

Combining data: merge, append, reshape, lag

Programming & Data Management, Part II (Python) · Session 3

Magnus Lodefalk
Örebro University

NA446A, Örebro University
Autumn 2026



Today's goals

By the end of today you can:

1. **Merge** two tables with key checks: `validate=`, `indicator=`, and the number of rows predicted before you run
2. **Append** yearly files and **reshape** between long and wide
3. Build a **person-year panel** with a lag that is really last year's value

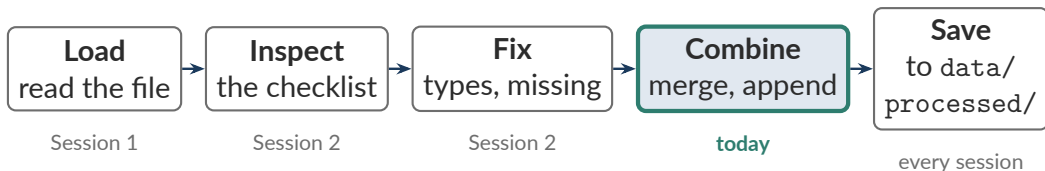
From last time

Answer on your own first, then compare with a neighbour

1. A person's age is missing. What do Stata's `gen old = age > 65` and pandas' `(df["age"] > 65).astype(int)` give that person?
2. Each worker's firm mean wage on the worker's own row: `agg` or `transform`?

Every data task follows one recipe

Today: combine



- Today's errors never crash a script either: rows that vanish in a merge, and rows that multiply.

Eight terms for today

key	the column(s) that identify one row: <code>LopNr</code> ; in a panel <code>LopNr</code> and <code>year</code>
merge	add columns from another table by matching keys (Stata <code>merge</code>)
append, concat	stack the rows of tables (Stata <code>append</code>)
how=	which unmatched rows a merge keeps: "inner", "left", "outer"
validate=	the key shape you promise, e.g. "m:1"; pandas stops if broken
indicator=True	a column <code>_merge</code> : both, left_only, right_only
wide, long	one row per unit / one row per unit and period (Stata <code>reshape</code>)
lag	the same unit's value one period earlier (Stata <code>L.</code>)

- Every term on this slide is about the key. Check it before you combine.

Rule A: a key is unique and never missing

```
RAW = Path("pair_00/data/raw")
persons = pd.read_csv(RAW / "persons.csv", dtype=str)
print(len(persons), persons["LopNr"].nunique())
print(persons["LopNr"].is_unique)
```

```
1005 1000
False
```

“The variables that form the key never take on missing values, and a key’s value is never duplicated across rows of the table.” (Gentzkow and Shapiro 2014, ch. 5, today’s reading)

- 1,005 rows, 1,000 persons: the file breaks rule A, and every merge on LopNr inherits it.

`nunique()`: how many different ids; `is_unique`: True if none repeats. Pair 00, read as text. Stata: `isid lopnr`.

Steps 1–6: count the rows in and out

Question	Tool (Stata)
1 what does a merge keep by default?	merge with no how= (merge)
2 which rows matched, and which did not?	how="outer", indicator=True (_merge)
3 how do we stop the rows from multiplying?	validate="m:1" (merge m:1)
4 which ids repeat, and are they copies?	duplicated (duplicates report)
5 did every row match, and none multiply?	assert on the rows (merge, assert())
6 how do we stack five yearly files?	concat (append)

- Every step ends with the same question: how many rows went in, and how many came out?

Two small tables for the merge mechanics

```
people = pd.DataFrame({"pid": [1, 2, 3, 4],  
                        "firm": ["A", "B", "B", "C"]})  
firms = pd.DataFrame({"firm": ["A", "B", "D"],  
                      "size": [12, 40, 7]})
```

	pid	firm
0	1	A
1	2	B
2	3	B
3	4	C

	firm	size
0	A	12
1	B	40
2	D	7

Firm C has no row in firms;
firm D has no people.

`pd.DataFrame({column:
values})` types a small table
by hand.

- `firm` is a key in `firms` and a *foreign* key in `people` (it points to rows of another table): many people, one firm row.

Step 1: what does a merge keep by default?

```
# add the firm columns to each person, matched on firm
m = people.merge(firms, on="firm")
m
```

	pid	firm	size
0	1	A	12
1	2	B	40
2	3	B	40

- Four people in, three rows out. Person 4 vanished without a word: pandas keeps only keys found in *both* tables.

Stata's `merge m:1 firm using firms` keeps all five rows and reports `_merge`: 1 master only, 2 using only, 3 matched. The opposite default.

Step 2: which rows matched, and which did not?

```
m = people.merge(firms, on="firm",  
                 how="outer", indicator=True)
```

m

	pid	firm	size	_merge
0	1.0	A	12.0	both
1	2.0	B	40.0	both
2	3.0	B	40.0	both
3	4.0	C	NaN	left_only
4	NaN	D	7.0	right_only

- left_only: in people, no firm (person 4). right_only: a firm with no people (D). Now you can decide what to keep.

pid shows 1.0: a column that gets a missing value becomes a decimal column. Stata: tab _merge.

Firm B listed twice: rows after a left merge?

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

- A. 4: a left merge keeps one row per person
- B. 5
- C. 6

A left merge kept every person and still added rows

firms2: A 12, B 40, B 45; a left merge of people on firm

- Person 1 (firm A): one row
- Persons 2 and 3 (firm B): two rows each, one for each B row
- Person 4 (firm C): one row, with a missing size
- 4 people in, 6 rows out. Nothing warns you: the key on the right was not unique.

Step 3: how do we stop the rows from multiplying?

```
people.merge(firms2, on="firm", how="left",  
             validate="m:1")
```

```
MergeError: Merge keys are not unique in right  
dataset; not a many-to-one merge
```

```
Duplicates in right:  
  firm  
    B ...
```

- "m:1" is your promise: many people per firm, one row per firm. The data break it, so pandas stops and names the key.

Stata: merge m:1 stops too: "firm does not uniquely identify observations in the using data".

Step 4: which ids repeat, and are they copies?

```
print(persons.duplicated("LopNr").sum()) # same id
print(persons.duplicated().sum()) # same in every column
persons = persons.drop_duplicates()
dup = persons.duplicated("LopNr", keep=False) # all copies
persons.loc[dup, ["LopNr", "FodelseLandNamn"]]
```

		LopNr	FodelseLandNamn
5			
3	21	P000182	SVERIGE
	54	P000358	SVERIGE
	217	P000358	NORGE
	988	P000182	NORGE

5 repeated ids: 3 exact copies, 2 persons with two *different* records, which `drop_duplicates()` cannot see.

- Drop the copies. For the conflicts: decide, and log what you decided.

Step 5: did every row match, and none multiply?

```
lisa23 = pd.read_csv(RAW / "lisa_2023.csv", dtype=str)
persons = persons.drop_duplicates("LopNr")
m = lisa23.merge(persons, on="LopNr", how="left",
                 validate="m:1", indicator=True)
assert len(m) == len(lisa23)
m["_merge"].value_counts().to_dict()  # count each value
```

```
['both': 971, 'left_only': 0, 'right_only': 0]
```

First line: keep the first of the two records (logged: P000182, P000358).

- All 971 match, and the assert proves that no row was added. Four parts: how, validate, indicator, assert.

Stata: merge m:1 lopnr using persons, assert(master match).

Step 6: how do we stack five yearly files?

```
frames = []  
for y in range(2019, 2024):  
    d = pd.read_csv(RAW / f"lisa_{y}.csv", dtype=str)  
    frames.append(d.assign(year=y))  
pd.concat(frames).shape
```

```
| (4884, 27)
```

f"lisa_{y}.csv" puts the year into the file name; assign(year=y) adds the year, which the files do not carry; concat stacks the list of tables.

- 4,884 rows: $975 + 984 + 978 + 976 + 971$, so the rows add up. The columns do not: see the next question.

Stata: a foreach loop with append using.

Check your understanding

Answer on your own first, then compare with a neighbour

Each yearly file has 21 columns, with year. The stack has 27. Where do the extra columns come from? (concat lines columns up by name.)

Steps 7–13: the question each step answers

	Question	Tool (Stata)
7	why does the stack have extra columns?	share filled per year (tabstat)
8	how do we join the two halves again?	rename, then stack (rename)
9	are person and year a key?	duplicated (isid lopnr year)
10	how do we get one row per person?	pivot (reshape wide)
11	back to one row per person-year?	melt (reshape long)
12	what was each person's income last year?	groupby, shift (L.)
13	is the lag really last year's value?	the year gap, where (xtset)

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

Step 7: why does the stack have extra columns?

```
lisa = pd.concat(frames, ignore_index=True)
filled = lisa.notna().groupby(lisa["year"]).mean()
filled[["SyssStat11", "SyssStat19"]]
```

	SyssStat11	SyssStat19
year		
2019	1.0	0.0
2020	1.0	0.0
2021	0.0	1.0
2022	0.0	1.0
2023	0.0	1.0

- Share filled per year (notna: True if filled): employment status sits in two columns, each empty in some years.

Step 8: how do we join the two halves again?

```
RENAME = {"SyssStat19": "SyssStat11"}  
for i in range(len(frames)): # i = 0, 1, 2, 3, 4  
    frames[i] = frames[i].rename(columns=RENAME)  
lisa = pd.concat(frames, ignore_index=True)  
print(lisa.shape)
```

```
| (4884, 26)
```

- One column fewer. In the lab you fill in the other renames from the codebook, and the filled-share test tells you whether the codebook was complete.

rename takes a dictionary {old name: new name}. Stata: rename sysssstat19 sysssstat11 inside the loop, before append.

Step 9: are person and year a key?

```
print(lisa.duplicated(["LopNr", "year"]).sum())  
print(lisa.duplicated().sum())  
lisa = lisa.drop_duplicates()  
print(lisa.duplicated(["LopNr", "year"]).sum(),  
      len(lisa))
```

```
1  
1  
0 4883
```

- One person-year appears twice, as an exact copy: drop it. Now (LopNr, year) is a key: 4,883 person-years.

Stata: duplicates report lopnr year, duplicates drop, isid lopnr year.

Step 10: how do we get one row per person?

```
emp = lisa[["LopNr", "year", "SyssStat11"]]  
wide = emp.pivot(index="LopNr", columns="year",  
                  values="SyssStat11")  
wide.head(3)
```

year	2019	2020	2021	2022	2023
LopNr					
P000001	1	6	5	5	6
P000002	1	1	1	1	1
P000003	1	5	1	1	NaN

- One row per person, one column per year, as in a table in a paper. NaN: P000003 is not in the 2023 file.

index: rows, columns: new columns, values: cells. Codes: 1 employed, 5 and 6 not. Stata: reshape wide.

Step 11: back to one row per person-year?

```
long = wide.reset_index().melt(id_vars="LopNr",  
                               value_name="status")  
  
print(len(long))  
long = long.dropna()  
print(len(long), list(long.columns))
```

```
5000
```

```
4883 ['LopNr', 'year', 'status']
```

- 1,000 persons $\times$ 5 years = 5,000 cells. 117 were empty because that person-year does not exist; after dropna we are back at 4,883.

reset_index: LopNr back to a column; melt: the year columns become rows. Keep panels long while you work (Gentzkow and Shapiro's rule B), wide for the final table. Stata: reshape long also keeps the empty cells.

Step 12: what was each person's income last year?

```
lisa["inc"] = pd.to_numeric(lisa["ForvErs"])
lisa = lisa.sort_values(["LopNr", "year"])
g = lisa.groupby("LopNr")
lisa["inc_lag"] = g["inc"].shift(1) # same person, row above
lisa[["LopNr", "year", "inc", "inc_lag"]].head(4)
```

	LopNr	year	inc	inc_lag
0	P000001	2019	2724	NaN
975	P000001	2020	967	2724.0
1959	P000001	2021	1029	967.0
2937	P000001	2022	0	1029.0

- `shift(1)` within each person gives the previous row's income.

ForvErs: income from work, hundreds of SEK. Stata: `xtset lopnr year, then gen inc_lag = L.inc.`

Step 13: is the lag really last year's value?

```
lisa["gap"] = lisa["year"] - g["year"].shift(1)
print(lisa["gap"].value_counts().to_dict())
lisa["inc_lag"] = lisa["inc_lag"].where(lisa["gap"] == 1)
print(lisa["inc_lag"].notna().sum())
```

```
{1.0: 3831, 2.0: 51, 3.0: 1}
3831
```

- 52 “lags” jump over a year in which the person is missing. Keep a lag only where the gap is one year: 3,831 remain.

Persons leave a yearly file for a year (abroad, say). Stata's `L.` uses the calendar after `xtset`, so it gives missing across a gap.

The codebook is data too, and it can be wrong

The register lab's codebook disagrees with its data in three places

- It was written by people, from an earlier delivery, as real codebooks are
- **Your job: find all three.** Today's lab takes you to one of them
- Run checklist items 1 (names), 6 (categories) and 8 (IDs and keys) on *every* file, not only the ones you need
- Write each as an erratum (a correction note): the codebook says · the data say · how you found it
- Where the codebook and the data disagree, the data are what you clean. Read the documentation, then check it.

The same holds for SCB's documentation on MONA: variable names, code lists and special codes (such as —9) change between deliveries.

Today's lab

1. Pick a free number 1-30, cross it out on the whiteboard, download `pair_NN.zip` from the Data page. Late? Use pair 00, in the lab folder
 2. `s3_lab.py`, Steps 1-7: append, key, merge, reshape, lag; check with `check_s3.py` *goals 1-3*
 3. Step 4, *catch the wrong merge an AI proposes*: predict its rows, run it, name two faults, write the right merge *rules 3-4*
 4. Step 8, bug race (12 min): make `clean_registers_flawed.py` run on your data and find bugs in its *outputs*. A bug counts with file, fault and a number
 - A merge that runs is not always right: `validate`, `indicator`, `assert`.
- The data are synthetic; never paste real personal or register data into a chatbot.

Summary

1. Check the key first. `how=` says which unmatched rows you keep (pandas' default is inner, Stata's keeps all); `validate=` stops duplicate keys; `indicator=True` shows what matched; assert the rows
2. Append with a loop and `concat`, renaming *before* you stack, then check the new key; `pivot` to wide, `melt` to long, and keep panels long
3. Sort, `groupby`, `shift`: a lag is last year's value only where the gap is one year

Before next time: the bug hunt (course site), with its self-check; about 2 hours. Solution after three days. Before Session 4: Gentzkow and Shapiro, chapter 2, *Automation*.

Test yourself before next time

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

1. What does pandas' `merge` keep by default, and Stata's? Write the pandas line that keeps what Stata keeps and shows what matched.
2. 1,000 workers, left-merged onto a firm file in which one firm with 30 workers appears twice. How many rows? Which argument would have stopped it?
3. You append five yearly files and get two income columns, each half empty. What happened, and where in the code is the fix?
4. A table has 290 municipalities and one column per year, 2015–2024. Write the line that makes it long. How many rows?
5. Wages 100 in 2021, 120 in 2023, no row for 2022: the 2023 lag with `shift(1)` by person, and with Stata's `L.wage`?

Appendix

pandas and Stata side by side

Task	pandas	Stata
Is it a key?	<code>df["id"].is_unique</code>	<code>isid id</code>
Merge, keep all	<code>how="outer", indicator=True</code>	<code>merge, _merge 1/2/3</code>
Keep the left table	<code>how="left"</code>	<code>keep(master match)</code>
Keep matches only	<code>how="inner" (the default)</code>	<code>keep(match)</code>
Promise the key shape	<code>validate="1:1", "m:1", "1:m"</code>	<code>merge 1:1, m:1, 1:m</code>
Check the result	<code>assert len(m) == len(left)</code>	<code>assert(master match)</code>
Stack rows	<code>pd.concat([a, b])</code>	<code>append using</code>
Wide to long	<code>melt</code>	<code>reshape long</code>
Long to wide	<code>pivot</code>	<code>reshape wide</code>
Lag	<code>groupby(id)[x].shift(1) (a row)</code>	<code>L.x after xtset (a period)</code>

1:1, m:1, 1:m, m:m: which merge is which?

- **1:1**: one row per key on both sides: 2023 person file and 2023 grades per person
- **m:1**: many rows on the left, one on the right: person-years and the person file; workers and firms
- **1:m**: the same, the other way round: firms and their workers
- **m:m**: many on both sides. Every left row meets every right row with the same key, so the rows multiply. Almost always a sign that one table should first be aggregated to the key of the other
- Name the shape before you merge, and write it into `validate=`.

Stata's `merge m:m` does not form all pairs, and its own manual advises against it; `joinby` does. Either way: aggregate first.

The same failure on the register data

- `lisa23` is the 2023 person file (971 rows); `persons`, as loaded, repeats 5 ids
- `lisa23.merge(persons, on="LopNr", how="left", validate="m:1")` stops with the same `MergeError`: `LopNr` is not unique in the *right* table
- Without `validate` it runs and returns 976 rows for 971 persons, with no warning
- The toy example's fault, on real-sized data: Step 4 finds the duplicates.

`Stata: merge m:1 lopnr using persons refuses, r(459).`