Lecture IX - Data Visualization

Programming with Python

Dr. Tobias Vlćek
Kühne Logistics University Hamburg - Fall 2026

Episode 9: The Pitch Deck

Charts are arguments

Friday, the boardroom. Picture it: the investor slides a pencil out of her jacket and **sharpens it** while you set up, the sound louder than it has any right to be.

. . .

Last week you handed her honest numbers, a table she could trust. She read exactly one row, looked up, and said the thing this whole session is about:

> **"Tables are homework. Charts are arguments."**

. . .

A number convinces the careful. A **picture** convinces the room. Today: draw the right chart for the question, and refuse to draw a dishonest one.

🔥 Warm-up

Three questions from Episode 8. Commit. Hands up **before** the reveal.

Question 1

```
orders[orders["zone"] == "Nord"]
```

returns…

a) a True/False column b) only the Nord rows c) an error from comparing text

Answer 1

b) only the Nord rows: the True/False column is what the **inner** expression `orders["zone"] == "Nord"` makes. Wrapped in `orders[...]`, that mask keeps the rows where it's `True` and drops the rest.

Question 2

Tobi's AI wrote this. `orders` is a normal DataFrame. What happens?

```
orders.summarize()
```

a) a summary table b) an empty DataFrame c) an `AttributeError`

Answer 2

c) `AttributeError`: pandas has `.describe()`, not `.summarize()`. The AI invented a plausible name. An AI that sounds sure is not the same as an API that exists. You verify, every time.

Question 3

```
orders["total_eur"].describe()
```

shows…

a) the first five rows of the column, like `.head()` b) count / mean / std / min / quartiles / max c) the column's dtype, like `.dtypes`

Answer 3

b) count / mean / std / min / quartiles / max. One line, the column's whole statistical fingerprint. That's the real method Tobi's AI was reaching for.

The first chart

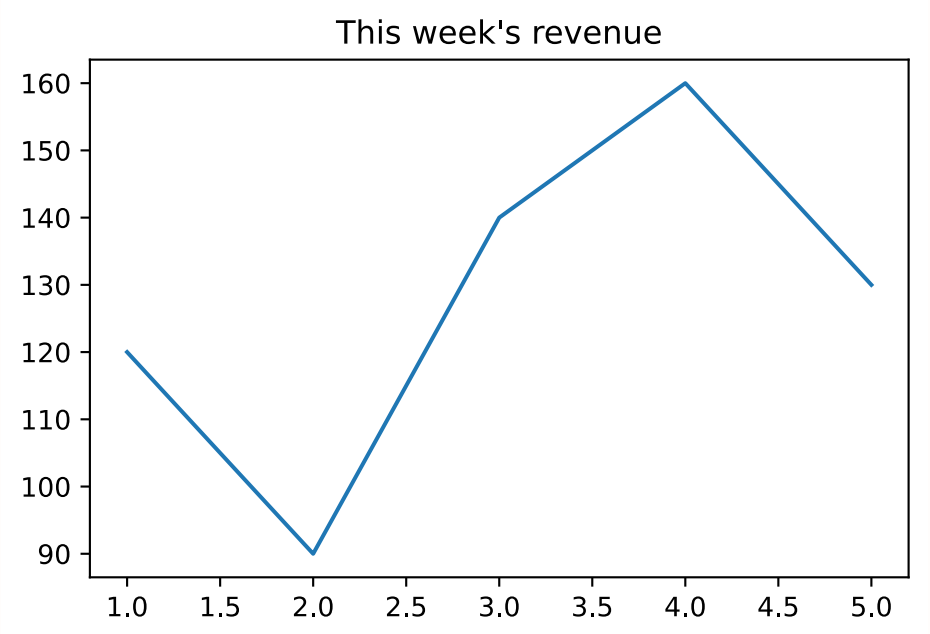
Matplotlib in four lines

Everyone plots with **matplotlib**, imported as `plt`. Four lines turn a list of numbers into a picture:

```
import matplotlib.pyplot as plt

days = [1, 2, 3, 4, 5]
revenue = [120, 90, 140, 160, 130]

plt.figure()          # OPEN: a fresh canvas, so this chart
                      #       doesn't draw on the last one
plt.plot(days, revenue)   # x, then y
plt.title("This week's revenue")
plt.gca()             # SHOW: "get current axes" (the chart)
```



...

> **Note**
>
> Two habits for every chart cell: open with `plt.figure()` (a clean canvas) and end with `plt.gca()`. In marimo there's no `plt.show()`; the figure appears as the cell's **last expression**, and `plt.plot(...)` alone returns line objects, not a picture. (A plain script run from a terminal does need `plt.show()` at the end; that's Session X.) Same frame around every chart in today's lab.

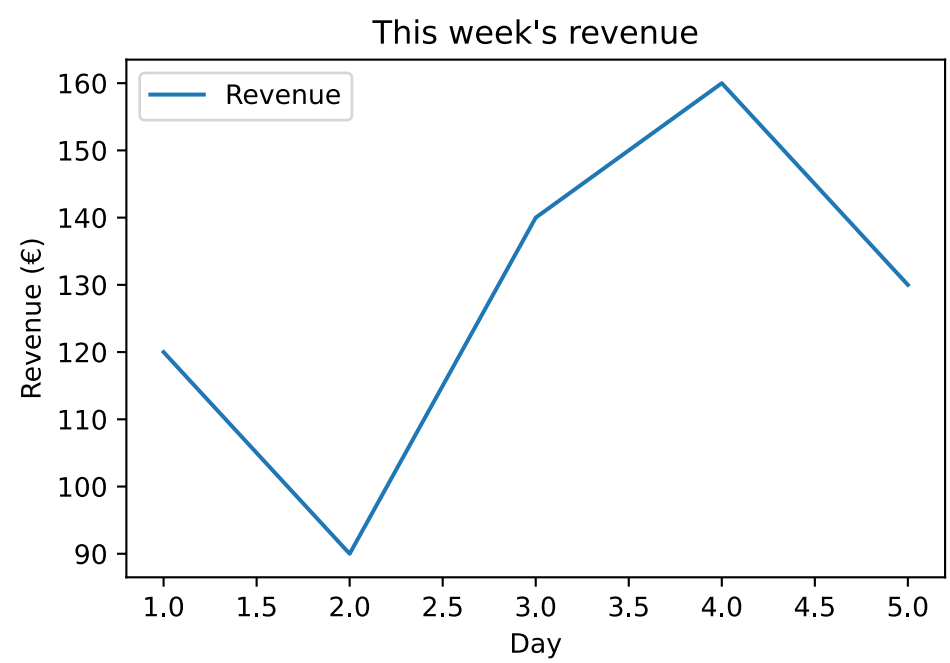
The parts of a chart

Every honest chart names its axes. `xlabel`, `ylabel`, `title`, and (when there's more than one line) a `legend`:

```
import matplotlib.pyplot as plt

days = [1, 2, 3, 4, 5]
revenue = [120, 90, 140, 160, 130]

plt.figure()
plt.plot(days, revenue, label="Revenue")   # label feeds the legend
plt.xlabel("Day")        # what the x-axis counts
plt.ylabel("Revenue (€)")  # what the y-axis measures
plt.title("This week's revenue")  # what the picture is about
plt.legend()             # the little key, one entry per label
plt.gca()
```



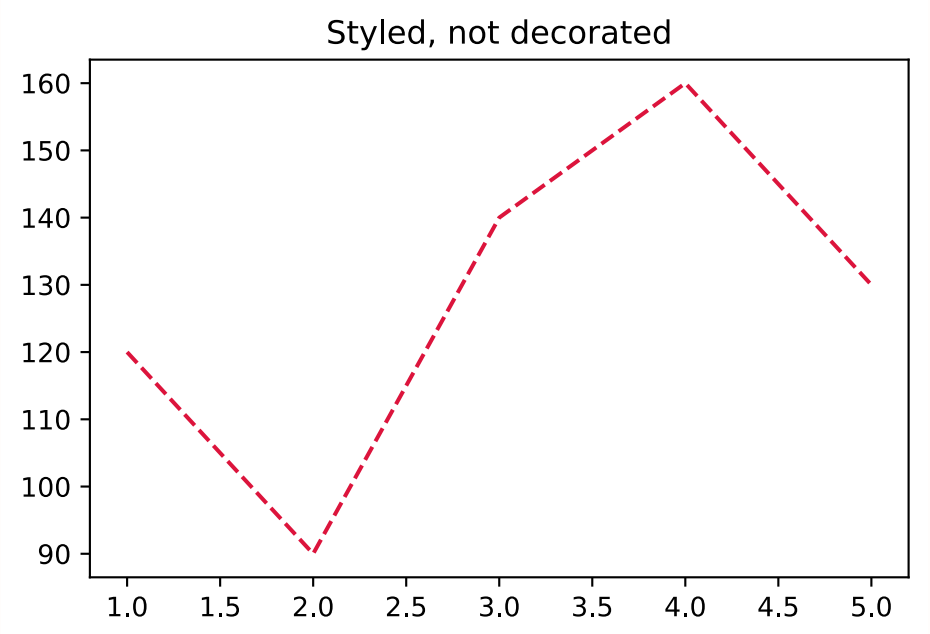
A little styling — and only a little

Two keywords cover almost everything: `color` and `linestyle`. Restraint is the professional move. One clear line beats a rainbow:

```
import matplotlib.pyplot as plt

days = [1, 2, 3, 4, 5]
revenue = [120, 90, 140, 160, 130]

plt.figure()
plt.plot(days, revenue, color="crimson", linestyle="--")
plt.title("Styled, not decorated")
plt.gca()
```



Predict: one chart or two?

Tobi runs **two** `plot` calls, back to back, with no `plt.figure()` between them:

```
plt.plot([3, 1, 2])
plt.plot([1, 2, 3])
```

a) one chart with two lines b) two separate charts, one per call c) an error: the figure is already in use

. . .

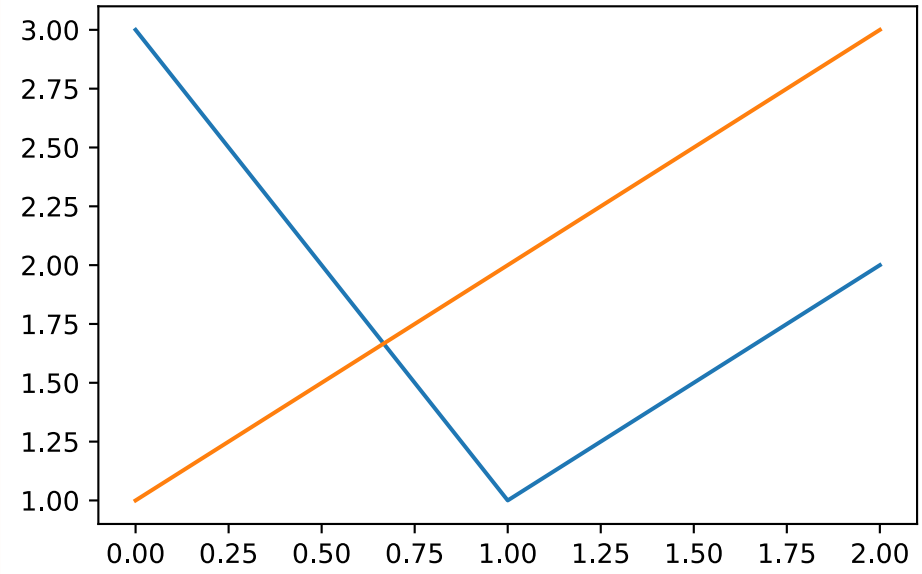
Predict first. Pick a letter, then I reveal the answer.

Answer: one chart, two lines

a) one chart with two lines, because matplotlib keeps drawing on the **current** figure until you start a new one with `plt.figure()`:

```
import matplotlib.pyplot as plt

plt.figure()
plt.plot([3, 1, 2])
plt.plot([1, 2, 3])   # same canvas: a second line, not a second chart
plt.gca()
```



...

That's exactly why every chart cell **opens** with `plt.figure()`: it's how you say "new picture, start clean."

Your turn — 10 minutes

Open the exercise (scan the QR or type the link):

python.tobiasvlcek.com/notebooks/ex_09_a/

First **predict** what happens, then run it.

The right chart for the question

Match the chart to the question

The wrong chart is its own kind of lie. Start from the **question**, not the chart:

The question	The chart	The call
Compare **categories**?	bar	`plt.bar(labels, heights)`
See a **distribution**?	histogram	`plt.hist(values)`
Two numbers per order?	scatter	`plt.scatter(x, y)`
Change over **time**?	line	`plt.plot(x, y)`

. . .

> **i** Note
>
> **Line** you already own from block one. Missing on purpose: the **pie chart**. The eye can't compare slice sizes; a bar is clearer.

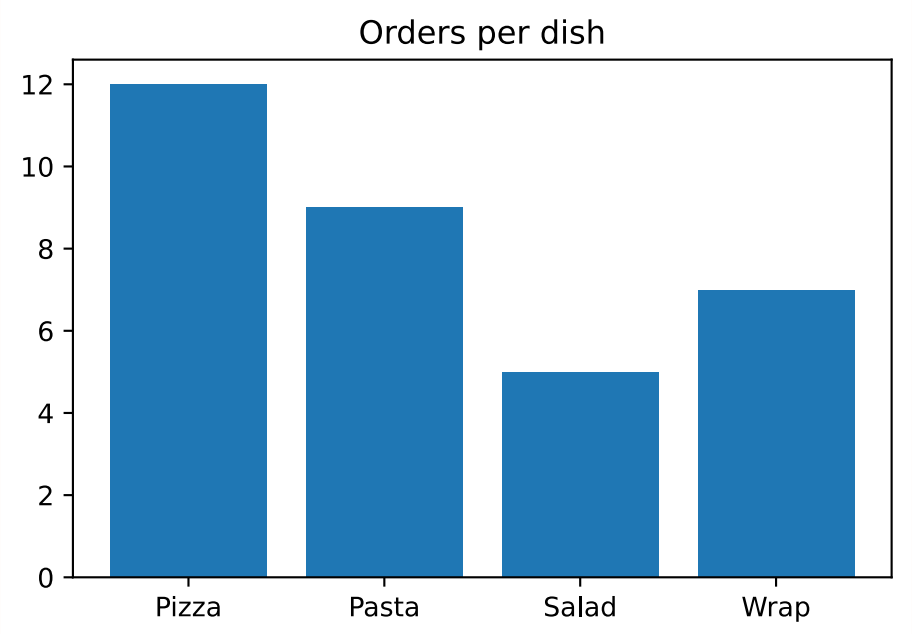
Categories → bar

"Which dish sells best?" Categories to compare, so that's a **bar**:

```
import matplotlib.pyplot as plt

dishes = ["Pizza", "Pasta", "Salad", "Wrap"]
sold = [12, 9, 5, 7]

plt.figure()
plt.bar(dishes, sold)
plt.title("Orders per dish")
plt.gca()
```



The tallest bar answers the question at a glance: Pizza.

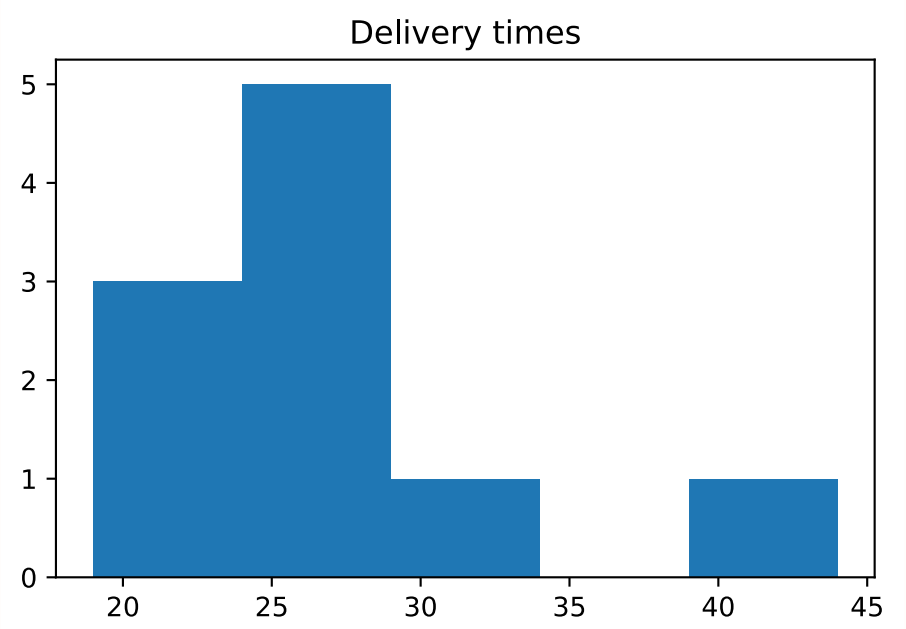
A distribution → histogram

"How are our delivery times spread out?" One column of numbers dropped into buckets. That's a **histogram**:

```
import matplotlib.pyplot as plt

minutes = [19, 22, 23, 24, 24, 25, 26, 28, 31, 44]

plt.figure()
plt.hist(minutes, bins=5)
plt.title("Delivery times")
plt.gca()
```



Most deliveries land in the mid-20s, with one lonely slow one far right. A histogram shows **shape**, not individual values.

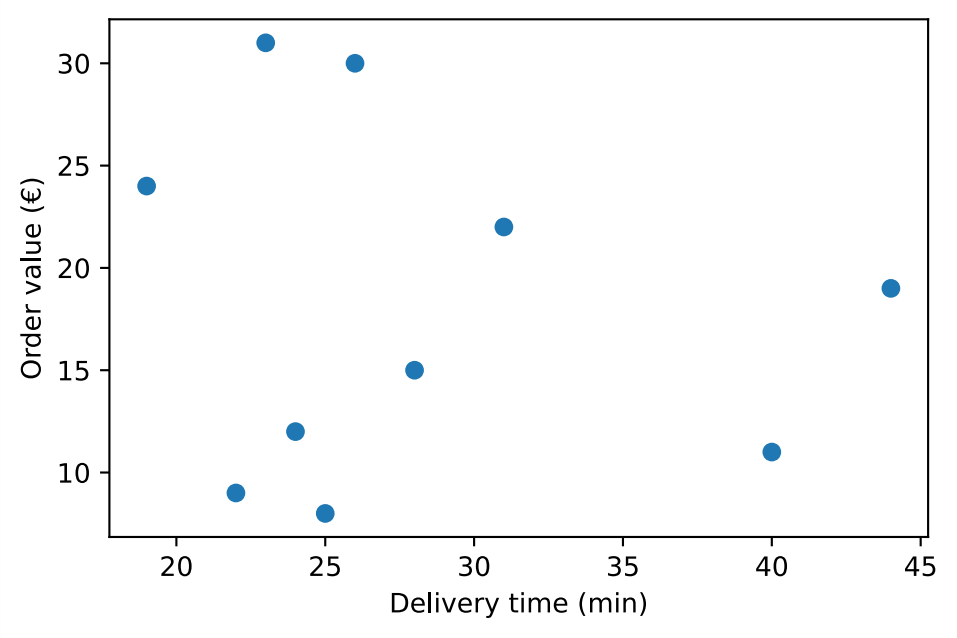
Two numbers → scatter

"Do bigger orders take longer?" Two numbers per order, one on each axis, which makes it a **scatter**:

```
import matplotlib.pyplot as plt

minutes = [19, 22, 23, 24, 25, 26, 28, 31, 40, 44]
euros   = [24, 9, 31, 12, 8, 30, 15, 22, 11, 19]

plt.figure()
plt.scatter(minutes, euros)
plt.xlabel("Delivery time (min)")
plt.ylabel("Order value (€)")
plt.gca()
```



Just a cloud. No upward drift. Sometimes the honest answer is **"there's no pattern here."**

Predict: how many bars?

The **same four numbers**, two ways. On the left, a bar per number; on the right, a histogram:

```
values = [8, 12, 9, 15]
plt.bar(range(4), values)   # left
plt.hist(values, bins=3)    # right
```

The bar chart shows 4 bars. How many bars does the **histogram** show?

a) 4: one bar per number b) 15: one bar per unit up to the max c) about 2–3 lumps: it counts ranges

. . .

Predict first. Pick a letter, then I reveal the answer.

Answer: about 2–3 lumps

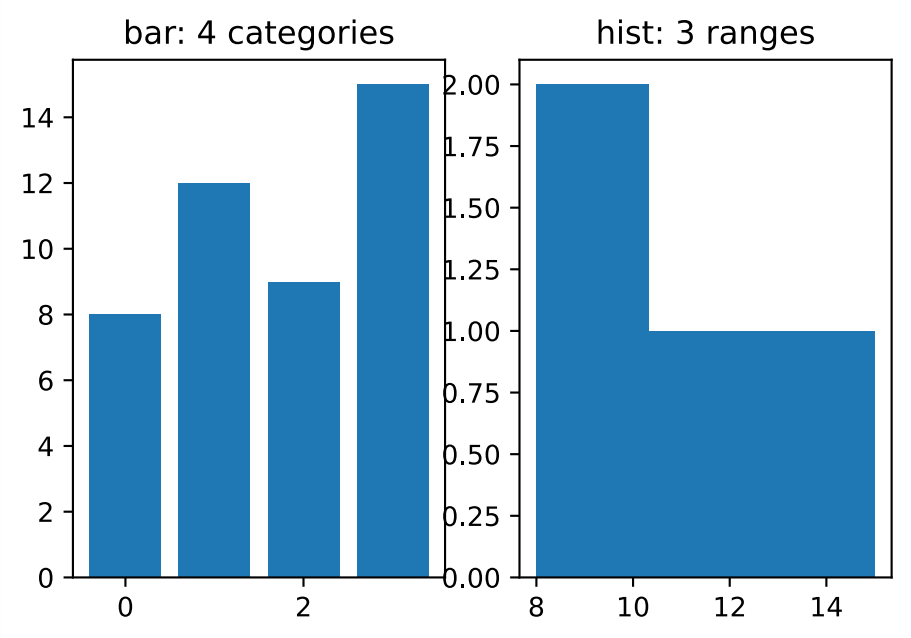
c) about 2–3 lumps: the histogram groups the numbers into ranges and counts how many fall in each. `8` and `9` land in the same bucket, so that bar is two tall:

```
import matplotlib.pyplot as plt

values = [8, 12, 9, 15]

plt.figure()
plt.subplot(1, 2, 1)
```

```
plt.bar(range(4), values)     # one bar per NUMBER
plt.title("bar: 4 categories")
plt.subplot(1, 2, 2)
plt.hist(values, bins=3)      # counts how many per RANGE
plt.title("hist: 3 ranges")
plt.gca()
```



. . .

Bar counts CATEGORIES; histogram counts RANGES. Same numbers, different question.

Your turn — 10 minutes

Open the exercise (scan the QR or type the link):

python.tobiasvlcek.com/notebooks/ex_09_b/

First **predict** what happens, then run it.

> Tip
>
> A five-minute pause after this one, then honest charts.

Honest charts + the AI chart assistant

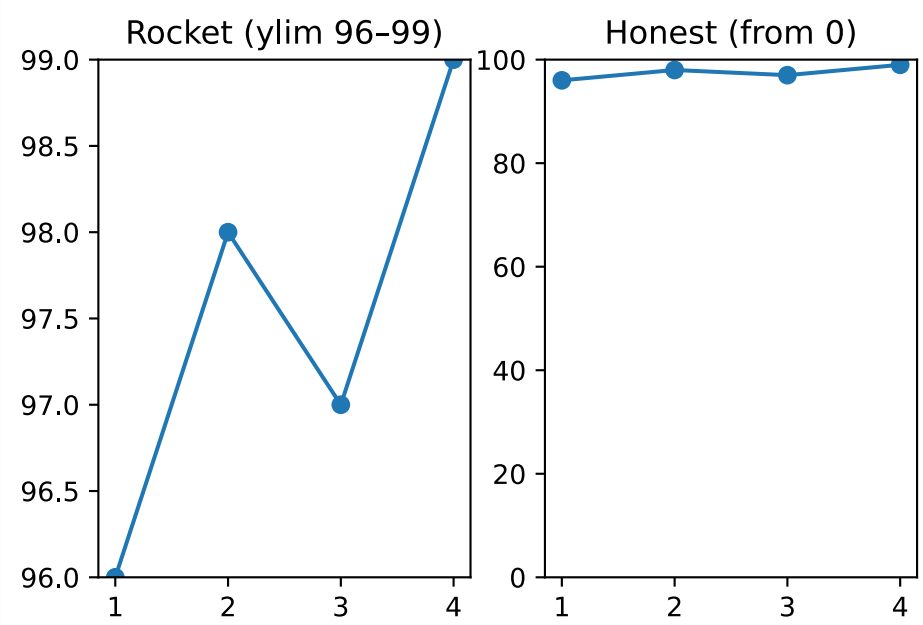
The same data, told two ways

Four weeks of revenue, `[96, 98, 97, 99]`, barely moving. **Where the y-axis starts** decides: the left gets you a term sheet, the right gets you trusted. Same numbers, and the room sees a rocket or the truth:

```
import matplotlib.pyplot as plt

weeks = [1, 2, 3, 4]
revenue = [96, 98, 97, 99]

plt.figure()
plt.subplot(1, 2, 1)
plt.plot(weeks, revenue, marker="o")
plt.ylim(96, 99)             # tight: DRAMA
plt.title("Rocket (ylim 96–99)")
plt.subplot(1, 2, 2)
plt.plot(weeks, revenue, marker="o")
plt.ylim(0, 100)             # from zero: TRUTH
plt.title("Honest (from 0)")
plt.gca()
```



The rule: growth starts at zero

- A **growth claim** on a chart starts the y-axis at **0**. Anything else magnifies a wobble into a cliff.
- The investor **will** check: she reads the axis before she reads the line.

. . .

> Tip
>
> She's the type who reads Formular 27b/6 for fun. A stretched axis is the first thing she catches, and the last thing you want to explain.

AI as your chart assistant

AI is genuinely good at plotting code, if you drive it like a co-pilot:

1. **Describe** the data and the question: *"I have* `orders` *with columns* `zone` *(text) and* `total_eur` *(float). Bar chart of total revenue per zone."*
2. Let it **draft** the plot code.
3. **Verify** three things before you believe the picture:
   - Do the **columns** it used actually exist?
   - Does the **y-axis** start where you claim?
   - Does the **chart type** fit the question?

. . .

The pilot flies; the co-pilot advises. You still land the plane.

Two ways AI charts lie

- **Invented column names.** The AI writes `orders["revenue"]`, but your column is `total_eur`: a `KeyError`, exactly the confident-nonsense you caught in Episode 8. Read the code before you run it.
- **Silently "dramatic" defaults.** Ask for a growth chart and some assistants hand back a **tight y-axis** that flatters the trend: no warning, no comment. The chart runs; that's what makes it dangerous.

. . .

> ! Important
>
> Both threads, one rule: **an AI chart runs long before it's true.** You verify the columns *and* the axis, every time.

Your turn — 10 minutes

Open the exercise (scan the QR or type the link):

python.tobiasvlcek.com/notebooks/ex_09_c/

First **predict** what happens, then run it.

Before Session X

The finale leaves the browser for **your own machine**. Four things before you arrive, in this order, about 45 minutes:

1. **GitHub account** from Session I. None yet? github.com, tonight
2. **git + GitHub CLI (`gh`)**: install, log in once. Git Basics
3. **uv**: the Python environment manager. uv guide
4. **Zed + Mistral Vibe**: editor and the AI agent inside it, on your Session VI key. AI-tools guide

...

> ! Important
>
> **Do it in advance**, and bring the laptop plus one downloaded lab `.py`. Session X opens with Checkpoint 5 (Sessions VIII–IX), then builds on a working toolchain. Install fights you? Ask in class or by e-mail.

To the Lab

After the break: the lab

- Head to the lab notebook: Episode 9 — The Pitch Deck
- Same eighty orders, now with pictures: a line for daily revenue, a bar for zones, a histogram for the value spread, a scatter that finds **no** pattern, and Tobi's cliff-edge growth slide, which you'll flatten into the truth
- Every chart opens with `plt.figure()` and closes with `plt.gca()`
- AI is allowed: draft with it, then **verify** the columns and the axis before you believe it
- It runs entirely in your browser: no setup, just click and code

...

> ! Important
>
> **Download your `.py` before you leave.** It's the appendix of your pitch: the file that proves every chart came from the real data.

Wrap-up

Three things to remember

1. **Every chart, one frame.** Open with `plt.figure()`, close with `plt.gca()`; label the axes and title it. A chart nobody can read argues nothing.
2. **Match the chart to the question.** Categories → bar, distribution → histogram, two numbers → scatter, time → line. The wrong chart is a lie; and "no pattern" is a real finding.
3. **Honest axis, verified code.** Growth charts start at **0**, and every AI-drafted plot gets checked: do the columns exist, does the axis start where you claim?

...

> i Note
>
> **Season finale next: git, real files, and the project kickoff.** We leave the browser for your own machine: the toolchain you install this week is the price of admission.

Literature

Books to start with

- Wilke, C. (2019). Fundamentals of data visualization: A primer on making informative and compelling figures (First edition). O'Reilly Media. Link to the free book website
  - The best single book on *why* a chart works: principles over code. Highly recommended.
- Downey, A. B. (2024). Think Python: How to think like a computer scientist (Third edition). O'Reilly. Link to free online version

. . .

> **Note**
>
> Building charts with AI this session? Revisit the AI Tools page: describe-then-verify is the whole workflow. Matplotlib's own pyplot tutorial is a friendly next step.

. . .

For more, see the literature list of this course.