

Working with data: inspect, fix, create

Programming & Data Management, Part II (Python) · Session 2

Magnus Lodefalk
Örebro University

NA446A, Örebro University
Autumn 2026



Today's goals

By the end of today you can:

1. Apply the **intake checklist** to a variable: type, missing values, range
2. **Fix types and missing values**, including sentinel codes such as 999
3. **Create, subset, sort and aggregate** variables, and summarise data in a table and a figure

From last time

Answer on your own first, then compare with a neighbour

1. You have cleaned a dataset. Where does your code save it, and why not in `data/raw/`?
2. Write the pandas line that gives mean mpg by origin.

From last time

Answer on your own first, then compare with a neighbour

1. You have cleaned a dataset. Where does your code save it, and why not in data/raw/?

Answer: In data/processed/. Raw data is read-only, so anyone can rerun from the original

2. Write the pandas line that gives mean mpg by origin.

Answer: `cars.groupby("origin")["mpg"].mean()`. Today: what else groupby can do

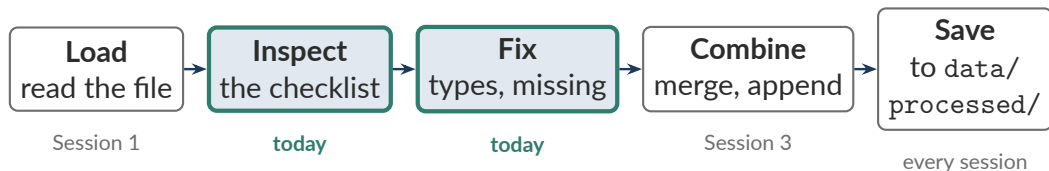
Before we start: open the course folder, not the lab

na446a-starter/ ← **open this folder in VS Code**
 .venv/ the course Python, with pandas
 s1-lab/
 s2-lab/ ← today's lab goes *inside*

- VS Code asks for *ipykernel*? Not *Install: Change Kernel*, refresh the list (↻), then na446a venv.

Every data task follows one recipe

Today: inspect and fix, then create and aggregate



- Today's two errors never crash a script: a code read as a number, and a missing value counted as a zero.

Six terms for today

<code>dtype</code>	the type of a column: <code>float64</code> (number), <code>int64</code> , <code>str</code> (text)
<code>NaN</code> , <code><NA></code>	pandas' missing value; Stata's <code>.</code>
sentinel code	a number that means "missing", such as <code>-9</code> or <code>999</code>
True/False mask	a True/False column that picks rows: <code>cars["mpg"] > 40</code>
<code>groupby(...).agg</code>	one row per group: Stata's <code>collapse</code>
<code>groupby(...).transform</code>	a group value on every row: Stata's <code>egen ... , by()</code>

- A sentinel code looks like data. Only the range check, or the codebook, gives it away.

The intake checklist: ten questions for every variable

Today we practise the four in green

1. Name, label and unit
 2. Type: number, text or date?
 3. Missing values, incl. sentinel codes
 4. Range: possible min and max?
 5. Distribution and outliers
 6. Categories: levels and typos
 7. Dates: read as dates, no gaps
 8. IDs: unique, never blank
 9. Logic across variables
 10. Documentation of every change
- Ask all ten before any regression. Items 7–8 return in Session 3.

Inspect and fix

Steps 1-6: check, find, fix

	Check	We find	Fix
horsepower	Step 1: types	text ("?")	Steps 2-3: <code>to_numeric</code>
mpg	Steps 4-5: missing, range	999 codes	Step 6: <code>mask</code>

- Two problems, one rhythm: check, find, fix, then count again.

Now I'll show you in VS Code

Step 1, check: are the types right?

```
RAW = Path("data/raw")  
cars = pd.read_csv(RAW / "auto_messy.csv")  
cars.dtypes.head(5)
```

```
mpg           float64  
cylinders      int64  
displacement  float64  
horsepower     str  
weight        int64  
dtype: object
```

- Horsepower is a number, yet pandas read it as text (`str`): at least one value in the column is not a number.

`auto_messy.csv`: the Session 1 file with two typical problems planted. Stata: `describe`.

Step 2, find: which values are not numbers?

```
# trial conversion into hp; cars is unchanged  
hp = pd.to_numeric(cars["horsepower"], errors="coerce")  
# isna = is missing: the rows that failed  
cars.loc[hp.isna(), ["name", "horsepower"]]
```

	name	horsepower
56	toyota corona hardtop	?
71	amc matador (sw)	?
114	chevrolet monte carlo s	?
151	chevrolet nova	?

coerce: unreadable values become NaN. .loc[rows, columns].

- Four cars have “?” for unknown horsepower. Look before you convert.

Step 3, fix: horsepower becomes a number

```
cars["horsepower"] = hp
print(cars["horsepower"].dtype)
print(cars["horsepower"].isna().sum())
```

```
float64
```

```
4
```

- Horsepower is now a number, and the 4 “?” are real missing values. Count before and after: `coerce` turns anything it cannot read into missing, silently.

Stata: `destring horsepower, replace force ("4 missing values generated")`.

Step 4, check: how many values are missing?

```
# after sum(), each row is a variable; head(4): first four  
cars.isna().sum().head(4)
```

```
mpg            0  
cylinders      0  
displacement   0  
horsepower     4  
dtype: int64
```

`isna()` is True where a value is missing; `sum()` counts the Trues (True = 1) in each column. It only sees what pandas already knows is missing.

- According to this, mpg has no missing values. Is that true? The next check decides.

```
Stata: misstable summarize, or count if missing(mpg).
```

Step 5, check: is the range possible?

```
stats = cars[["mpg", "horsepower"]].describe()
stats.loc[["count", "mean", "min", "max"]].round(1)
```

	mpg	horsepower
count	392.0	388.0
mean	30.9	104.3
min	9.0	46.0
max	999.0	230.0

Ford Pinto (sw), 1972, row 79 mpg 999: one of 3 cars where “missing” was written as a number.

- 999 is a **sentinel code**, and it pulls the mean up from 23.4 to 30.9.

Step 6, fix: the 999 code becomes missing

```
is_code = cars["mpg"] == 999
print(is_code.sum())
# mask = tape over the 999s: they become missing, rows stay
cars["mpg"] = cars["mpg"].mask(is_code)
print(cars["mpg"].isna().sum())
print(cars["mpg"].mean().round(1))
```

```
3
3
23.4
```

- All 392 cars kept, 3 now missing; the mean skips them: 23.4.

Stata: mvdecode mpg, mv(999) ("3 missing values generated").

A car with missing mpg: what does `mpg > 25` give it?

Your bet first: hands up. The answer is on the next slide.

- A. 1: a high-mpg car
- B. 0: a low-mpg car
- C. Missing

Stata says 1, pandas says 0, and both are wrong

Stata: `gen high = mpg > 25`

Missing counts as larger than any number, so the car gets **1**: 157 high-mpg cars

pandas: `(cars["mpg"] > 25).astype(int)`

NaN > 25 is False, so the car gets **0**: 154 high-mpg cars, and 3 unknowns counted as low

Ford Pinto (sw) Stata calls it a high-mpg car, pandas a low-mpg car. We do not know its mpg.

- Keep missing values missing: Step 8 shows how. Verified in both languages.

Create, subset and aggregate

Steps 7–13: the question each step answers

	Question	Tool (Stata)
7	mpg in km per litre, origin as names?	new column, map (gen, label)
8	high mpg yes/no, Pinto still unknown?	where (gen ... if)
9	which cars do more than 40 mpg?	.loc, sort_values (keep, gsort)
10	how do the regions compare?	groupby().agg(collapse)
11	how does each car compare with its region?	transform(egen, by())
12–13	how do we show it?	a figure, saved by code

- Every step answers one question. Say the question before you write the code.

Now I'll show you in VS Code

Step 7: mpg in km per litre, origin as names

```
cars["kpl"] = cars["mpg"] * 0.425144
names = {1: "US", 2: "Europe", 3: "Japan"}
cars["region"] = cars["origin"].map(names)
cars[["mpg", "kpl", "region"]].head(3).round(1)
```

	mpg	kpl	region
0	18.0	7.7	US
1	15.0	6.4	US
2	18.0	7.7	US

- A new column is computed for every row at once, with no loop. `map` turns codes into labels: check them against the codebook (Lab 1's bug).

Stata: `gen kpl = mpg * 0.425144; label define and label values`. One mile per US gallon is 0.425144 km per litre.

Step 8: a high-mpg dummy, unknowns stay unknown

```
ok = cars["mpg"].notna()
high = (cars["mpg"] > 25).astype("Int8")
cars["high_mpg"] = high.where(ok)
cars["high_mpg"].value_counts(dropna=False)
```

```
high_mpg
0         235
1         154
<NA>         3
Name: count, dtype: Int64
```

- $235 + 154 + 3 = 392$: every car counted, and the Pinto stays unknown.

`where(ok)` keeps a value where `ok` is True, missing elsewhere (the mirror of `mask`); `Int8` is an integer type that can hold `<NA>`. Stata: `gen high = mpg > 25 if !missing(mpg)`.

Step 9: which cars do more than 40 mpg?

```
fast = cars.loc[cars["mpg"] > 40,  
                ["name", "mpg"]]  
fast = fast.sort_values("mpg",  
                        ascending=False)  
fast.head(3)
```

	name	mpg
320	mazda glc	46.6
327	honda civic 1500 gl	44.6
323	vw rabbit c (diesel)	44.3

- A mask picks the rows (8 cars), a list the columns; cars is unchanged.

Stata: `gsort -mpg, list name mpg if mpg > 40.` pandas returns a sorted copy (assign it); row labels travel with the rows; missing values go last.

Step 10: how do the regions compare? (agg)

```
by_region = cars.groupby("region").agg(  
    n=("mpg", "count"),  
    mean_mpg=("mpg", "mean"),  
    mean_hp=("horsepower", "mean"))  
by_region.round(1)
```

	n	mean_mpg	mean_hp
region			
Europe	67	27.6	80.6
Japan	79	30.5	79.6
US	243	20.0	118.9

- One row per region, as Stata's collapse; the means of Session 1.

Step 11: each car against its region (transform)

```
grp = cars.groupby("region")["mpg"]
cars["region_mpg"] = grp.transform("mean")
cars["gap"] = cars["mpg"] - cars["region_mpg"]
show = cars[["mpg", "region_mpg", "gap"]]
show.head(3).round(1)
```

	mpg	region_mpg	gap
0	18.0	20.0	-2.0
1	15.0	20.0	-5.0
2	18.0	20.0	-2.0

- Still 392 rows: each car gets its region's mean, so you can compare the car with its group.

Stata: `bysort origin: egen region_mpg = mean(mpg).`

Check your understanding

Answer on your own first, then compare with a neighbour

You want each car's horsepower relative to the mean horsepower of cars with the same number of cylinders. `agg` or `transform`? How many rows does the result have?

Check your understanding

Answer on your own first, then compare with a neighbour

You want each car's horsepower relative to the mean horsepower of cars with the same number of cylinders. `agg` or `transform`? How many rows does the result have?

Answer: `transform`: 392 rows, one per car, each with its cylinder group's mean. `agg` would give 5 rows, one per cylinder count, and no car to compare.

Show

Now I'll show you in VS Code

Step 12: show the region table as a figure

```
m = by_region["mean_mpg"].sort_values()
fig, ax = plt.subplots(figsize=(6, 2.4))
ax.barh(m.index, m, color="0.35")
ax.set_xlabel("Miles per US gallon (mean)")
ax.spines[["top", "right"]].set_visible(False)
```

fig is the page, ax the drawing area: every change is ax.something. Grey bars, sorted, read as a ranking and survive black-and-white printing.

- Draw from the table you already checked, never from numbers typed by hand.

Needs import matplotlib.pyplot as plt at the top of the script. Stata: graph hbar (mean) mpg, over(origin).

Step 13: a title that states the finding, then save

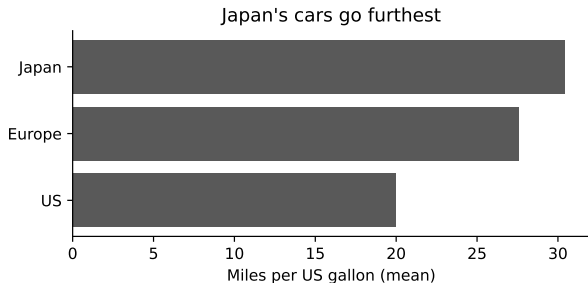
```
ax.set_title("Japan's cars go furthest")
fig.text(0, -0.15, "Source: auto.csv, 389 cars")
for ext in ["png", "pdf"]:
    fig.savefig(f"output/figures/mpg.{ext}",
                bbox_inches="tight")
```

The title states the finding, not the variable; the source line names the data and sample. The loop saves the figure twice: PDF for papers, PNG for slides.

- Figures go to `output/figures/`, written by code, so a rerun redraws them.

Stata: `graph export "output/figures/mpg.pdf", replace.`

What makes a figure publication-ready



Source: auto.csv, 389 cars

The figure exactly as Step 13 saved it.

1. The title states the finding, not the variable
2. Axes labelled, with units
3. A source line: data and sample
4. Grey that survives black-and-white printing
5. Nothing else: no box, grid, 3D or legend for one series

- If a reader must read the text to understand the figure, the figure is not done.

Today's lab

1. Fill in the TODOs in `s2_lab.py`, one cell at a time *goals 1-3*
 2. Check your steps: run `check_s2.py`, fix what it flags *goals 1-3*
 3. **Ask an AI, then check it:** paste your *intake output* (not the data) into a chatbot and ask what problems it shows. Test two of its claims with one line of code each; note claim, test, true or false *rules 3-4*
- A chatbot can read an output well and still be wrong about the data behind it. Your line of code decides.

Never paste personal or register data into a chatbot. Stuck? Use the course tutor prompt (course site, AI page): it gives hints, not answers.

Summary

1. The checklist first: `dtypes`, `isna().sum()`, `describe()` catch a text column and a sentinel code
2. Fix with `to_numeric(errors="coerce")` and `mask`; count missing values before and after every fix
3. Create with `=`, `map` and `where`; pick rows with `.loc`; `agg` gives one row per group, transform one value per row; a figure states its finding and its source

Before next time: the Titanic intake exercise (course site), with its self-check; about 90 minutes. Solution after three days.

Test yourself before next time

Without AI: this is what you should manage on your own

1. `income` was read as `str`. Name two likely causes and the line that fixes it.
2. One value was "1,200". What does `pd.to_numeric(..., errors="coerce")` give it, and how would you notice?
3. Write the line that turns the codes -9 and 999 in `age` into missing values.
4. Do `cars["mpg"] <= 25` and `~(cars["mpg"] > 25)` pick the same cars? Why not?
5. Name three things that make a figure publication-ready.

Appendix

The checklist in code

Question	pandas	Stata
1 Name, label, unit	<code>df.columns</code> , a codebook file	<code>label variable</code>
2 Type	<code>df.dtypes</code> , <code>pd.to_numeric</code>	<code>describe</code> , <code>destring</code>
3 Missing, sentinels	<code>df.isna().sum()</code> , mask	<code>misstable sum</code> , <code>mvdecode</code>
4 Range	<code>df.describe()</code>	<code>summarize</code>
5 Distribution, outliers	<code>df["x"].hist()</code> , <code>quantile</code>	<code>histogram</code> , <code>centile</code>
6 Categories	<code>value_counts(dropna=False)</code>	<code>tab x, missing</code>
7 Dates	<code>pd.to_datetime</code>	<code>date()</code> , <code>tsset</code>
8 IDs and keys	<code>df["id"].is_unique</code>	<code>isid id</code>
9 Cross-variable logic	<code>assert (df["wage"] > 0).all()</code>	<code>assert wage > 0</code>
10 Documentation	comments, README, codebook	notes, do-file

count or size? And two kinds of missing

- `count` counts non-missing values: Europe has 67 cars with mpg
- `size` counts rows, missing or not: Europe has 68 cars
- Report the one you mean; a table of means should show the count
- `NaN` is the missing value of ordinary number columns (`float64`): `NaN > 25` is `False`
- `<NA>` is the missing value of types with a capital letter (`Int8`, `Float64`): `<NA> > 25` is `<NA>`, so missing stays missing

One more trap: “25 or less” is not “not above 25”

- `len(cars[cars["mpg"] <= 25])` gives 235
- `len(cars[~(cars["mpg"] > 25)])` gives 238. The tilde (~) means *not*: it flips True and False (Stata's !)
- The difference is the 3 cars with missing mpg: `NaN > 25` is False, and *not* False is True, so they are kept
- In Stata both give 235, since missing counts as greater than 25
- Write the condition you mean, and count the rows it keeps.