Lecture VIII - Pandas and AI Craft

Programming with Python

Dr. Tobias Vlćek
Kühne Logistics University Hamburg - Fall 2026

📋 Checkpoint 4

Sessions VI–VII. The first 40 minutes are the checkpoint. It starts now, before the investor opens her data room.

- **Individual work**: no neighbors, no chat
- **AI tools are allowed**: being able to **VERIFY** the output is the skill being graded
- The **link and QR** are handed out in class. Open it and start
- ~6 short tasks: write code, fix a bug, a seeded draw, answer a multiple choice
- It sweeps **Sessions VI–VII**: modules, `random` and seeds, NumPy arrays and masks

When you're done

Menu → *Download* → *Download Python code* → upload the `.py` to the **"Checkpoint 4"** assignment on Moodle. **No retakes**: one sitting.

. . .

> ℹ Note
>
> The green ✅ live checks are **provisional**. The final grading runs on my side. And take a breath: everything in it was rehearsed in the labs.

Episode 8: The Data Room

Show me the data

Pens down. The checkpoint is behind you. This morning the rival chain MunchCorp put out a press release bragging about their "data-driven growth." The evidence attached: exactly **one pie chart.** The investor slid it across the table and said, *"Yours will be better. Show me the data."*

. . .

Tobi didn't wait. He'd pasted the question into a chatbot, proudly turned the laptop around, and hit run:

```
AttributeError: 'DataFrame' object has no attribute 'overview'
```

. . .

The AI sounded certain. The method it called **does not exist.** Today is about that difference, and about the tool that answers the investor for real: **pandas.**

AI joins the team — professionally

Prompting: context + constraints

Two things turn a vague request into a useful answer:

- **Context**. Say what the data *is*: "I have a DataFrame `orders` with columns `zone` (text) and `total_eur` (float)."
- **Constraints**. Say exactly what you want *back*: "Give me the total `total_eur` for zone `Nord`, as a single number."

...

> *"DataFrame `orders`, text column `zone`, float column `total_eur`. Write one line that returns the total `total_eur` where `zone` is `Nord`."*

Vague in, vague out. Specific in, checkable out, and then you **iterate**.

The verify workflow

Tobi's mistake wasn't *using* AI. It was **shipping without checking.** Every line an AI hands you gets three passes:

1. **Read it**: do you understand what each line claims to do?
2. **Run it**: does it actually execute, or does it crash?
3. **Test it**: try it on a small case where **you already know the answer.**

Step 3 in action

Five orders with values you can add in your head, and the AI's suggested line for the Nord average:

```
import pandas as pd

check = pd.DataFrame({
    "zone":      ["Nord", "Sued", "Nord", "Hafen", "Nord"],
    "total_eur": [10.0, 7.5, 20.0, 5.0, 30.0],
})
print(check[check["zone"] == "Nord"]["total_eur"].mean())
```

```
20.0
```

...

The Nord orders are 10, 20 and 30. You can average those in your head. Does the AI's answer match? Then the line has earned some trust on eighty rows. Verification is the job now, not typing.

Predict: does `.summarize()` exist?

The AI wrote this for Tobi. `orders` is a tiny two-row frame. What happens on the last line?

```
import pandas as pd
orders = pd.DataFrame({"zone": ["Nord", "Sued"], "total_eur": [12.0, 9.5]})

orders.summarize()
```

a) prints a summary table b) raises an `AttributeError` c) returns an empty DataFrame

...

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

Answer: the method never existed

b) **`AttributeError`**: pandas has no `.summarize()`. The AI invented it:

```
import pandas as pd
orders = pd.DataFrame({"zone": ["Nord", "Sued"], "total_eur": [12.0, 9.5]})
orders.summarize()                    # the method the AI invented
```

```
AttributeError: 'DataFrame' object has no attribute 'summarize'
␛[31mAttributeError␛[39m␛[31m:␛[39m 'DataFrame' object has no attribute
'summarize'
```

...

The real method is `.describe()`:

```
print(orders.describe())              # the method that actually exists
```

```
       total_eur
count   2.000000
mean   10.750000
std     1.767767
min     9.500000
25%    10.125000
50%    10.750000
75%    11.375000
max    12.000000
```

...

An AI that sounds sure is not the same as an API that exists.

Disclosure: one line, every time

From Session VI the course rule stands: every submission that used AI carries a **one-line note** saying what you used it for.

- *"Used the chatbot to draft the `groupby` line; I checked the totals by hand."*
- Not a confession. A professional reflex.

. . .

It protects **you**: it separates the work you understand from the work you pasted, so when a reviewer (or the investor) asks "how does this line work?", you're never caught claiming something you can't explain.

When NOT to reach for AI

Sometimes the fastest path is the one you already own. You built a whole muscle in Part I:

- A red traceback? **You've read those since Episode 5.** The last line names the error and points at the file, often faster than describing it to a chatbot.
- A one-line filter you've written ten times? Just write it.

. . .

> Tip
>
> Use AI to draft the unfamiliar and to explain the confusing, not to dodge the thinking you're perfectly able to do. The pilot flies; the co-pilot advises.

Your turn — 10 minutes

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

python.tobiasvlcek.com/notebooks/ex_08_a/

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

The DataFrame

What pandas adds over NumPy

Last session, a NumPy array turned a thousand numbers into one fast object, but every value had to share **one type**, and columns had no names.

- A **DataFrame** is a table: named columns, and each column can be a different type
- Text zones next to float euros next to integer counts, all in one object
- It's the spreadsheet the investor handed you, loaded into Python

. . .

The convention everyone uses: `import pandas as pd`.

A DataFrame from a dictionary

Keys become **column names**; each list becomes a **column**; one order per row, mixed types together:

```
import pandas as pd
df = pd.DataFrame({
    "order_id":  [101, 102, 103, 104, 105],
    "zone":      ["Nord", "Sued", "Nord", "Hafen", "Sued"],
    "items":     [2, 1, 3, 1, 4],
    "total_eur": [18.50, 7.20, 24.00, 6.80, 31.40],
})
print(df)
```

```
   order_id   zone  items  total_eur
0       101   Nord      2       18.5
1       102   Sued      1        7.2
2       103   Nord      3       24.0
3       104  Hafen      1        6.8
4       105   Sued      4       31.4
```

. . .

> **Note**
>
> Nobody types eighty orders by hand: today's lab loads a real CSV in one line, `orders = pd.read_csv("public/orders.csv")`. In the browser that line fetches over the web instead of from disk; your code doesn't change.

First look: `.head()` and `.info()`

Before analyzing a table, glance at it. `.head()` shows the top rows; `.info()` is your first sanity check: right number of rows? Any column a surprising type?

```
print(df.head(3))     # first 3 rows
df.info()             # columns, dtypes, non-null counts
```

```
   order_id  zone  items  total_eur
0       101  Nord      2       18.5
1       102  Sued      1        7.2
2       103  Nord      3       24.0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   order_id   5 non-null      int64
 1   zone       5 non-null      object
 2   items      5 non-null      int64
 3   total_eur  5 non-null      float64
dtypes: float64(1), int64(2), object(1)
memory usage: 292.0+ bytes
```

The numbers at a glance: `.describe()`

`.describe()` computes count, mean, min, max and quartiles for every **numeric** column at once:

```
print(df.describe())
```

```
         order_id    items  total_eur
count    5.000000  5.00000   5.000000
mean   103.000000  2.20000  17.580000
std      1.581139  1.30384  10.688873
min    101.000000  1.00000   6.800000
25%    102.000000  1.00000   7.200000
50%    103.000000  2.00000  18.500000
75%    104.000000  3.00000  24.000000
max    105.000000  4.00000  31.400000
```

. . .

One call, the whole numeric summary: this is the `.summarize()` the AI wished existed, spelled correctly.

Selecting a column, filtering rows

Pick one column by name; keep rows with a **boolean mask**, exactly the NumPy idea from last session:

```
print(df["zone"])                 # one column (a Series)
print()
print(df[df["zone"] == "Nord"])   # only the Nord rows, case-sensitive!
```

```
0     Nord
1     Sued
2     Nord
3    Hafen
4     Sued
Name: zone, dtype: object

   order_id  zone  items  total_eur
0       101  Nord      2       18.5
2       103  Nord      3       24.0
```

...

`df["zone"] == "Nord"` builds a column of `True`/`False`; indexing with it keeps the `True` rows. `.loc` exists for label-based selection, but a plain mask covers today.

Adding a column, sorting

Compute a new column from existing ones, then sort. Work on a **copy** so the original stays intact:

```
priced = df.copy()
priced["eur_per_item"] = priced["total_eur"] / priced["items"]
print(priced.sort_values("total_eur", ascending=False))
```

```
   order_id   zone  items  total_eur  eur_per_item
4       105   Sued      4       31.4          7.85
2       103   Nord      3       24.0          8.00
0       101   Nord      2       18.5          9.25
1       102   Sued      1        7.2          7.20
3       104  Hafen      1        6.8          6.80
```

...

`sort_values` reorders rows by a column, biggest orders first here. The new column is just arithmetic on two others, computed for every row at once.

Predict: filtering with the wrong case

Tobi filters for the Nord zone, but types it **lowercase**. The data spells it `"Nord"`. What does this print?

```
print(df[df["zone"] == "nord"])
```

a) an empty table, no error b) the Nord rows anyway: case is ignored c) a `KeyError` for the missing zone

...

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

Answer: an empty table, no warning

a) an empty table. `"nord"` matches nothing, so the mask is all `False` and pandas hands back **zero rows**. No error, no complaint:

```
print(df[df["zone"] == "nord"])          # nothing matches "nord"
print("rows:", len(df[df["zone"] == "nord"]))
```

```
Empty DataFrame
Columns: [order_id, zone, items, total_eur]
Index: []
rows: 0
```

...

This is the dangerous kind of bug: it **runs**, it just returns nothing. **pandas won't warn you; YOU verify**, which is exactly why you test filters on a case whose answer you know.

One more tool: `groupby`

The investor's real question is **per zone**: which area brings in the most?

- `groupby` splits the table by a category, then computes **once per group**
- `df.groupby("zone")["total_eur"].sum()` → one total for each zone

...

> **i** Note
>
> That's the teaser: today's lab does the heavy lifting with `groupby`, turning eighty raw orders into the handful of numbers the investor actually asked for.

Your turn — 10 minutes

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

python.tobiasvlcek.com/notebooks/ex_08_b/

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

To the Lab

After the break: the lab

- Head to the lab notebook: Episode 8 — The Data Room, in your browser as always
- The investor slides a **USB stick** across the table: every order of two weeks, now `orders.csv`, eighty rows, one `pd.read_csv` line
- You'll `.head()` it, ask its `.shape`, filter with masks, add a column on a safe copy, and answer per-zone questions with `groupby`
- Tobi has discovered AI; your real job is to **supervise** it and catch the confident nonsense

. . .

> ! Important
>
> **Download your `.py` before you leave.** Closing the tab without downloading loses your work, the same motion you used to hand in the checkpoint at the start of the session.

Wrap-up

Three things to remember

1. **AI is a co-pilot you verify.** Give it context and constraints, then read it → run it → test it on a case you know.
2. **A DataFrame is a named, mixed-type table.** `pd.DataFrame` from a dict, `pd.read_csv` from a file; `.head()`, `.info()`, `.describe()` to look before you leap.

3. **Select, filter, add, group.** `df["col"]`, a boolean mask (`df[df["zone"] == "Nord"]`, case-sensitive!), a new column on a `.copy()`, and `groupby` for per-category answers. pandas won't warn you; you verify.

. . .

> **Note**
>
> **Next episode: charts the investor can't argue with.** Numbers convince the careful; a good plot convinces the room. We turn the data room into pictures.

Literature

Books to start with

- Downey, A. B. (2024). Think Python: How to think like a computer scientist (Third edition). O'Reilly. Link to free online version
- Elter, S. (2021). Schrödinger programmiert Python: Das etwas andere Fachbuch (1. Auflage). Rheinwerk Verlag.

. . .

> **Note**
>
> Working with AI this session? Revisit the AI Tools page: context-and-constraints prompting, the verify workflow, and the one-line disclosure habit. For pandas itself, the official 10 minutes to pandas guide is the friendliest next step.

. . .

For more, see the literature list of this course.