The full two-hour version of this lab, with timings, is **Lab 10** in the Lab Manual (shared with Chapter 11).

Deliverable: notebook plus a one-page threshold-justification memo.

1. Take your best model from Lab 11. Produce the confusion matrix at thresholds 0.05 to 0.60 in steps of 0.05.
2. Plot precision, recall, F1, and number flagged against threshold on one figure.
3. Plot ROC and precision–recall side by side. Mark the baseline on each. Explain why they tell different stories.
4. Assume three advisers can each contact 40 students. Find the threshold that fits that capacity and report the recall it achieves.
5. Produce a calibration curve. State whether the probabilities can be used directly for triage.
6. Audit by study_mode, first_generation, and programme. Report base rate, flag rate, precision, recall, and n for every group. Flag any cell too small to interpret.
7. Write the memo: recommended threshold, the error costs justifying it, which fairness criterion you chose and why, and how a flagged student can contest the decision.

Chapter 13 — Clustering and Dimensionality Reduction

> *Unsupervised methods always return an answer. Whether the answer corresponds to anything real is a separate question, and the data cannot settle it.*

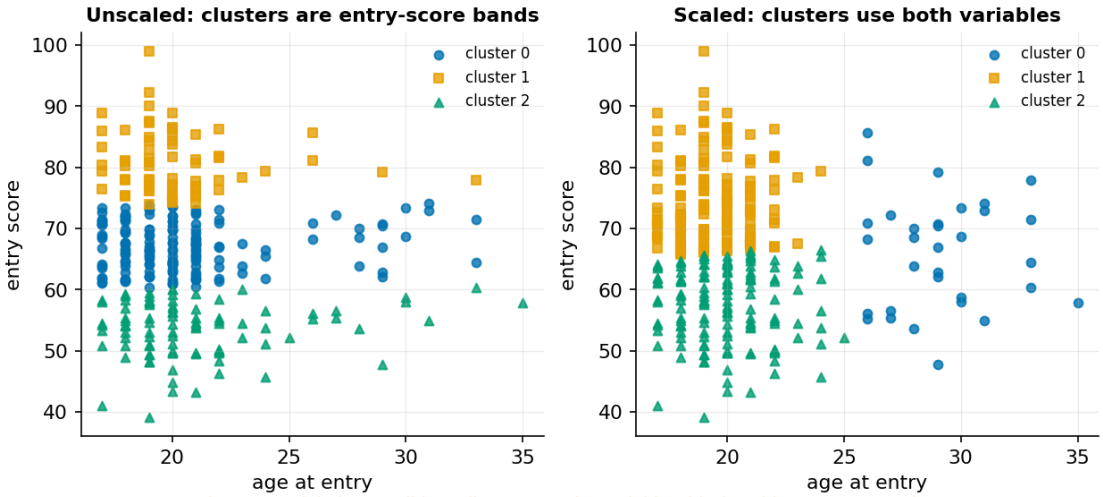
After this chapter you can: apply k-means; choose and defend k; apply PCA; interpret components; and name segments responsibly.

13.1 Learning without labels

Supervised learning has a target and therefore a scorecard. Unsupervised learning has neither. k-means will happily partition uniform random noise into five tidy clusters and report a respectable objective value. **The absence of an error metric is the defining difficulty**, not a detail.

13.2 Distance and scaling

k-means minimises within-cluster sum of squared Euclidean distances. Distance is the whole model, so the scale of each feature is a modelling decision.



k-means minimises Euclidean distance, so the variable with the widest raw range wins. Scaling is not a formality here — it decides the answer.

Unscaled, the clusters are entry-score bands.

Always scale before k-means unless the raw units are genuinely commensurate.

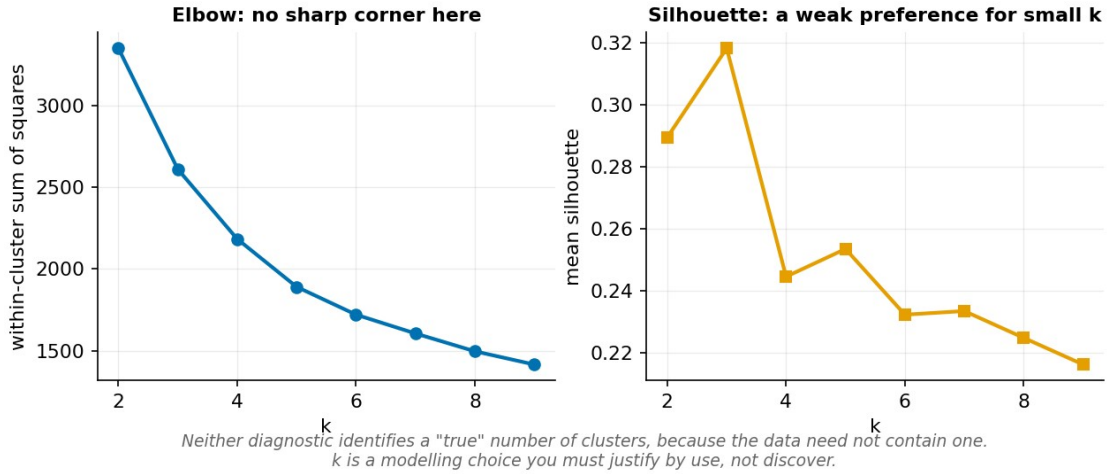
13.3 The k-means algorithm

1. Choose k initial centroids.
2. Assign each point to its nearest centroid.
3. Recompute each centroid as the mean of its members.
4. Repeat until assignments stop changing.

It converges to a **local** optimum that depends on initialisation, which is why n_init runs it repeatedly and keeps the best. Its assumptions — roughly spherical, similarly sized, similarly dense clusters — are strong and frequently violated.

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

cols = ["early_logins", "early_submissions", "entry_score", "age_at_entry"]
Z = StandardScaler().fit_transform(d[cols])
km = KMeans(n_clusters=3, n_init=20, random_state=0).fit(Z)
```

13.4 Choosing k



Neither diagnostic identifies a true k, because the data need not contain one.

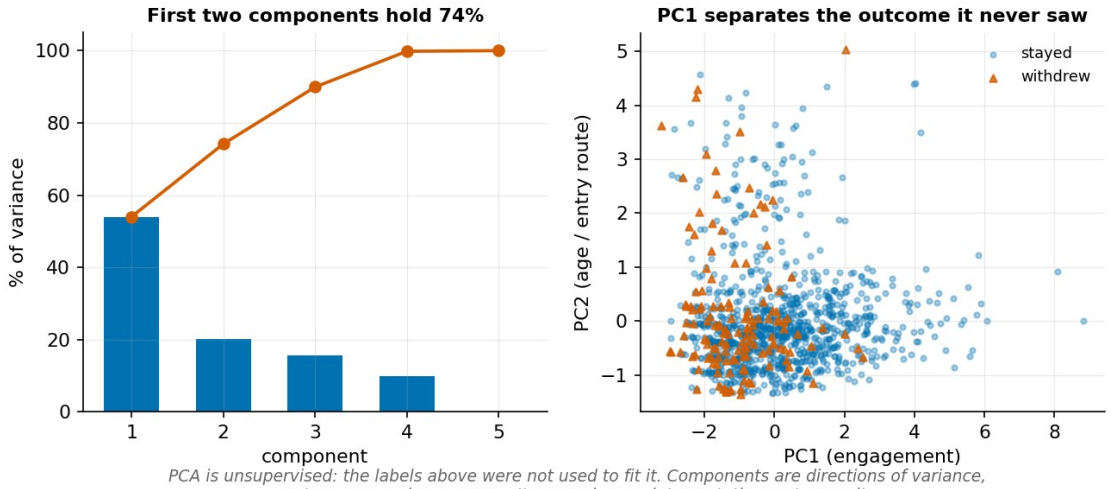
Elbow: plot within-cluster sum of squares against k and look for a bend. It decreases monotonically, so there is always *some* value — often no clear corner.

Silhouette: measures how much closer each point is to its own cluster than to the next nearest, on a scale of −1 to 1. On our data it peaks around 0.32 at k = 3, which is weak-to-moderate structure.

Neither identifies a true k, because the data need not contain clusters at all. k is a **modelling choice justified by use**: if you have capacity for three distinct interventions, k = 3 is defensible whatever the silhouette says.

13.5 Principal component analysis

PCA finds orthogonal directions of maximum variance. The first component is the direction along which the data vary most; each subsequent one is orthogonal to all before it.



PCA is unsupervised: the labels above were not used to fit it. Components are directions of variance, not causes, and a component's name is your interpretation, not a result.

PC1 separates an outcome it never saw.

Uses: visualising high-dimensional data in two dimensions, reducing dimensionality before a distance-based model, and diagnosing correlated features.

Components are directions, not causes. A component's meaning is your interpretation of its loadings, and you should hold it loosely.

```
from sklearn.decomposition import PCA
pca = PCA().fit(Z)
print(pca.explained_variance_ratio_.round(3))     # [0.468 0.252 0.185 0.095]
```

13.6 Naming segments responsibly

A cluster label is a name applied to people, and names have effects. "Cluster 2" is neutral. "At-risk students" is a prediction. "Low-effort students" is an accusation the data cannot support — low engagement is compatible with employment, illness, or caring responsibilities.

Three rules:

1. **Name from the features, not the inferred cause.** "Low early engagement, high entry score" — not "coasting".
2. **Never let a cluster label become a permanent attribute.** Clusters change with k, with scaling, and with a random seed.
3. **Do not use clusters to allocate anything scarce** without validating against an outcome and auditing across groups.

Worked example 13.1 — Scale decides the answer

Cluster students on age and entry score.

Step 1 — Unscaled.

```
X = d[["age_at_entry", "entry_score"]].values.astype(float)
lab_raw = KMeans(3, n_init=20, random_state=0).fit_predict(X)
print(pd.DataFrame(X, columns=["age", "score"]).groupby(lab_raw).mean().round(1))
```

The cluster means differ sharply in entry_score and barely in age.

Step 2 — Work out why, arithmetically. Take two students: (19, 55) and (19, 85) — same age, 30 marks apart. And (19, 55) versus (45, 55) — same score, 26 years apart.

$$d_1 = \sqrt{0^2 + 30^2} = 30 \quad d_2 = \sqrt{26^2 + 0^2} = 26$$

Comparable — but the *ranges* are not. entry_score spans roughly 35 to 99 (64 units) and age_at_entry spans 17 to 47 (30 units). Squared, score contributes about 4.5 times the variance. The algorithm is not choosing to prioritise score; the units are choosing for it.

Step 3 — Scaled.

```
Z2 = StandardScaler().fit_transform(X)
lab_scaled = KMeans(3, n_init=20, random_state=0).fit_predict(Z2)
```

Now clusters separate on both dimensions, and a distinct mature-part-time group appears that the unscaled version dissolved.

Step 4 — Quantify the disagreement.

```
from sklearn.metrics import adjusted_rand_score
print(adjusted_rand_score(lab_raw, lab_scaled).round(3))
```

An ARI well below 1 means the two solutions genuinely disagree about who belongs with whom — **the scaling decision, not the data, produced different groups of people.**

Step 5 — Note that neither is "correct". Scaling asserts that one SD of age matters as much as one SD of entry score. That is an assumption, not a fact. If age genuinely matters less for your purpose, weighting it down is legitimate — provided you say so. What is indefensible is arriving at a weighting by accident, through the units the registry happened to record.

> **Faded variant.** Cluster on early_logins and early_minutes, which are correlated and on very different scales. Compare unscaled, standardised, and min–max scaled solutions with ARI. Then explain why min–max is the most sensitive of the three to a single extreme value.

Worked example 13.2 — Choosing k, and admitting the uncertainty

How many student segments are there?

Step 1 — The framing is wrong. "There are k segments" presupposes discrete groups exist. Test that before assuming it.

Step 2 — Compute both diagnostics.

```
from sklearn.metrics import silhouette_score
for k in range(2, 7):
    km = KMeans(k, n_init=10, random_state=0).fit(Z)
    print(f'k={k}  inertia={km.inertia_:8.1f}  '
          f'silhouette={silhouette_score(Z, km.labels_):.3f}')
# k=2  inertia=  3351.4  silhouette=0.290
# k=3  inertia=  2610.8  silhouette=0.318
# k=4  inertia=  2183.5  silhouette=0.245
# k=5  inertia=  1890.5  silhouette=0.254
# k=6  inertia=  1720.2  silhouette=0.232
```

Step 3 — Read them honestly. Inertia falls smoothly with no sharp corner. Silhouette peaks at k = 3 with 0.318 — which, on the conventional reading, indicates *weak* structure. Values above 0.5 suggest reasonable separation; 0.32 says the clusters overlap substantially.

Step 4 — Test whether the structure is real at all. Run the same diagnostics on data with the same marginals but no cluster structure:

```
rng = np.random.default_rng(0)
Z_null = rng.permutation(Z, axis=0)   # destroys joint structure
for k in [2, 3, 4]:
    km = KMeans(k, n_init=10, random_state=0).fit(Z_null)
    print(f'null k={k}  silhouette={silhouette_score(Z_null, km.labels_):.3f}')
```

If the null silhouettes are close to 0.3, the observed value is not evidence of structure — it is what k-means produces on any dataset of this shape. **This check takes four lines and is almost never done.**

Step 5 — Check stability across seeds.

```
labs = [KMeans(3, n_init=10, random_state=s).fit_predict(Z) for s in range(10)]
aris = [adjusted_rand_score(labs[0], l) for l in labs[1:]]
print(f'pairwise ARI vs seed 0: min {min(aris):.3f}, mean {np.mean(aris):.3f}')
```

If solutions vary across seeds, the "segments" are artefacts of initialisation and should not be given names, let alone used for anything.

Step 6 — Choose k from the intervention, not the curve. Suppose student services can run three distinct programmes. Then k = 3 is defensible on operational grounds — and it happens to coincide with the silhouette peak, which is corroboration rather than justification.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

The honest statement:

> *k-means with k = 3 achieves a mean silhouette of 0.32, indicating weak and overlapping structure. The elbow plot shows no clear inflection. k = 3 was chosen because student services can deliver three distinct interventions, not because the data contain three natural groups. Cluster membership should be treated as an operational grouping, not a property of students.*
>
> **Faded variant.** Run the permutation null test above with 20 repetitions and report the null silhouette distribution. State whether the observed 0.318 falls outside it, and what you would conclude in each case.

Worked example 13.3 — Neutral names

k-means with k = 3 produces these profiles (standardised means):

Cluster	early_logins	early_submissions	entry_score	age_at_entry	n
0	−0.52	−0.61	−0.42	−0.25	639
1	−0.39	+0.04	−0.17	**+2.60**	110
2	+0.83	+0.86	+0.63	−0.28	451

Step 1 — Describe before naming. Cluster 0: below average on engagement and entry score, slightly younger. Cluster 1: average on submissions, slightly below on logins and entry score, and 2.6 SD above average on age. Cluster 2: above average on engagement and entry score.

Step 2 — Reject the tempting names.

Tempting name	Applied to	Why it fails
“High achievers”	Cluster 2	Achievement is not measured. Entry score and logins are inputs.
“Disengaged students”	Cluster 0	Attributes a disposition. Low logins are compatible with employment or illness.
“At-risk students”	Cluster 0	A prediction. This model was fitted without any outcome.
“Mature students”	Cluster 1	Descriptive of age — but it will be read as explanatory.

Step 3 — Name from the observed features.

- Cluster 0: *below-average early engagement, below-average entry score* (n = 639)
- Cluster 1: *older entrants, average submissions* (n = 110)
- Cluster 2: *above-average early engagement, above-average entry score* (n = 451)

Longer and duller. Also unable to smuggle in a claim.

Step 4 — Check what the labels correlate with before anyone acts on them.

```
d["cluster"] = km.labels_
print(d.groupby("cluster")[["withdrew"]].mean().round(3))
print(pd.crosstab(d.cluster, d.study_mode, normalize="index").round(3))
```

Running this on our data returns:

```
withdrew by cluster:        study_mode by cluster:
0   0.161                   0   92.8% FT,   7.2% PT
1   0.200                   1    0.0% FT, 100.0% PT
2   0.029                   2   96.2% FT,   3.8% PT
```

Cluster 1 is 100% part-time. "Cluster 1" is not a discovered student type — it is the variable study_mode, rediscovered through age. Any intervention targeted at cluster 1 is an intervention targeted at part-time students, whether or not anyone intended that, and it must be evaluated and justified on those terms. An analyst who reported "the algorithm identified a distinct third segment" without running this cross-tabulation would be presenting a known administrative category as a data-driven finding.

Step 5 — Validate against something external. The clustering never saw withdrew, yet withdrawal rates across the three clusters are 16.1%, 20.0%, and 2.9% — a 7-fold spread. That is mild external validation: the grouping carries some information about an outcome it was not shown. If they do not, the clusters may still be operationally useful but carry no information about the outcome anyone cares about.

Step 6 — State the instability in the deliverable. Cluster membership depends on k, on scaling, and on the seed. It must never be written back into a student record as an attribute. Write it to an analysis table with the date, the parameters, and the code version — and treat it as a snapshot of a method, not a fact about a person.

> **Faded variant.** Refit with k = 4 and produce the new profiles. Identify which original cluster splits. Then write the neutral names for all four, and state what changes about the recommended intervention — including whether any student has moved between groups in a way that would alter what they are offered.

Common mistakes

Mistake	Consequence	Prevention
Clustering unscaled features	Widest-range feature decides everything	StandardScaler in a pipeline
Treating k as discoverable	Overstates structure that may not exist	Justify k by use; run a null test
Interpretive cluster names	Smuggles unsupported claims about people	Name from features only
Not checking seed	Names artefacts of initialisation	Refit across seeds, compare

Mistake	Consequence	Prevention
stability		with ARI
Interpreting PCs as causes	Components are variance directions	Report loadings; hold names loosely

Chapter summary

Unsupervised methods always return an answer, and the absence of a scorecard is the central difficulty. k-means is a distance model, so scaling determines the result and is a modelling assumption rather than a formality. Neither elbow nor silhouette identifies a true k, because the data need not contain one — a permutation null test tells you whether the structure is real, and takes four lines. PCA finds directions of variance, not causes. Cluster names applied to people should describe observed features and nothing more, and cluster membership should never become a stored attribute.

Check your understanding

1. Why must features be scaled before k-means?

2. The elbow plot shows no clear bend. What do you do?

3. What does the first principal component represent?

4. Your silhouette score is 0.32. Is that good?

> **Model answers are in the Reference volume, Part 1.** Write your own answer first — comparing a finished attempt against a model answer teaches far more than reading the model answer cold.

Lab work — Lab 11: Segmentation with honest uncertainty

> The full two-hour version of this lab, with timings, is **Lab 11** in the Lab Manual.

Deliverable: notebook plus a one-page segment brief.

1. Build a feature matrix from outcomes.csv and students.csv. Scale it in a pipeline.
2. Cluster with k from 2 to 8. Plot inertia and silhouette.
3. Run a permutation null test. Report whether the observed silhouette exceeds the null distribution.
4. Check stability: refit k = 3 across ten seeds and report pairwise ARI.
5. Repeat the whole clustering unscaled and compare with ARI. Quantify how much scaling changed the answer.
6. Fit PCA. Report explained variance and the loadings of the first two components. Plot the data in PC space, coloured by withdrew.
7. Profile your chosen clusters and give each a neutral name. Cross-tabulate against study_mode and first_generation.

8. Write the brief: what the segments are, how confident you are they exist, and one intervention per segment with a stated risk of getting it wrong.

Chapter 14 — Time Series and Text

> *Time-ordered data break the assumption every method so far has relied on: that the rows are exchangeable.*

After this chapter you can: identify trend, seasonality, and autocorrelation; build lag and rolling features without leakage; use chronological validation; represent text numerically; and fit a simple text classifier.

Time series and text shared a chapter with four other topics in the first edition. Each now has room.

14.1 Why time changes the rules

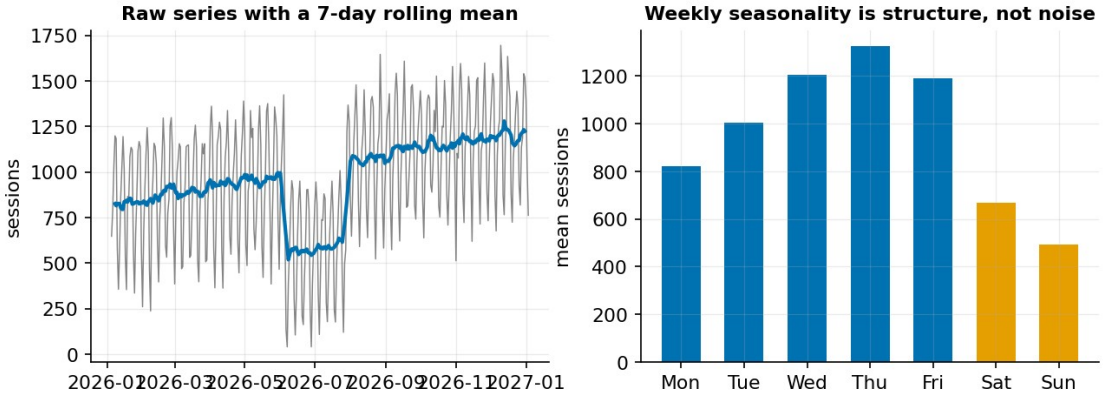
Every method so far assumed rows are independent and exchangeable — you could shuffle them without loss. Time-ordered data violate this. Today's value is correlated with yesterday's, the order carries information, and the future must never inform the past.

Consequences: random train/test splits are invalid, k-fold cross-validation is invalid, and standard errors assuming independence are too small.

14.2 Structure in a series

Four components:

- **Trend** — long-run direction
- **Seasonality** — repeating cycles of fixed period
- **Cyclic** — repeating without fixed period
- **Noise** — the remainder



A forecast that ignores the weekend effect will be wrong in a completely predictable way twice a week.

Weekly seasonality is structure, not noise.

Our platform_daily.csv has all four: an upward trend, strong weekly seasonality, a term-break level shift, and daily noise. Weekday means run from 491 sessions on Sunday to 1,325 on Thursday — a 2.7-fold swing that a model ignoring day-of-week would treat as unexplained error.

14.3 Lag and rolling features

```
daily = pd.read_csv("data/platform_daily.csv", parse_dates=["date"]).sort_values("date")

daily["lag_1"]  = daily.sessions.shift(1)
daily["lag_7"]  = daily.sessions.shift(7)
daily["roll_7"] = daily.sessions.shift(1).rolling(7).mean()
daily["dow"]    = daily.date.dt.dayofweek
```

Note .shift(1) **before** .rolling(7). Without it, the window includes the current day — the value you are predicting. This is temporal leakage in one line, and it is easy to write by accident.

14.4 Baselines for forecasting

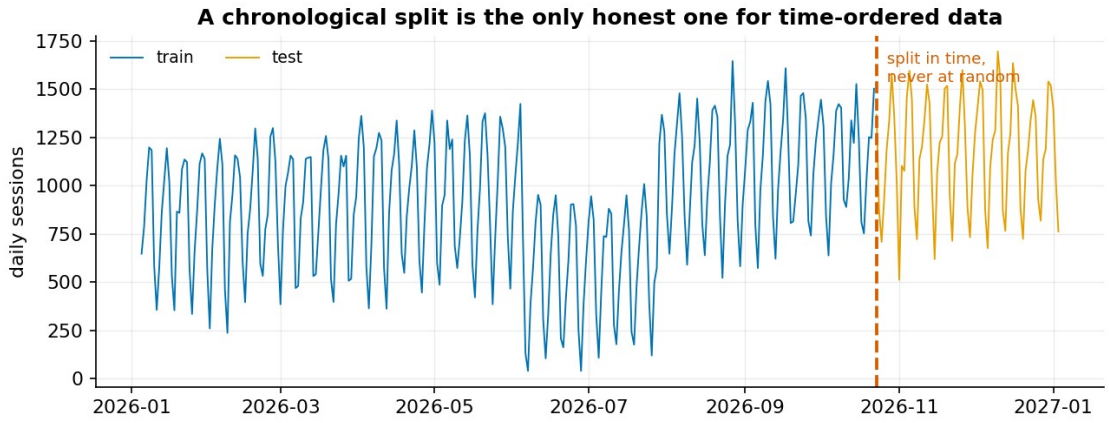
Forecasting baselines are unusually strong, and skipping them is how people convince themselves a model works.

- **Naive:** tomorrow equals today.
- **Seasonal naive:** tomorrow equals the same weekday last week.
- **Mean:** tomorrow equals the training mean.

```
cut = int(len(daily) * 0.8)
train, test = daily.sessions[:cut], daily.sessions[cut:]

naive  = np.full(len(test), train.iloc[-1])
snaive = daily.sessions.shift(7)[cut:]
mean_f = np.full(len(test), train.mean())

for name, f in [("naive", naive), ("seasonal naive", snaive), ("mean", mean_f)]:
    print(f"{name:16s} MAE {np.abs(test.values - np.asarray(f)).mean():.1f}")
# naive           MAE 345.8
# seasonal naive   MAE  88.9
# mean            MAE 338.0
```

Seasonal naive is nearly four times better than either alternative. Any model that does not beat 88.9 has added nothing over “look at last Tuesday.”

14.5 Chronological validation



Random k-fold here trains on Thursday to predict the preceding Tuesday. The score will look excellent and mean nothing.

Random k-fold here trains on Thursday to predict the preceding Tuesday.

```
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5)
```

Each fold trains on a prefix and tests on the block immediately after, so the model never sees the future.

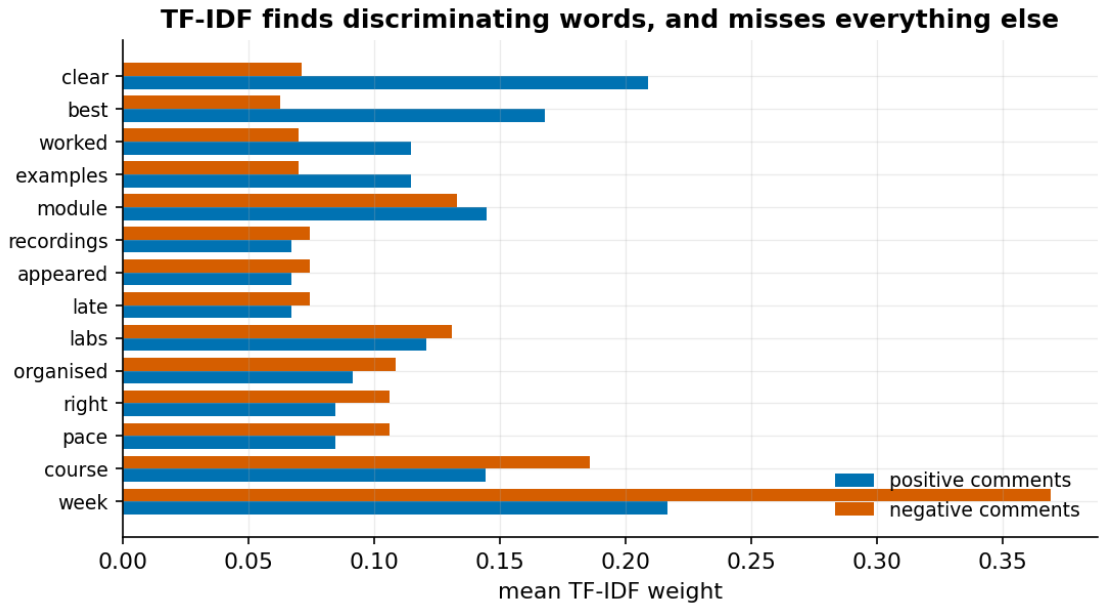
14.6 Text as data

Text must become numbers. The simplest useful representation:

Bag of words counts token occurrences, discarding order. **TF-IDF** weights each token by its frequency in the document and its rarity across the corpus:

$$\text{tfidf}(t, d) = \text{tf}(t, d) \times \log \frac{N}{\text{df}(t)}$$

A word appearing in every document gets a weight near zero. A word appearing in a few gets a high one.



Bag-of-words has no notion of order or negation: "not clear at all" and "clear" share a token.

TF-IDF finds discriminating words and misses everything else.

What bag-of-words cannot represent: word order, negation, sarcasm, or context. “Not clear at all” and “clear” share the token *clear*. This is not a tuning problem; it is the representation.

14.7 A text classifier

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

fb = pd.read_csv("data/feedback.csv")
pipe = make_pipeline(TfidfVectorizer(stop_words="english"),
                     LogisticRegression(max_iter=1000))
cv = cross_val_score(pipe, fb.comment, fb.sentiment, cv=5)
print(f"CV accuracy {cv.mean():.3f} (±{cv.std():.3f})")   # 0.895 (±0.019)
print("majority baseline:",
    fb.sentiment.value_counts(normalize=True).max().round(3))   # 0.519
```

89.5% against a 51.9% baseline. Substantial — and the remaining 10.5% is concentrated in exactly the comments containing negation and mixed sentiment, which the representation cannot express.

Worked example 14.1 — Why a random split lies

Forecast daily sessions. Compare a random split with a chronological one.

Step 1 — Build features.

```
daily["lag_1"] = daily.sessions.shift(1)
daily["lag_7"] = daily.sessions.shift(7)
daily["dow"]   = daily.date.dt.dayofweek
model_df = daily.dropna()
X = pd.get_dummies(model_df[["lag_1", "lag_7", "dow"]], columns=["dow"])
y = model_df.sessions
```

Step 2 — Random split (wrong).

```
from sklearn.model_selection import train_test_split
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.2, random_state=0)
lm = LinearRegression().fit(X_tr, y_tr)
print(f"random-split MAE: {mean_absolute_error(y_te, lm.predict(X_te)):.1f}")
```

Step 3 — Chronological split (right).

```
cut = int(len(X) * 0.8)
lm2 = LinearRegression().fit(X[:cut], y[:cut])
print(f"chronological MAE: {mean_absolute_error(y[cut:], lm2.predict(X[cut:])):.1f}")
```

Step 4 — Explain the gap. The random split places 20 March in training and 19 March in test. The model has seen the day *after* the one it is predicting, and since consecutive days are highly correlated, that is close to observing the answer. The chronological split forbids this.

Step 5 — Establish what "good" means. Compare against seasonal naive, MAE 88.9. A model with chronological MAE of 80 is a genuine but modest improvement over copying last Tuesday. A model with random-split MAE of 60 has beaten nothing, because its evaluation was invalid.

Step 6 — Note the subtler version. Even chronological splitting is optimistic if features were engineered on the full series — a rolling mean computed before splitting has already smoothed test values into training features. The shift(1) before rolling(7) is what prevents it, and its absence produces no error message.

> **Faded variant.** Fit the same model with TimeSeriesSplit(n_splits=5) and report MAE per fold. Are later folds better or worse? Explain what an increasing trend in fold error would tell you about model drift.

Worked example 14.2 — Beating the seasonal baseline

Can a model improve on "same weekday last week"?

Step 1 — Establish the target to beat. Seasonal naive MAE = 88.9.

Step 2 — Fit a model with the same information plus more.

```
feats = ["lag_1", "lag_7", "roll_7"]
Xf = pd.get_dummies(model_df[feats + ["dow"]], columns=["dow"])
cut = int(len(Xf) * 0.8)
lm = LinearRegression().fit(Xf[:cut], y[:cut])
pred = lm.predict(Xf[cut:])
print(f"model MAE: {mean_absolute_error(y[cut:], pred):.1f}")
```

Step 3 — Decompose the improvement. Fit progressively and see which feature earns its place:

Features	Expected behaviour
lag_1 only	Poor — yesterday is a bad guide across weekends
dow only	Captures most of the seasonality
lag_7 only	Approximately the seasonal naive baseline
lag_7 + dow	Should beat the baseline
all	Marginal further gain

This decomposition matters more than the final number. It tells you *what* the model learned, and it will usually reveal that one or two features carry nearly everything.

Step 4 — Check residuals over time.

```
resid = y[cut:].values - pred
plt.plot(model_df.date[cut:], resid); plt.axhline(0)
```

Residuals should look like noise. Structure means a component is unmodelled: a run of same-sign residuals indicates missed trend or a level shift, and a repeating pattern indicates missed seasonality.

Step 5 — Confront the term break. Our series has a level shift around days 150 to 205. If the test period contains one and the training period does not, MAE will be poor for reasons no feature can fix. Forecasting cannot extrapolate an event type it has never seen — the honest response is a calendar feature marking term breaks, not a more flexible model.

Step 6 — Report against the right comparison.

> *A linear model with day-of-week and 7-day lag features achieves a chronological test MAE of X sessions, against a seasonal-naive baseline of 88.9. The improvement is Y%. Residuals show no remaining weekly pattern but do show elevated error across the term break, which the model has no feature for.*
>
> **Faded variant.** Add a binary is_term_break feature covering days 150–205 and refit. Report the MAE change. Then state the problem with this feature for forecasting *next* year, and what you would need instead.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

Worked example 14.3 — What TF-IDF cannot see

Classify course feedback. Then find where it fails.

Step 1 — Fit and score.

```
pipe = make_pipeline(TfidfVectorizer(stop_words="english"),
                     LogisticRegression(max_iter=1000))
cv = cross_val_score(pipe, fb.comment, fb.sentiment, cv=5)
print(f"CV accuracy {cv.mean():.3f}")     # 0.895
```

89.5% against a 51.9% baseline — roughly a 78% reduction in error rate.

Step 2 — Inspect what it learned.

```
pipe.fit(fb.comment, fb.sentiment)
vec = pipe.named_steps["tfidfvectorizer"]
lr  = pipe.named_steps["logisticregression"]
coefs = pd.Series(lr.coef_[0], index=vec.get_feature_names_out()).sort_values()
print("most negative:"); print(coefs.head(6).round(2).to_string())
print("most positive:"); print(coefs.tail(6).round(2).to_string())
```

Sensible: words like *rushed*, *struggled*, *poorly* pull negative; *excellent*, *recommend*, *helped* pull positive.

Step 3 — Find the errors, and look at them.

```
from sklearn.model_selection import cross_val_predict
pred = cross_val_predict(pipe, fb.comment, fb.sentiment, cv=5)
wrong = fb[pred != fb.sentiment]
print(f"{len(wrong)} misclassified of {len(fb)}")
for c in wrong.comment.head(5):
    print("-", c)
```

Step 4 — Diagnose the pattern. The errors concentrate in two kinds of comment:

"Not clear at all in week 1, but it became well structured later on." — labelled positive. The model sees *clear*, *structured*, *week* and also *not*. But not is a stop word and was removed, and even if kept, bag-of-words has no mechanism to attach it to clear. **Negation is invisible to this representation.**

"Really engaging lecturer. The course itself was poorly organised." — labelled negative. The model sees strong positive and strong negative tokens and averages them. It has no way to know which clause carries the overall verdict.

Step 5 — Establish the ceiling. These failures are not fixable by tuning. Options that would actually help:

- **n-grams:** ngram_range=(1,2) captures “not clear” as one token. Try it and measure.
- **Keep negation words:** removing not from the stop list.
- **A model with word order:** an RNN or transformer, at much greater cost and with far less interpretability.

```
pipe2 = make_pipeline(TfidfVectorizer(stop_words="english", ngram_range=(1, 2)),
                      LogisticRegression(max_iter=1000))
print(cross_val_score(pipe2, fb.comment, fb.sentiment, cv=5).mean().round(3))
# 0.923
```

Bigrams recover about a quarter of the remaining error — from 10.5% to 7.7% — by capturing “not clear” as a single token. They do nothing for the mixed-verdict comments, because those need clause structure rather than adjacency.

Step 6 — Decide whether the ceiling matters. For counting the rough balance of positive and negative feedback across 732 comments, 89.5% is ample. For flagging individual comments for a course leader to read, the 10.5% error rate concentrated in mixed and negated comments is a serious problem — those are precisely the most informative comments.

Responsible-use note. These are student comments about named staff. Sentiment classification of them should never feed a performance evaluation. The model cannot distinguish a criticism of the room timetable from a criticism of the teaching, and the aggregate score would carry an authority the method does not support.

> **Faded variant.** Compare unigram and bigram models on accuracy. Then hand-construct five comments designed to break the classifier — using negation, sarcasm, and mixed clauses — and report how many it gets wrong. State which of your five a bigram model could plausibly fix.

Common mistakes

Mistake	Consequence	Prevention
Random split on time data	Trains on the future	Chronological split or TimeSeriesSplit
Rolling window without shift(1)	Includes the current value	.shift(1).rolling(n)
No seasonal-naive baseline	Model looks good against nothing	Always compute it first
Fitting the vectorizer before splitting	Vocabulary leakage	Vectorizer inside the pipeline
Treating text accuracy as sentiment truth	Negation and sarcasm invisible	Read the misclassified examples

Chapter summary

Time-ordered data are not exchangeable, so random splits and k-fold cross-validation are invalid and standard errors assuming independence are too small. Seasonal naive is a strong baseline — on our data four to five times better than the alternatives — and any model must beat it to have earned anything. Lag and rolling features must trail, never centre. Text becomes numbers through bag-of-words or TF-IDF, which discard order and therefore cannot represent negation, sarcasm, or clause structure; the resulting errors concentrate in the most informative comments, and no amount of tuning removes them.

Check your understanding

1. Why is k-fold cross-validation invalid for time series?

2. What does the seasonal naive baseline predict, and why is it strong here?

3. Why does rolling(7) without shift(1) leak?

4. Your sentiment classifier is 89.5% accurate. What can you conclude about the 10.5%?

> **Model answers are in the Reference volume, Part 1.** Write your own answer first — comparing a finished attempt against a model answer teaches far more than reading the model answer cold.

Lab work — Lab 12: Forecasting and text

> The full two-hour version of this lab, with timings, is **Lab 12** in the Lab Manual (shared with Chapter 15).

Deliverable: notebook covering both halves.

Time series

1. Load platform_daily.csv. Plot the series, a 7-day rolling mean, and the weekday profile.
2. Compute naive, seasonal-naive, and mean baselines on a chronological 80/20 split.
3. Build lag and rolling features with correct shifting. Fit a model and beat 88.9 MAE.
4. Repeat with a random split. Report the difference and explain it.
5. Plot residuals over time. Identify any period the model handles badly.

Text

6. Load feedback.csv. Fit a TF-IDF and logistic regression pipeline. Report CV accuracy against the 51.9% majority baseline.
7. Extract the ten most positive and ten most negative coefficients.
8. Use cross_val_predict to find every misclassified comment. Read them and classify the failures into types.
9. Compare unigram and bigram models. Report which failure type improves.

10. Write three sentences on whether this model should inform any decision about a member of staff.

Chapter 15 — Dashboards, Delivery, and Responsible Practice

> *The analysis is not finished when the model fits. It is finished when someone can act on it and challenge it.*

After this chapter you can: build a dashboard as a decision product; apply privacy protections including cell suppression; structure a report and a reproducible submission; and apply a responsible-practice checklist end to end.

15.1 Dashboards are decision products

A dashboard is not a collection of charts. It is a tool someone uses to make a recurring decision. Design questions come before any code:

- Who uses it, how often, and to decide what?
- What action follows from each state it can display?
- What must it *not* show?
- How will a user know the data are stale or broken?

A dashboard with no action attached to any state is a report with extra maintenance cost.

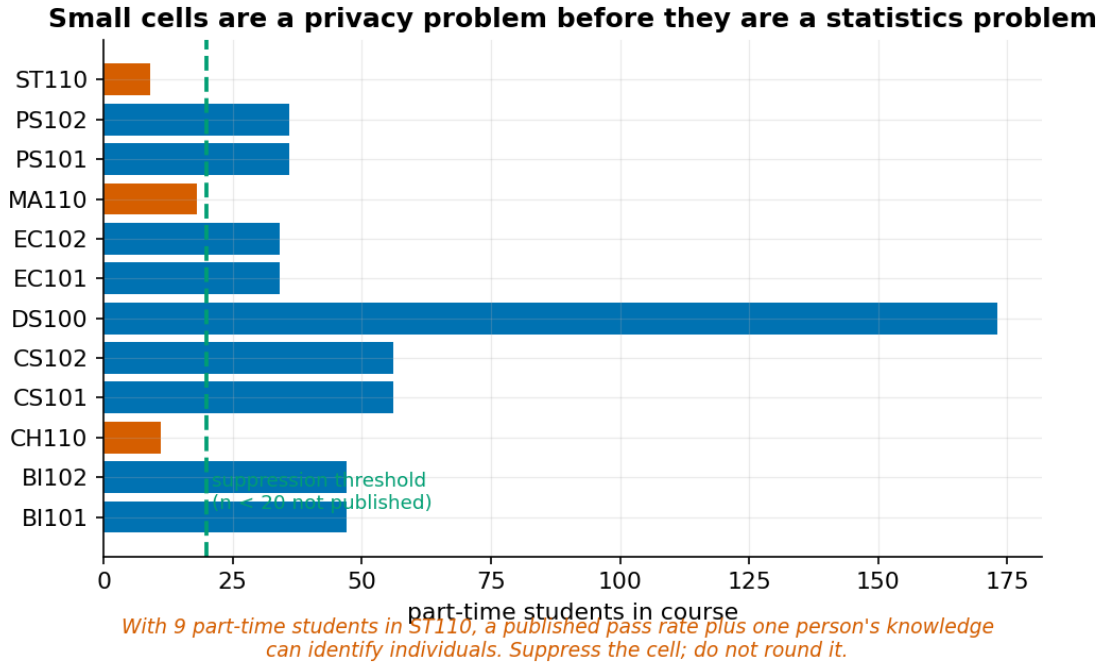
15.2 A minimal Streamlit dashboard

```
import streamlit as st
import pandas as pd

MIN_CELL = 20      # suppression threshold — see 15.3

@st.cache_data
def load():
    students = pd.read_csv("data/students.csv")
    enrol    = pd.read_csv("data/enrolments.csv")
    return enrol.merge(students, on="student_id", validate="many_to_one")

df = load()

st.title("Course outcomes")
st.caption(f"Synthetic data. Cells with fewer than {MIN_CELL} students are suppressed.")

prog = st.selectbox("Programme", ["All"] + sorted(df.programme.unique()))
view = df if prog == "All" else df[df.programme == prog]

summary = (view.groupby("course_code")
```

```
        .agg(students=("student_id", "nunique"),
             mean_grade=("final_grade", "mean"),
             pass_rate=("passed", "mean"))
        .round(3))

suppressed = summary.students < MIN_CELL
summary.loc[suppressed, ["mean_grade", "pass_rate"]] = None

st.dataframe(summary)
if suppressed.any():
    st.warning(f"{suppressed.sum()} course(s) suppressed: fewer than {MIN_CELL}
students.")
```

The suppression happens **before** the data reach the display, not as a formatting choice. A protection applied at render time is one refactor away from disappearing.

15.3 Privacy by construction



Small cells are a privacy problem before they are a statistics problem.

Nine part-time students take ST110. Publishing a pass rate for that cell, combined with one person's knowledge of one student, can reveal individuals. This is **differencing**: two published aggregates that each look safe can be subtracted to expose a single record.

Practical protections:

- **Cell suppression** below a minimum count. Twenty is a common floor.

- **Complementary suppression.** Suppressing one cell in a row whose total is published lets the reader recover it by subtraction. You must suppress a second.
- **Aggregation** to coarser categories.
- **Access control.** Not everything belongs on a public dashboard.

Rounding is not a protection. A rounded value plus a published total still permits differencing.

Data minimisation: a dashboard should load only the columns it displays. Loading first_generation into a dashboard that never shows it creates risk with no benefit.

15.4 The report

A structure that works for most analyses:

1. **Executive summary** — the finding and the recommendation, in under 200 words.
2. **Question and decision** — what was asked, what will be decided.
3. **Data and provenance** — source, licence, unit of observation, coverage, quality.
4. **Method** — in plain language first, technical detail second.
5. **Results** — with uncertainty.
6. **Limitations** — what this cannot establish.
7. **Recommendation** — proportionate to the evidence.
8. **Reproduction** — the notebook, the data, and how to run it.

The limitations section is the one that establishes credibility. A report without one invites the reader to supply their own, and they will be less charitable than you.

15.5 Making your work reproducible

A reproducible submission is one that someone else can run and get your numbers from. In Colab, that means a folder in Drive containing:

```
my_project/
├── analysis.ipynb      the notebook, running top to bottom
├── data/               the original files, never edited
└── figures/            anything the notebook saves out
```

Four rules make it work:

1. **Never edit the raw data files.** Load them, transform them in the notebook, and write any cleaned version to a new file. If you overwrite the original, nobody can check your cleaning — including you.
2. **Every output is produced by the notebook.** No numbers typed in by hand, no figures edited in another program.
3. **Every random operation has a seed.** np.random.default_rng(42), random_state=0. Without these, your results change every run and cannot be checked.

4. **The notebook runs top to bottom from clean.** Use **Runtime → Restart and run all** before you submit, every time. This is the test that catches most problems.

That last rule sounds trivial and is not. While exploring you will run cells out of order, define a variable in a cell you later delete, and build results that depend on a state no fresh run reproduces. The restart test finds this in thirty seconds; a marker finds it too, and later.

15.6 The responsible-practice checklist

Before analysis

- ☐ Who benefits, who bears risk?
- ☐ Is a less intrusive analysis sufficient?
- ☐ What is the intended action, and what oversight applies?

During analysis

- ☐ Is the sample representative of the population in the question?
- ☐ Are sensitive attributes used for auditing rather than as decision features?
- ☐ Has performance been disaggregated by group?
- ☐ Has leakage been checked feature by feature?

Before delivery

- ☐ Are limitations stated as prominently as findings?
- ☐ Are small cells suppressed, with complementary suppression?
- ☐ Can an affected person learn they were flagged and contest it?
- ☐ Is the claim on the rung the evidence supports?
- ☐ Does the recommendation match the strength of the evidence?

Worked example 15.1 — Suppression that does not work

A dashboard shows pass rates by course and study mode.

Step 1 — Find the exposed cells.

```
counts = (df.groupby(["course_code", "study_mode"]).student_id.nunique()
          .unstack(fill_value=0))
print(counts[counts.min(axis=1) < 20].to_string())
#             Full-time  Part-time
# CH110           249        11
# MA110           558        18
# ST110           220        9
```

Three courses have part-time cells below 20.

Step 2 — Apply naive suppression. Hide those three cells.

Step 3 — Break it. The dashboard also displays a course total. For ST110:

- Published: overall pass rate 0.214, total students 229
- Published: full-time pass rate 0.218, n = 220
- Suppressed: part-time cell (n = 9)

The reader computes:

$$\text{PT passes} = 0.214 \times 229 - 0.218 \times 220 = 49.0 - 48.0 = 1.0$$

With $n_{PT} = 9$ — itself published as part of the total — the part-time pass rate is $1/9 = 0.111$. **The suppressed cell has been recovered exactly by subtraction.**

Step 4 — Apply complementary suppression. Suppressing one cell in a row is insufficient whenever the margin is published. You must suppress a second cell, or the margin:

```
def suppress(table, counts, min_cell=20):
    out = table.copy()
    for course in out.index:
        row = counts.loc[course]
        small = row[row < min_cell].index.tolist()
        if not small:
            continue
        if len(small) == 1:                # complementary suppression
            second = row.drop(small).idxmin()   # suppress the next smallest too
            small.append(second)
        out.loc[course, small] = None
    return out
```

For ST110 this suppresses both the part-time *and* the full-time cell. The course total may still be published, because it cannot be decomposed.

Step 5 — Check the whole publication, not each table. Differencing works across releases too. If last term's dashboard published a cell and this term's suppresses it, and the cohort changed by two students, the difference exposes those two. Suppression rules must be consistent over time.

Step 6 — Record the rule where users see it. "Cells with fewer than 20 students are suppressed, along with a second cell where necessary to prevent recovery by subtraction." A user who does not know why a number is missing will ask for it, and someone will eventually provide it.

> **Faded variant.** The dashboard also offers a first_generation filter. With both filters active, compute how many cells fall below 20. Then determine whether allowing both filters simultaneously is defensible at all, and what you would change.

NUU • FUNDAMENTALS OF DATA SCIENCE • STUDENT COPY

Worked example 15.2 — Writing the recommendation

Your withdrawal model achieves ROC-AUC 0.708 on held-out data, and at threshold 0.15 gives precision 0.179 and recall 0.512. Write the recommendation.

Step 1 — State what the model does, in the units of the decision.

> At the proposed threshold, the model flags 117 of 360 students. Of those flagged, 21 withdraw and 96 do not. Of the 41 students who withdraw, the model finds 21 and misses 20.

No metrics yet — just counts. A committee can reason about counts.

Step 2 — State the error costs explicitly, and who bears them.

> A false positive means a student receives an offer of tutoring they did not need: a short email and about ten minutes of adviser time. A false negative means a student who withdraws was never offered support. The costs are asymmetric and fall on different parties — the false-positive cost falls on the institution, the false-negative cost on the student.

Step 3 — Derive the threshold from capacity, not from a curve.

> Three advisers can each contact approximately 40 students, giving a capacity of 120. The threshold of 0.15 flags 117, fitting that capacity. Lowering it to 0.11 would flag 158 and raise recall to 68%, which would require a fourth adviser.

Step 4 — Say what the model must not be used for.

> This model must not be used to allocate anything scarce, to inform academic progression decisions, or to populate a student record. At 18% precision, more than four in five flagged students would be wrongly affected by any consequential decision.

Step 5 — Specify oversight and contestability.

> Every flag is reviewed by an adviser before contact. The email offers support and does not mention risk scoring. Any student may ask whether they were flagged, see which factors contributed, and request review. Model performance will be audited by study mode and first-generation status each term, with results reported to the committee.

Step 6 — State the strength of the evidence.

> The model's discrimination is modest — ROC-AUC 0.708, where 0.5 is chance. It raises the withdrawal rate among flagged students from a base of 11.5% to 17.9%, a 1.6-fold concentration. It is a triage aid for allocating limited adviser time, not a predictor of individual outcomes.

Step 7 — Recommend proportionately.

> **Recommendation:** deploy for one term as a triage aid at threshold 0.15, with adviser review of every flag, and evaluate against a randomly selected control group of students who are contacted regardless of score. Do not extend to any other use without a fresh evaluation.

Note the control group. Without one, next term the team will observe that flagged-and-contacted students withdrew less and conclude the programme worked — when the students most likely to be flagged are also those whose behaviour was going to change anyway.

> **Faded variant.** Rewrite this recommendation for a threshold of 0.30 (30 flagged, precision 0.30, recall 0.22). Identify which sentences change and which do not, and state which threshold you would defend to a student who was missed.

Worked example 15.3 — Auditing your own capstone

Before submitting, run the checklist against your own work. This example uses a plausible student project.

The project: predicting DS100 final grade from entry score and early engagement, with a dashboard showing predicted grades by programme.

Step 1 — Who bears risk? Students. A predicted grade attached to a student can affect how staff treat them, regardless of accuracy. **Action:** the dashboard shows programme-level aggregates only; individual predictions are never displayed or stored.

Step 2 — Is a less intrusive analysis sufficient? Probably. If the purpose is identifying courses needing support, course-level pass rates answer it without any individual modelling. **Action:** state in the report that the model was built to demonstrate the workflow, and that the operational question is answerable without it.

Step 3 — Leakage, feature by feature.

Feature	Known at prediction time?
entry_score	Yes — at registration
early_logins (weeks 1–4)	Yes — from week 5
midterm_satisfaction	**No** — collected week 7, after the prediction point
passed	**No** — it is the target

Two features must be dropped. Finding this in the checklist rather than in a viva is the point of the checklist.

Step 4 — Disaggregate performance.

```
for level, g in test_df.groupby("study_mode"):
    print(f"{level:12s} n={len(g):4d}  MAE={mean_absolute_error(g.y, g.pred):.2f}")
```

If MAE is 7.5 for full-time and 11.2 for part-time, the model is substantially worse for the group with fewer training examples. **Action:** report it. A model that fails unevenly and says so is more useful than one that reports a single average.

Step 5 — Check the claim rung. Draft sentence: *"Early engagement drives final grades."* That is causal, from observational data. **Corrected:** *"Early engagement is associated with final grades; among students with similar entry scores, those with higher week 1–4 submissions had higher mean final grades. This is observational and does not establish that increasing engagement would raise grades."*

Step 6 — Small cells in the dashboard. With a programme filter and a study-mode filter, several cells fall below 20. **Action:** implement suppression with a complementary rule, and display the rule.

Step 7 — Reproduction. Use **Runtime → Restart and run all**, and confirm every figure and number regenerates. Then send the notebook to a classmate and have them run it in their own Colab. Roughly half of first attempts fail — usually on a file path that only exists in the author's Drive.

Step 8 — Limitations, written before the conclusion. Writing limitations first prevents the common failure of drafting a strong conclusion and then softening it. If the limitations make the conclusion untenable, the conclusion was wrong.

> **Faded variant.** Run this checklist against your own Lab 12 submission. Identify at least two items you would now change, and state whether either changes a conclusion.

Common mistakes

Mistake	Consequence	Prevention
Suppressing one cell per row	Recoverable by subtraction	Complementary suppression
Rounding as privacy protection	Differencing still works	Suppress, do not round
Dashboard with no attached action	Maintenance cost, no value	State the decision per view
Limitations after conclusions	Conclusions get written too strong	Write limitations first
A file path only you have	Nobody else can run it	Define one DATA = … variable at the top and use it everywhere
Deploying without a control group	Cannot tell whether it worked	Randomise a holdout

Chapter summary

A dashboard is a decision product, and every view should have an action attached. Privacy protections belong in the data layer, not the display layer; cell suppression requires a complementary rule because a single suppressed cell beside a published margin is recoverable by subtraction, and rounding protects nothing. A report's limitations section is what makes the rest credible, and writing it first prevents overstated conclusions. A recommendation should be proportionate to the evidence, should state what the model must not be used for, and should specify how an affected person can contest a decision.

Check your understanding

1. Why is suppressing one small cell in a row insufficient?

2. What makes a dashboard a decision product rather than a report?

3. Why write limitations before conclusions?

4. Your model is deployed and withdrawals fall. Did it work?

> **Model answers are in the Reference volume, Part 1.** Write your own answer first — comparing a finished attempt against a model answer teaches far more than reading the model answer cold.

Lab work — Lab 12: Capstone delivery

> The full two-hour version of this lab, with timings, is **Lab 12** in the Lab Manual (shared with Chapter 14).

Deliverable: dashboard, report, a notebook that runs from clean, and an 8-minute presentation.

1. Build a Streamlit dashboard over the course dataset with at least two filters. Implement cell suppression with a complementary rule in the data layer.
2. State, for each view, the decision it supports and the action that follows.
3. Write the report using the structure in 15.4. Draft the limitations section before the conclusions.
4. Assemble your submission folder. Seed every random operation, and define your data path once at the top.
5. Exchange notebooks with a classmate. Attempt to reproduce their headline number in your own Colab. Document what broke and why.
6. Run the responsible-practice checklist against your own work and report at least two things you changed as a result.
7. Present for 8 minutes to a non-technical audience. Spend at least one minute on what your analysis cannot establish.