Lesson 10: Merging and Joining

Data Science with Python – BSc Course

Data Science Program

BSc Course

45 Minutes

Where We Are

Last time: GroupBy split data into groups and aggregated them

GroupBy summarizes one table. But your data lives in **three different tables**. How do you combine them?

Real-world analysis requires combining data from multiple sources

Learning Objectives

After this lesson, you will be able to:

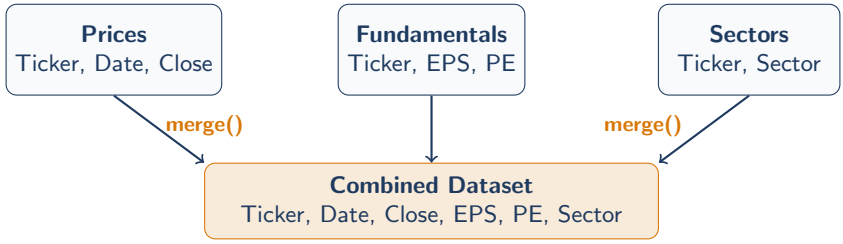
- **Stack** DataFrames vertically and horizontally with `pd.concat()`
- **Perform** SQL-style joins using `pd.merge()`
- **Distinguish** inner, outer, left, and right join types
- **Match** on single keys, multiple keys, and index values
- **Build** a multi-source financial data pipeline

Finance Application: Merge stock prices with sector data for cross-asset analysis.

Merging is the pandas equivalent of SQL JOIN

The Problem: Data in Separate Tables

Your data is in 3 different tables. How do you combine them?

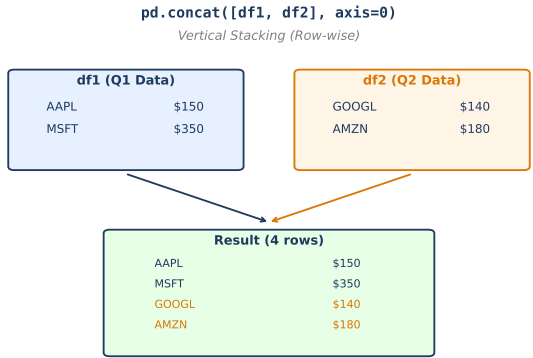


Two tools: **concat** (stacking same-structure data) and **merge** (joining on shared keys).

---

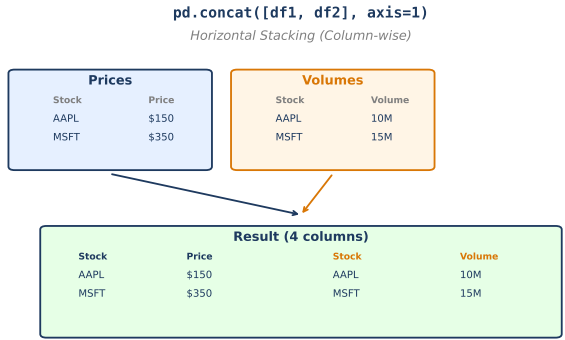
concat = glue tables together. merge = match rows by key columns.

Vertical Concatenation



pd.concat([df1, df2], axis=0) stacks DataFrames vertically

Horizontal Concatenation



pd.concat([df1, df2], axis=1) combines DataFrames side by side

pd.merge(): SQL-Style Joins

Match rows from two DataFrames by a shared key column

df_prices:

Ticker	Price
AAPL	$185
MSFT	$350
GOOGL	$140

df_info:

Ticker	Sector
AAPL	Tech
MSFT	Tech
JPM	Finance

Merge:

```
pd.merge(df_prices, df_info, on='Ticker')
```

Result (inner join):

Only AAPL and MSFT (both tables)

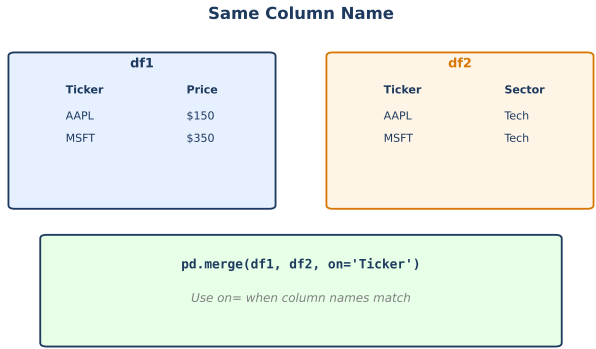
GOOGL dropped (not in df_info)
JPM dropped (not in df_prices)

Key insight: the on= column acts as the matching key between tables.

---

merge() matches rows where the key column values are equal

Inner Join: Only Matching Rows



how='inner' (default) – keeps only rows with matching keys in BOTH tables

Left Join and Right Join

Left Join (`how='left'`)
All rows from **left** table, matches from right.

```
pd.merge(prices, info,
    on='Ticker', how='left')
```

Left: A, B, C — Right: B, C, D
Result: **A, B, C**
A gets NaN for right-side columns

Right Join (`how='right'`)
All rows from **right** table, matches from left.

```
pd.merge(prices, info,
    on='Ticker', how='right')
```

Left: A, B, C — Right: B, C, D
Result: **B, C, D**
D gets NaN for left-side columns

Left join is the most common in practice – you keep all your base data and enrich it.

Left join = keep all from left table + add matching info from right table

Outer Join: Keep Everything

Outer Join (`how='outer'`)
All keys from **both** tables (union).

```
pd.merge(prices, info,
    on='Ticker', how='outer')
```

Left: A, B, C — Right: B, C, D
Result: **A, B, C, D**
A and D get NaN for missing columns

Join Type Summary:

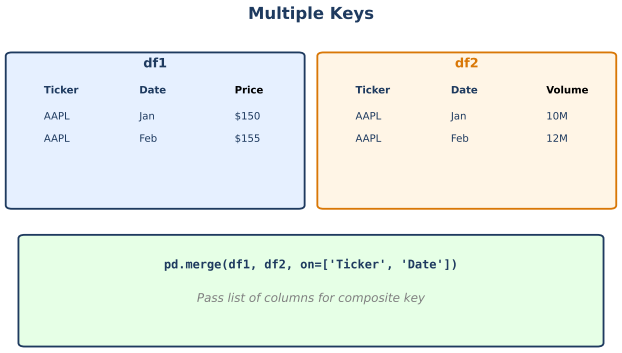
Type	Keeps
inner	intersection
left	all left + matches
right	all right + matches
outer	union (everything)

Default is inner.
Safest for avoiding NaN surprises.

Outer join preserves maximum data but introduces the most NaN values.

Inner = intersection, Outer = union, Left/Right = one side complete

Merge on Multiple Keys



pd.merge(df1, df2, on=['Ticker', 'Date']) for composite keys

Checkpoint: Test Your Understanding

Q1: A left join keeps ALL rows from which table – left or right?

Q2: You merge prices (500 rows) with sectors (30 rows) using inner join. Can the result have MORE than 500 rows? Why?

Q3: When would you use `pd.concat()` instead of `pd.merge()`?

Think for 30 seconds. Answers: Q1: Left table. Q2: Yes – if keys are duplicated (many-to-many). Q3: When stacking same-structure data (e.g., Q1 + Q2 + Q3 sales).

Duplicate keys in merge are the #1 source of unexpected row multiplication

concat() vs merge() vs join()

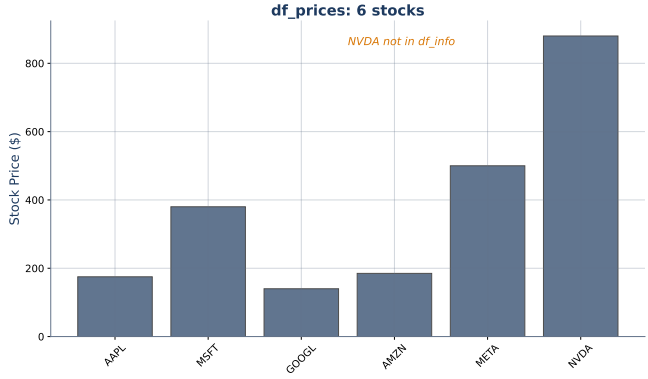
Quick Decision:

- Same columns, different rows? → `concat(axis=0)`
- Different columns, same rows? → `concat(axis=1)`
- Match by a shared key column? → `merge(on='key')`
- Match by index? → `join()` or `merge(left_index=True)`

---

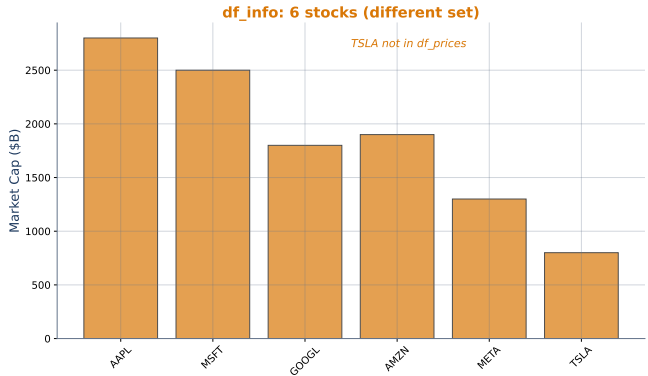
merge() is most flexible; concat() is for stacking; join() is fastest on index

Finance: Merging Prices with Fundamentals



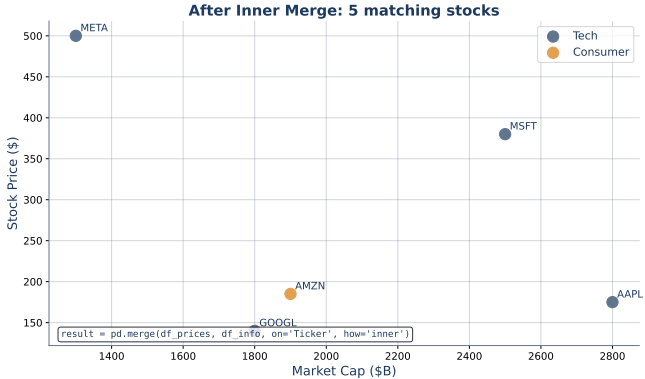
OHLCV price data from market data provider – one source of many

Finance: Company Reference Data



Sector and fundamental data from a separate reference database

Finance: Cross-Source Analysis



After merge: analyze relationships that span multiple data sources

Finance: Multi-Source Data Pipeline

Data Sources:

1. Yahoo Finance (df_prices)
   Ticker, Date, OHLCV
2. SEC Filings (df_fund)
   Ticker, Quarter, EPS
3. Reference (df_sectors)
   Ticker, Sector, Industry

Chained Merges:

```
df = (df_prices
    .merge(df_fund,
           on='Ticker')
    .merge(df_sectors,
           on='Ticker'))
```

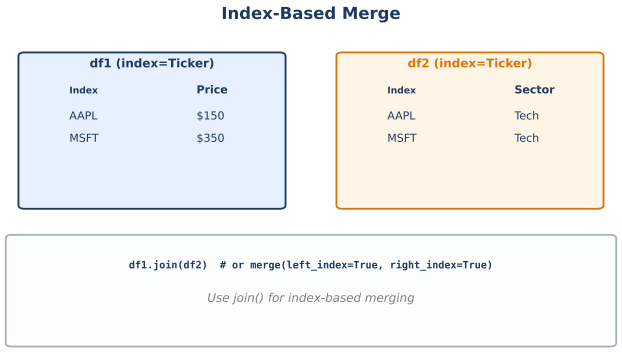
Result columns:

Ticker, Date, Price, Volume,
EPS, Sector, Industry

Chain `.merge()` calls to build a complete analytical dataset from separate sources.

Real financial analysis always combines 2–5 data sources via merge

Joining on Index



df1.join(df2) or pd.merge(left_index=True, right_index=True)

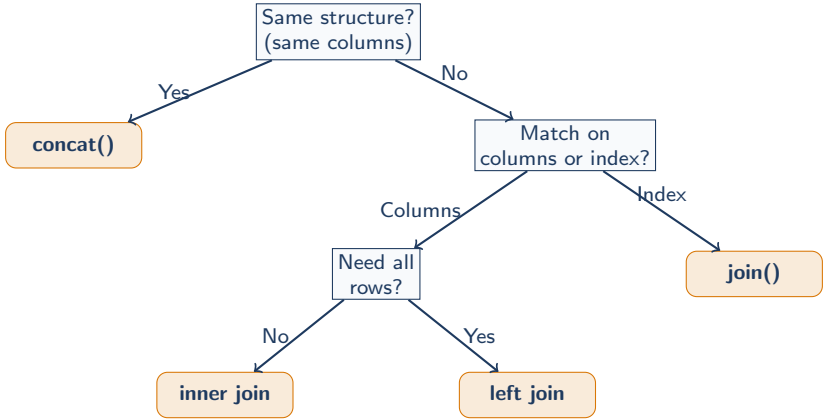
Validate Your Merges

Pre-Merge Checklist:

1. Check dtypes match: `df1['key'].dtype == df2['key'].dtype`
2. Check for duplicates: `df['key'].duplicated().sum()`
3. Check key overlap: `set(df1['key']) & set(df2['key'])`
4. Use `indicator=True` to see where rows came from

Always validate before and after merging – silent data loss is the worst bug

Which Join Should I Use?



Start with the question: same structure or different keys?

Hands-on Exercise (25 min)

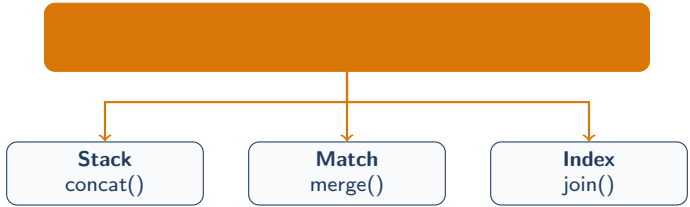
Build a multi-source stock analysis:

1. Create `df_prices` with Ticker, Date, Close for 5 stocks
2. Create `df_sectors` with Ticker, Sector for 4 of those stocks
3. Inner merge on Ticker – how many rows remain?
4. Left merge on Ticker – do you see NaN values? Where?
5. Outer merge – what happens to unmatched rows?
6. Chain two merges to combine prices + sectors + fundamentals
7. Use `indicator=True` and examine the `_merge` column

Bonus: Use `validate='one_to_many'` to catch unexpected duplicates.

Merging is a daily task in any data analytics role

The Big Idea



No single table has everything. Merging connects the pieces.

Data analysis = loading + cleaning + merging + grouping + visualizing

Key Takeaways

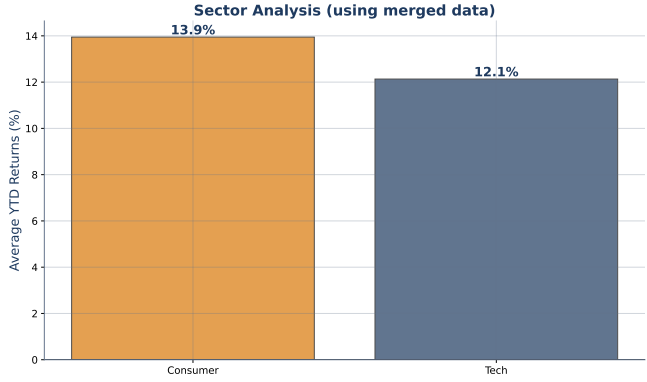
What you learned today:

- `pd.concat()` stacks DataFrames vertically (axis=0) or horizontally (axis=1)
- `pd.merge()` performs SQL-style joins on shared key columns
- Four join types: inner (intersection), outer (union), left, right
- Multiple keys with `on=['col1', 'col2']` for composite matching
- Always validate merges: check dtypes, duplicates, and key overlap

Next Lesson: L11 – NumPy Basics
pandas is built on top of NumPy. Next you learn the engine under the hood – fast array math that powers everything.

Merging is the glue that connects separate data sources into analysis-ready datasets

Finance: Sector Returns After Merge



Merge enables sector-level aggregation impossible from a single data source