جامعة بيرزيت
BIRZEIT UNIVERSITY

Data Science and Analytics
Comp 4381

Combining Datasets with Pandas

References

- **Books**:
  - Python for Data Analysis 3rd edition - Wes McKinney – O'RIELLY (Ch 2-10)
  - Python data science handbook 2nd edition - Jake VanderPlas – O'RIELLY (Ch 37-40)
  - Statistics unplugged 4th edition – Sally Cardwell - Wadsworth: (Ch 1, 2)
- **Material & Notebooks**:
  - Mr. Hussein Soboh.
- **Additional Resources:**
  - Computational and Inferential Thinking: The Foundations of Data Science 2nd Edition by Ani Adhikari, John DeNero, David Wagner. Link
  - https://www.w3schools.com/python

Merging/Joining Tables

Data Merging in Analysis

- **Why merge tables?**
  - Data often comes from multiple sources
  - Need to build a complete dataset for analysis
  - Merging brings related info into one place
- **What You'll Learn**
  - Merging tables using Pandas (Python)
  - Types of joins: inner, outer, left, right
  - Choosing the right merge strategy
  - Handling missing data
  - Managing overlapping columns
- **Importance of Merging**
  - Critical step in data preparation
  - Ensures consistency and completeness
  - Helps uncover deeper insights from combined data

Types of Joins in Pandas

- **Inner Join**
  Returns only matching records in both tables
  (Common for focused analysis with shared keys)

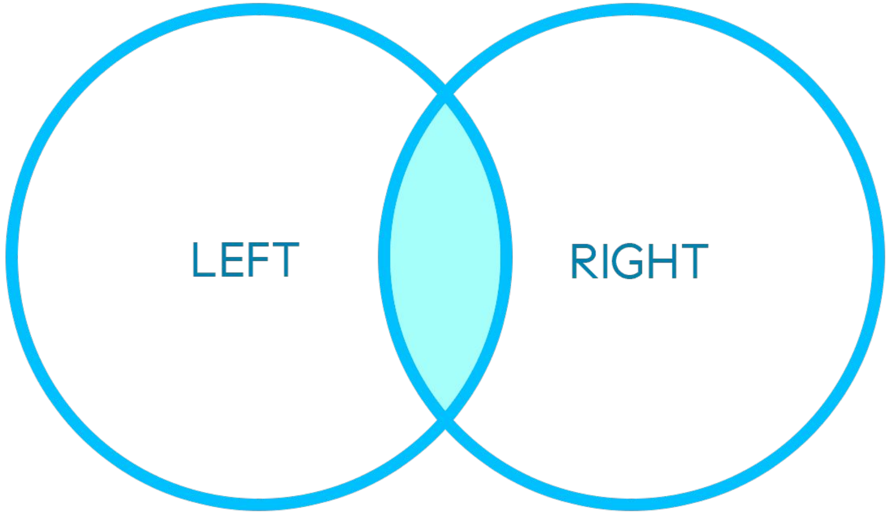
- **Left Join**
  All records from the left table + matches from the right
  (Preserve full left dataset, fill in where possible)

- **Right Join**
  All records from the right table + matches from the left
  (Useful when right table is the primary one)

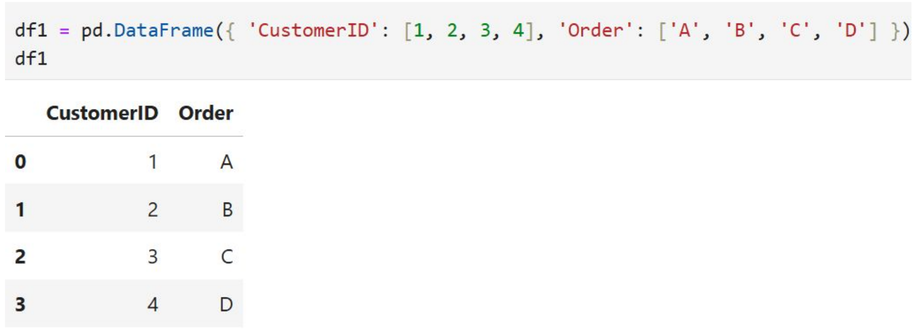
- **Outer Join**
  Combines all records and fills missing with NaN
  (For full picture, even unmatched data)

Inner Join

- An inner join returns only the rows where there is a **match in both tables**. If a record appears in only one of the tables, it won't be included in the final output. This join type is useful when you only want the common records between two datasets.

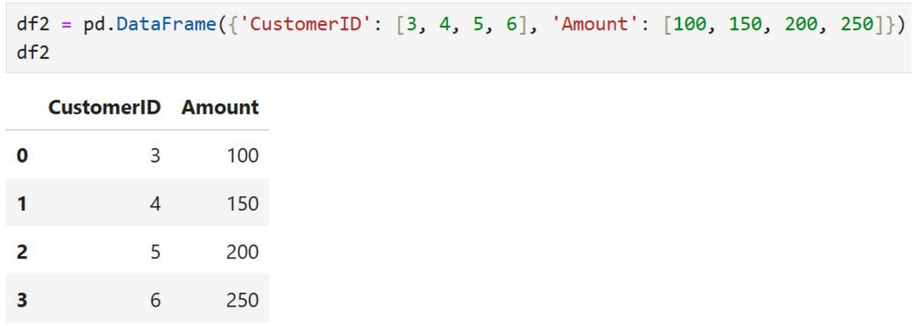


Inner Join Example



```python
df1 = pd.DataFrame({ 'CustomerID': [1, 2, 3, 4], 'Order': ['A', 'B', 'C', 'D'] })
df1
```

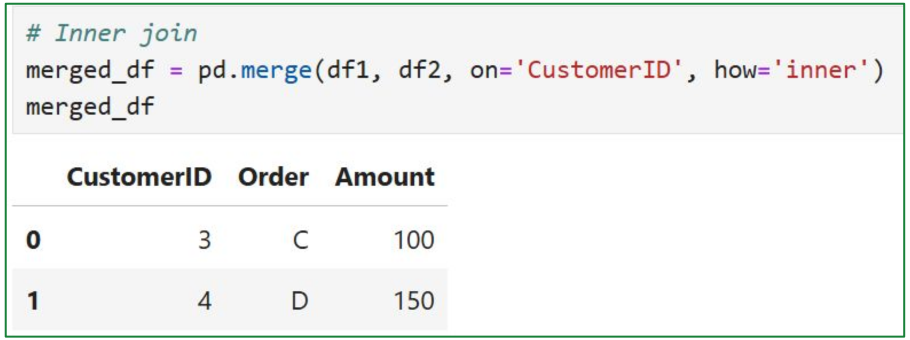
	CustomerID	Order
0	1	A
1	2	B
2	3	C
3	4	D



```python
df2 = pd.DataFrame({'CustomerID': [3, 4, 5, 6], 'Amount': [100, 150, 200, 250]})
df2
```

	CustomerID	Amount
0	3	100
1	4	150
2	5	200
3	6	250

Merges the `CustomerID` column, returning only the rows where `CustomerID` is common to both `df1` and `df2`.

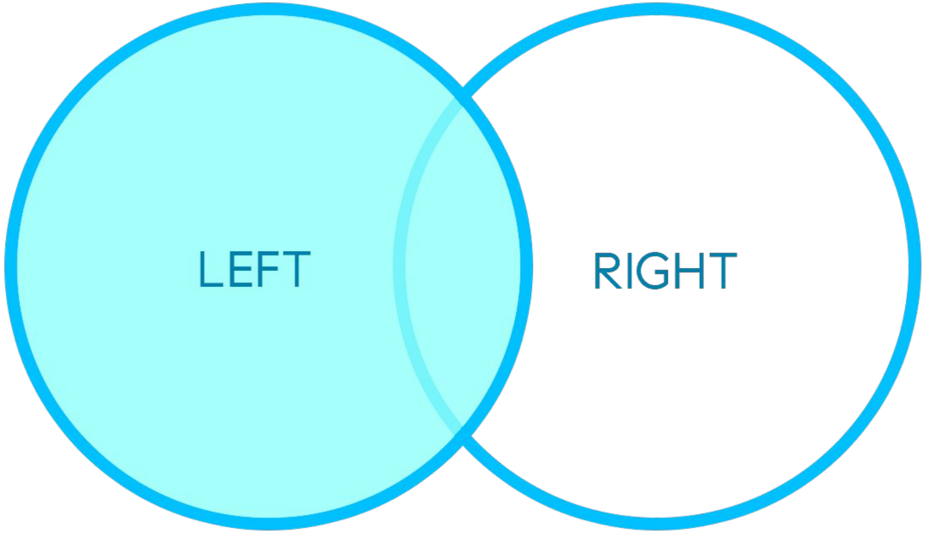


```python
# Inner join
merged_df = pd.merge(df1, df2, on='CustomerID', how='inner')
merged_df
```

	CustomerID	Order	Amount
0	3	C	100
1	4	D	150

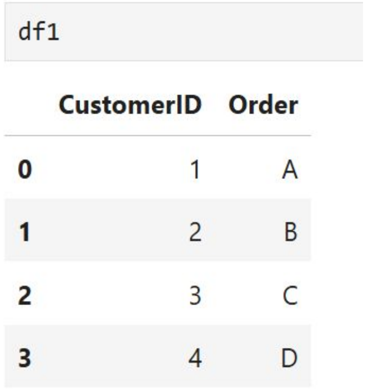
Left Join

- A left join keeps all rows from the left table and matches rows from the right table where possible. If there's no matching row in the right table, **NaN** is inserted in the resulting columns.



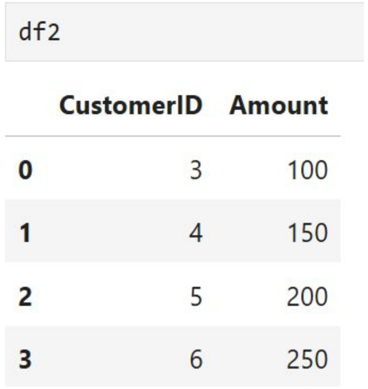
Left Join Example

- The left join keeps all **CustomerIDs** from **df1**, filling in **NaN** for those without a match in **df2**.



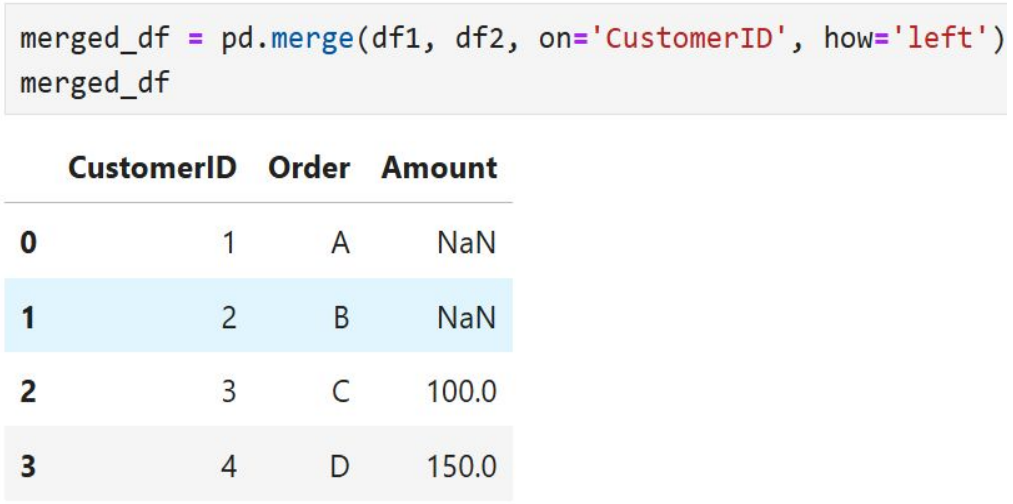
```
df1
```

	CustomerID	Order
0	1	A
1	2	B
2	3	C
3	4	D



```
df2
```

	CustomerID	Amount
0	3	100
1	4	150
2	5	200
3	6	250

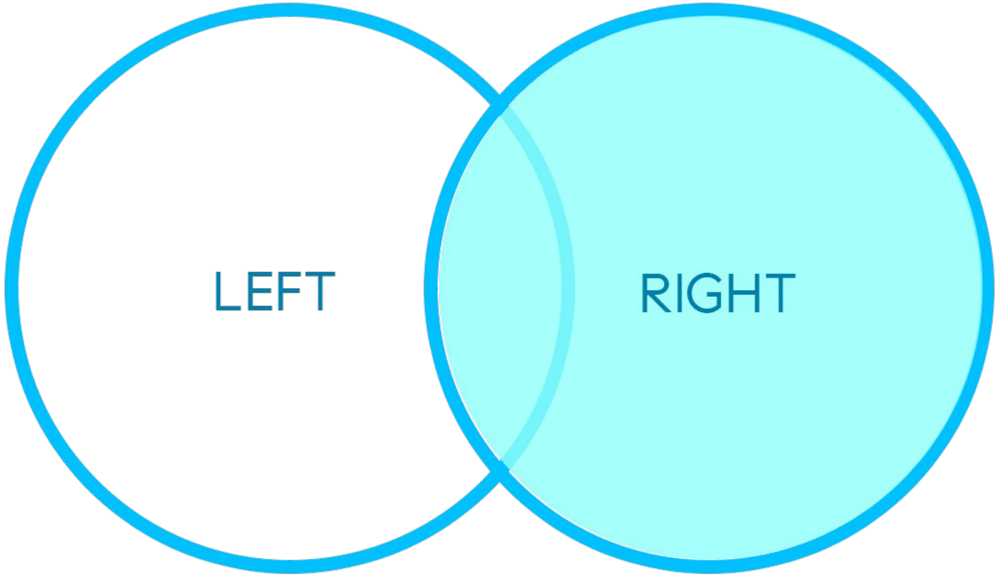


```
merged_df = pd.merge(df1, df2, on='CustomerID', how='left')
merged_df
```

	CustomerID	Order	Amount
0	1	A	NaN
1	2	B	NaN
2	3	C	100.0
3	4	D	150.0

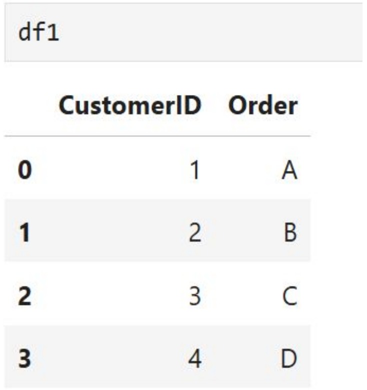
Right Join

- The right join is the opposite of the left join, retaining all rows from the right table and matching rows from the left where possible.



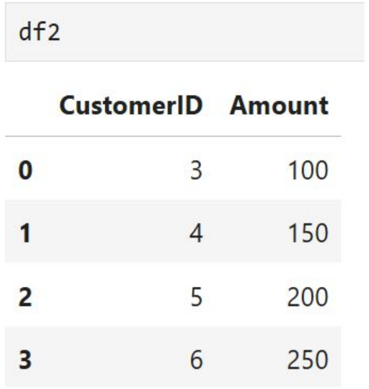
Right Join Example

- This result keeps all rows from **df2**, filling in **NaN** for rows in **df1** without a match



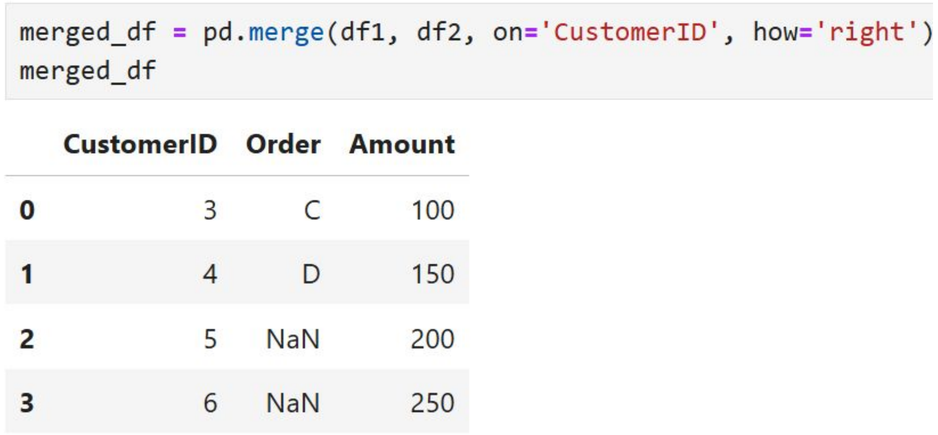
```
df1
```

	CustomerID	Order
0	1	A
1	2	B
2	3	C
3	4	D



```
df2
```

	CustomerID	Amount
0	3	100
1	4	150
2	5	200
3	6	250

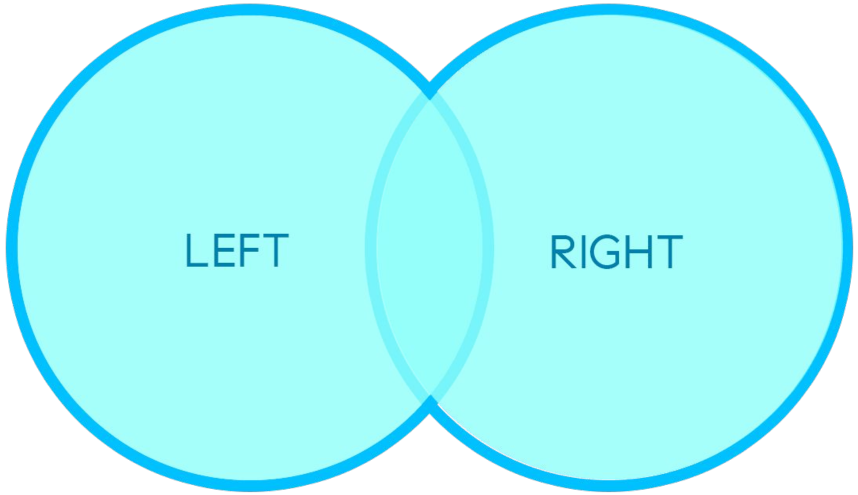


```
merged_df = pd.merge(df1, df2, on='CustomerID', how='right')
merged_df
```

	CustomerID	Order	Amount
0	3	C	100
1	4	D	150
2	5	NaN	200
3	6	NaN	250

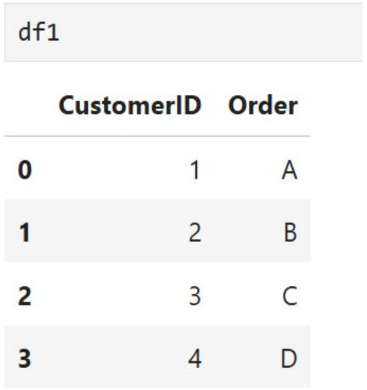
Outer Join

- An outer join includes all rows from both tables, filling **NaN** where there are missing matches in either table. This join is helpful when you want a complete view of both datasets, regardless of whether every row has a match



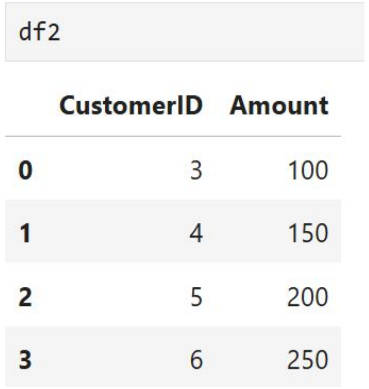
Outer Join Example

- With the outer join, you get all rows from both tables, with **NaN** values where matches were not found



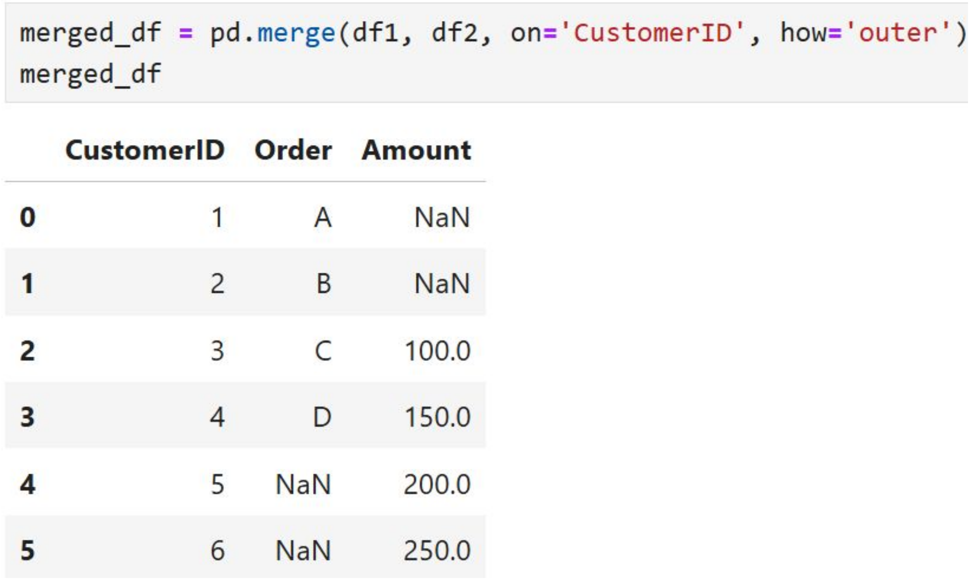
```
df1
```

	CustomerID	Order
0	1	A
1	2	B
2	3	C
3	4	D



```
df2
```

	CustomerID	Amount
0	3	100
1	4	150
2	5	200
3	6	250



```
merged_df = pd.merge(df1, df2, on='CustomerID', how='outer')
merged_df
```

	CustomerID	Order	Amount
0	1	A	NaN
1	2	B	NaN
2	3	C	100.0
3	4	D	150.0
4	5	NaN	200.0
5	6	NaN	250.0

Indicator Parameter

- When pass True to indicator

```
merged_df = pd.merge(df2, df1, on='CustomerID', how='outer', indicator=True)
merged_df
```

	CustomerID	Amount	Order	_merge
0	1	NaN	A	right_only
1	2	NaN	B	right_only
2	3	100.0	C	both
3	4	150.0	D	both
4	5	200.0	NaN	left_only
5	6	250.0	NaN	left_only

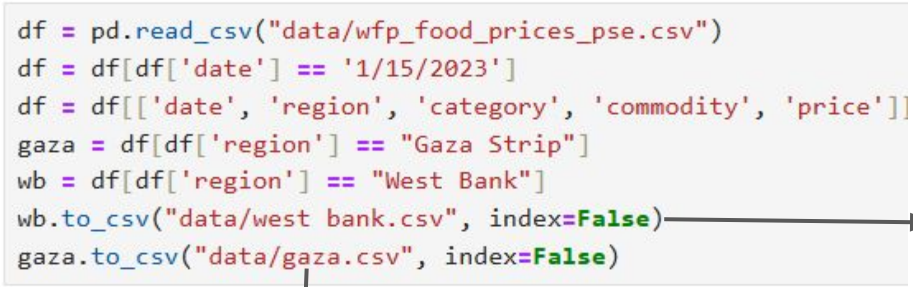
Summary

- **Inner join:** Use when you need only records that have matches in both tables.
- **Left join:** Use when you want to keep all records from the first (left) table and match where possible in the second (right) table.
- **Right join:** Use when you want to keep all records from the second (right) table and match where possible in the first (left) table.
- **Outer join:** Use when you want a complete record of all rows, regardless of whether they have matches in both tables.

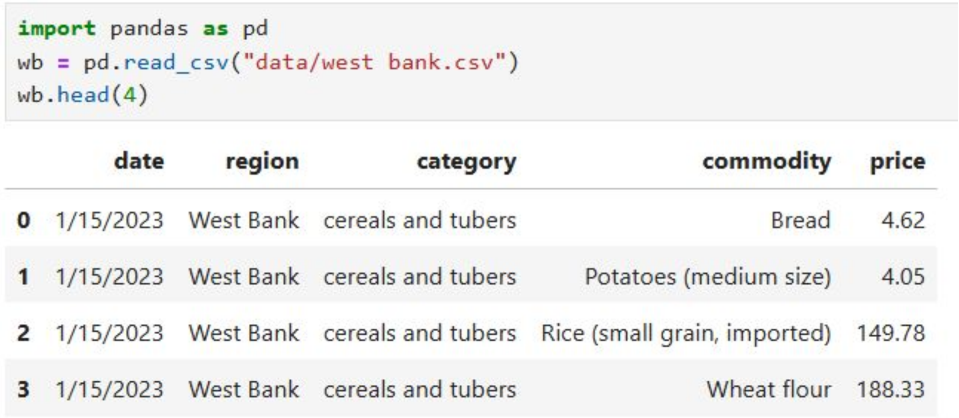
Practical Example

- We have two files: **"west bank.csv"**, which lists food prices of all items available in west bank for January 2023, and **"gaza.csv"**, which also lists food prices of all items available in Gaza for January 2023. By merging these files, we'll be able to compare the prices between the two regions for insightful analysis.

Practical Example

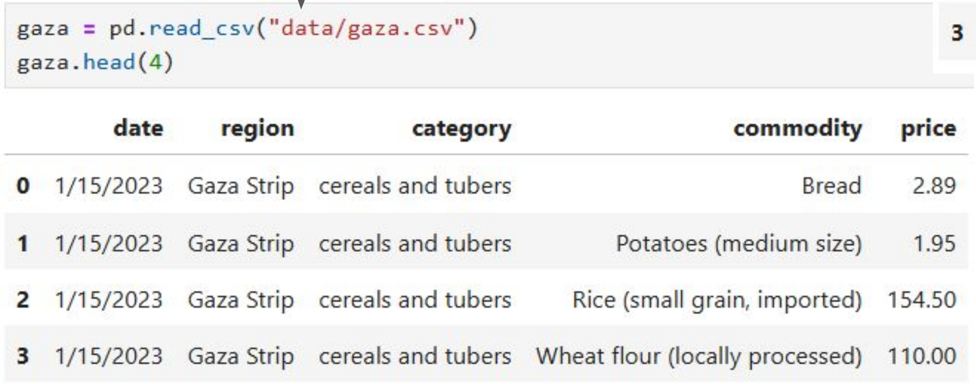


```python
df = pd.read_csv("data/wfp_food_prices_pse.csv")
df = df[df['date'] == '1/15/2023']
df = df[['date', 'region', 'category', 'commodity', 'price']]
gaza = df[df['region'] == "Gaza Strip"]
wb = df[df['region'] == "West Bank"]
wb.to_csv("data/west bank.csv", index=False)
gaza.to_csv("data/gaza.csv", index=False)
```



```python
import pandas as pd
wb = pd.read_csv("data/west bank.csv")
wb.head(4)
```

	date	region	category	commodity	price
0	1/15/2023	West Bank	cereals and tubers	Bread	4.62
1	1/15/2023	West Bank	cereals and tubers	Potatoes (medium size)	4.05
2	1/15/2023	West Bank	cereals and tubers	Rice (small grain, imported)	149.78
3	1/15/2023	West Bank	cereals and tubers	Wheat flour	188.33

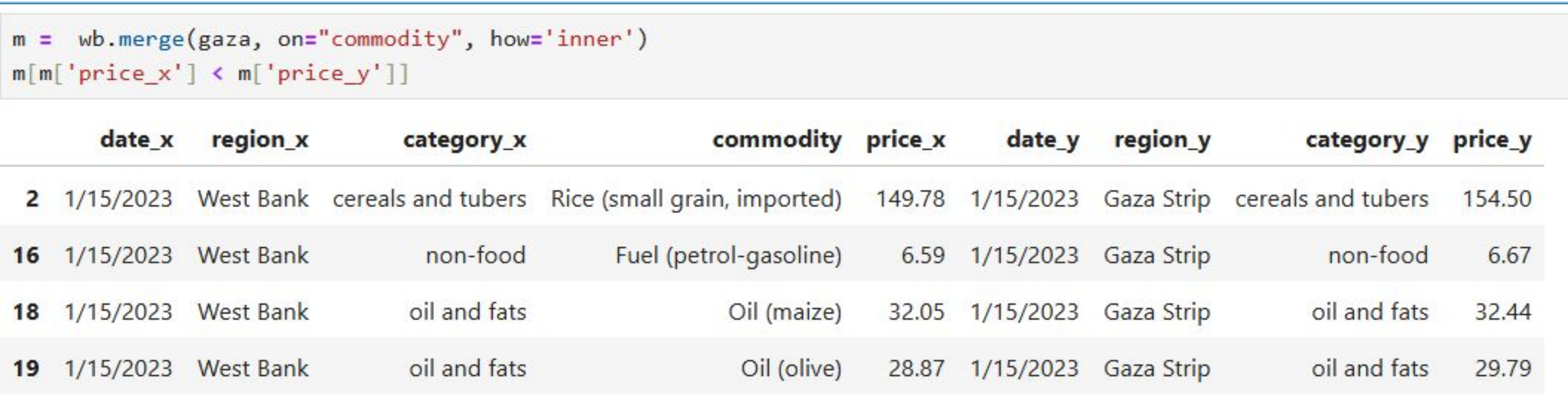


```python
gaza = pd.read_csv("data/gaza.csv")
gaza.head(4)
```

	date	region	category	commodity	price
0	1/15/2023	Gaza Strip	cereals and tubers	Bread	2.89
1	1/15/2023	Gaza Strip	cereals and tubers	Potatoes (medium size)	1.95
2	1/15/2023	Gaza Strip	cereals and tubers	Rice (small grain, imported)	154.50
3	1/15/2023	Gaza Strip	cereals and tubers	Wheat flour (locally processed)	110.00

Practical Example

- Identify commodities that are priced lower in the West Bank compared to Gaza, and vice versa

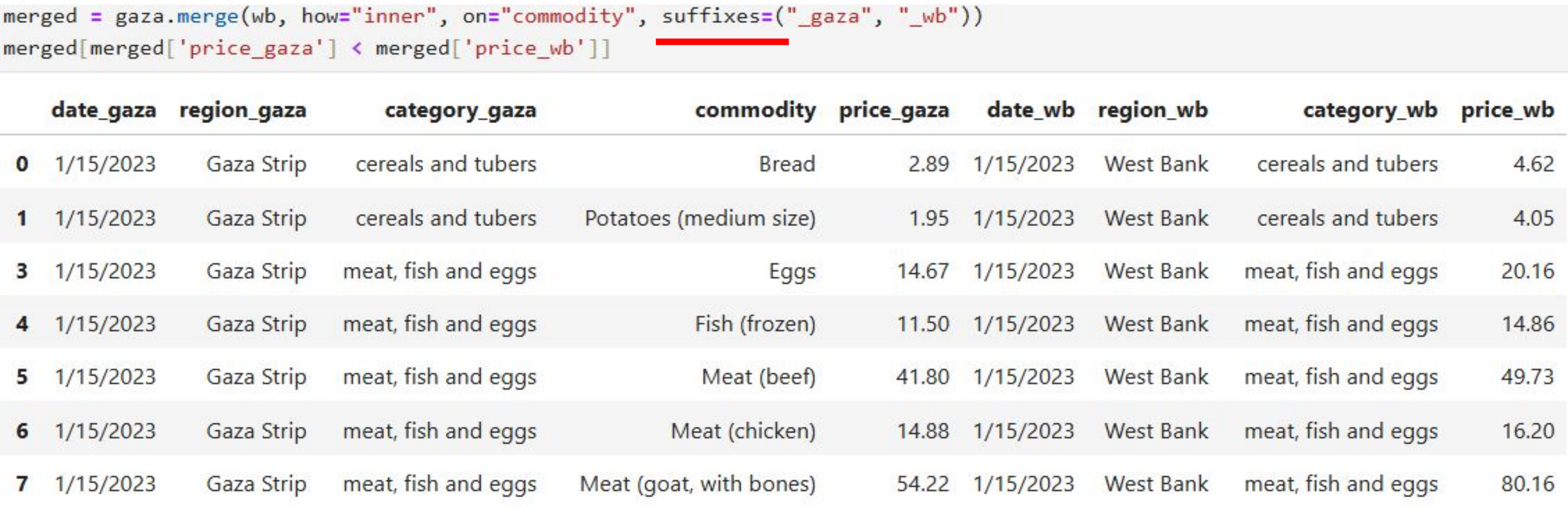


```
m =  wb.merge(gaza, on="commodity", how='inner')
m[m['price_x'] < m['price_y']]
```

	date_x	region_x	category_x	commodity	price_x	date_y	region_y	category_y	price_y
2	1/15/2023	West Bank	cereals and tubers	Rice (small grain, imported)	149.78	1/15/2023	Gaza Strip	cereals and tubers	154.50
16	1/15/2023	West Bank	non-food	Fuel (petrol-gasoline)	6.59	1/15/2023	Gaza Strip	non-food	6.67
18	1/15/2023	West Bank	oil and fats	Oil (maize)	32.05	1/15/2023	Gaza Strip	oil and fats	32.44
19	1/15/2023	West Bank	oil and fats	Oil (olive)	28.87	1/15/2023	Gaza Strip	oil and fats	29.79

Practical Example

- Identify commodities that are priced higher in the West Bank compared to Gaza, and vice versa

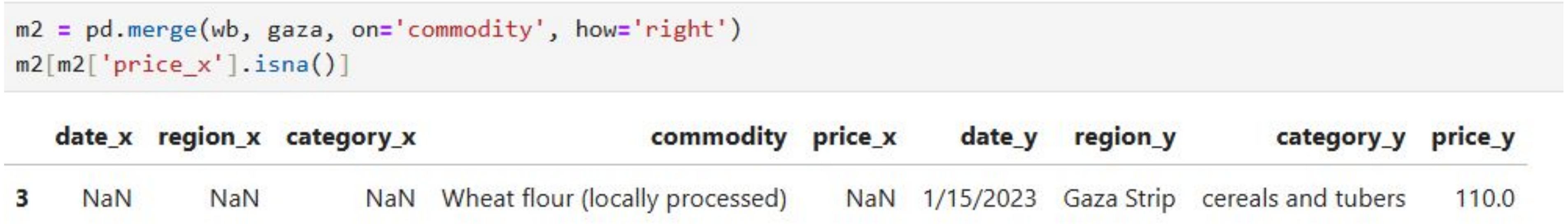


```
merged = gaza.merge(wb, how="inner", on="commodity", suffixes=("_gaza", "_wb"))
merged[merged['price_gaza'] < merged['price_wb']]
```

	date_gaza	region_gaza	category_gaza	commodity	price_gaza	date_wb	region_wb	category_wb	price_wb
0	1/15/2023	Gaza Strip	cereals and tubers	Bread	2.89	1/15/2023	West Bank	cereals and tubers	4.62
1	1/15/2023	Gaza Strip	cereals and tubers	Potatoes (medium size)	1.95	1/15/2023	West Bank	cereals and tubers	4.05
3	1/15/2023	Gaza Strip	meat, fish and eggs	Eggs	14.67	1/15/2023	West Bank	meat, fish and eggs	20.16
4	1/15/2023	Gaza Strip	meat, fish and eggs	Fish (frozen)	11.50	1/15/2023	West Bank	meat, fish and eggs	14.86
5	1/15/2023	Gaza Strip	meat, fish and eggs	Meat (beef)	41.80	1/15/2023	West Bank	meat, fish and eggs	49.73
6	1/15/2023	Gaza Strip	meat, fish and eggs	Meat (chicken)	14.88	1/15/2023	West Bank	meat, fish and eggs	16.20
7	1/15/2023	Gaza Strip	meat, fish and eggs	Meat (goat, with bones)	54.22	1/15/2023	West Bank	meat, fish and eggs	80.16

Practical Example

- Identify commodities that are available for sale in the Gaza but not in West Bank.

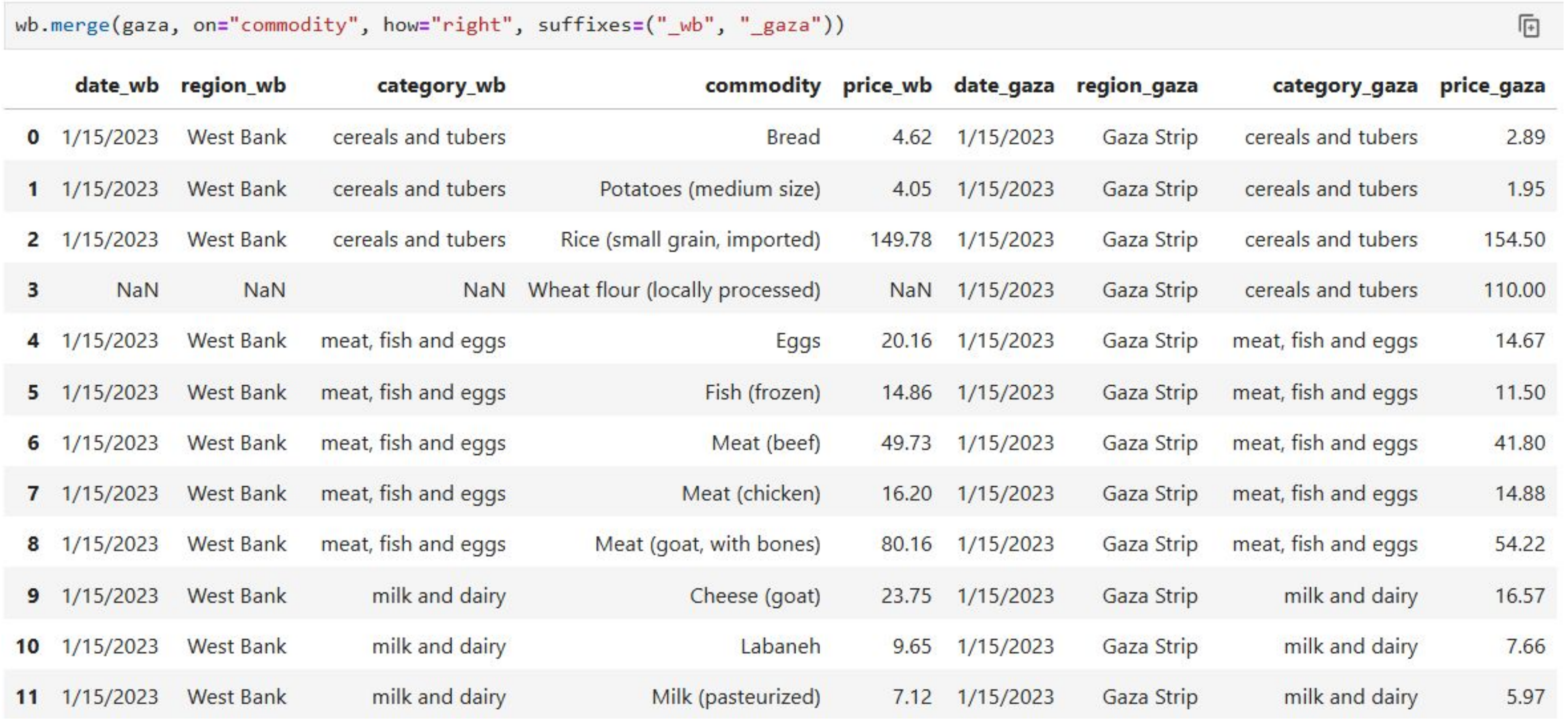


```python
m2 = pd.merge(wb, gaza, on='commodity', how='right')
m2[m2['price_x'].isna()]
```

	date_x	region_x	category_x	commodity	price_x	date_y	region_y	category_y	price_y
3	NaN	NaN	NaN	Wheat flour (locally processed)	NaN	1/15/2023	Gaza Strip	cereals and tubers	110.0

Practical Example

- Identify commodities that are NOT available for sale in the West Bank but available in Gaza

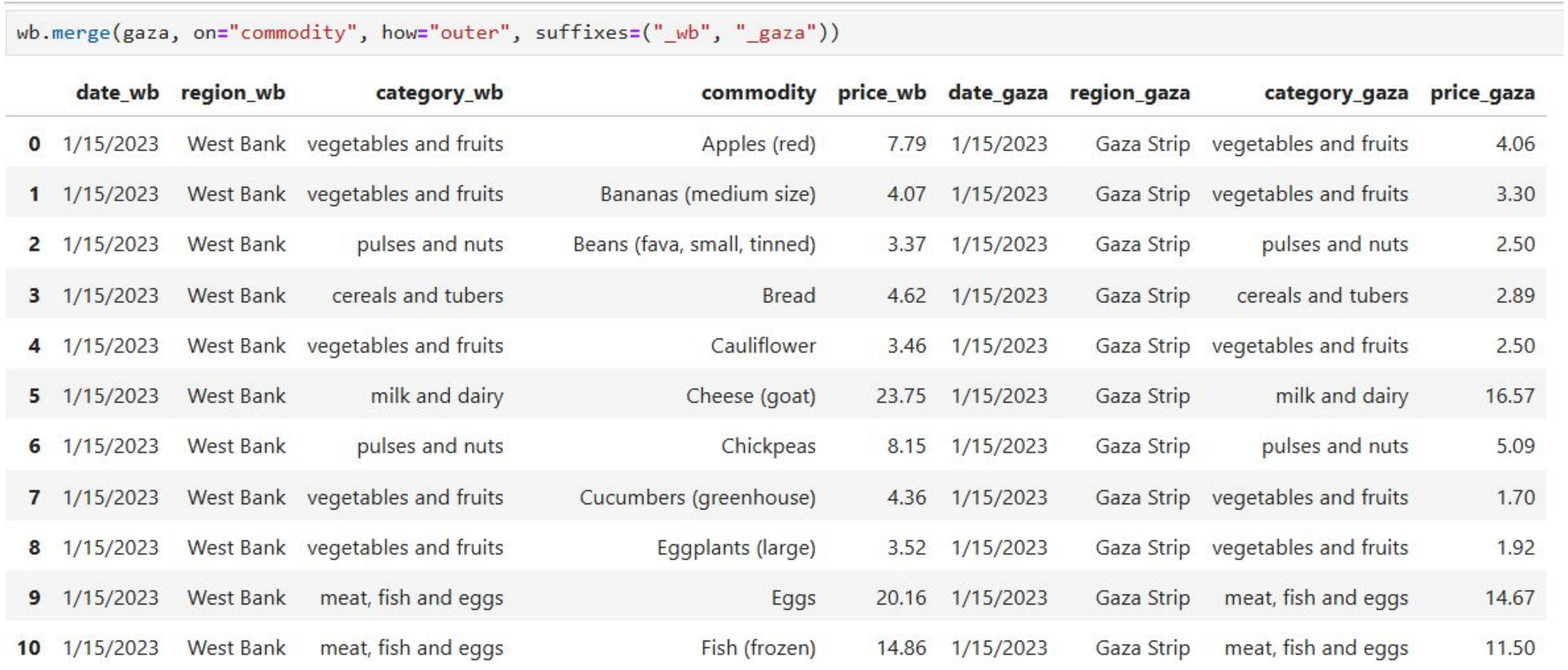


```
wb.merge(gaza, on="commodity", how="right", suffixes=("_wb", "_gaza"))
```

	date_wb	region_wb	category_wb	commodity	price_wb	date_gaza	region_gaza	category_gaza	price_gaza
0	1/15/2023	West Bank	cereals and tubers	Bread	4.62	1/15/2023	Gaza Strip	cereals and tubers	2.89
1	1/15/2023	West Bank	cereals and tubers	Potatoes (medium size)	4.05	1/15/2023	Gaza Strip	cereals and tubers	1.95
2	1/15/2023	West Bank	cereals and tubers	Rice (small grain, imported)	149.78	1/15/2023	Gaza Strip	cereals and tubers	154.50
3	NaN	NaN	NaN	Wheat flour (locally processed)	NaN	1/15/2023	Gaza Strip	cereals and tubers	110.00
4	1/15/2023	West Bank	meat, fish and eggs	Eggs	20.16	1/15/2023	Gaza Strip	meat, fish and eggs	14.67
5	1/15/2023	West Bank	meat, fish and eggs	Fish (frozen)	14.86	1/15/2023	Gaza Strip	meat, fish and eggs	11.50
6	1/15/2023	West Bank	meat, fish and eggs	Meat (beef)	49.73	1/15/2023	Gaza Strip	meat, fish and eggs	41.80
7	1/15/2023	West Bank	meat, fish and eggs	Meat (chicken)	16.20	1/15/2023	Gaza Strip	meat, fish and eggs	14.88
8	1/15/2023	West Bank	meat, fish and eggs	Meat (goat, with bones)	80.16	1/15/2023	Gaza Strip	meat, fish and eggs	54.22
9	1/15/2023	West Bank	milk and dairy	Cheese (goat)	23.75	1/15/2023	Gaza Strip	milk and dairy	16.57
10	1/15/2023	West Bank	milk and dairy	Labaneh	9.65	1/15/2023	Gaza Strip	milk and dairy	7.66
11	1/15/2023	West Bank	milk and dairy	Milk (pasteurized)	7.12	1/15/2023	Gaza Strip	milk and dairy	5.97

Practical Example

- Compile a list of all food prices available in both Gaza and the West Bank.



```
wb.merge(gaza, on="commodity", how="outer", suffixes=("_wb", "_gaza"))
```

	date_wb	region_wb	category_wb	commodity	price_wb	date_gaza	region_gaza	category_gaza	price_gaza
0	1/15/2023	West Bank	vegetables and fruits	Apples (red)	7.79	1/15/2023	Gaza Strip	vegetables and fruits	4.06
1	1/15/2023	West Bank	vegetables and fruits	Bananas (medium size)	4.07	1/15/2023	Gaza Strip	vegetables and fruits	3.30
2	1/15/2023	West Bank	pulses and nuts	Beans (fava, small, tinned)	3.37	1/15/2023	Gaza Strip	pulses and nuts	2.50
3	1/15/2023	West Bank	cereals and tubers	Bread	4.62	1/15/2023	Gaza Strip	cereals and tubers	2.89
4	1/15/2023	West Bank	vegetables and fruits	Cauliflower	3.46	1/15/2023	Gaza Strip	vegetables and fruits	2.50
5	1/15/2023	West Bank	milk and dairy	Cheese (goat)	23.75	1/15/2023	Gaza Strip	milk and dairy	16.57
6	1/15/2023	West Bank	pulses and nuts	Chickpeas	8.15	1/15/2023	Gaza Strip	pulses and nuts	5.09
7	1/15/2023	West Bank	vegetables and fruits	Cucumbers (greenhouse)	4.36	1/15/2023	Gaza Strip	vegetables and fruits	1.70
8	1/15/2023	West Bank	vegetables and fruits	Eggplants (large)	3.52	1/15/2023	Gaza Strip	vegetables and fruits	1.92
9	1/15/2023	West Bank	meat, fish and eggs	Eggs	20.16	1/15/2023	Gaza Strip	meat, fish and eggs	14.67
10	1/15/2023	West Bank	meat, fish and eggs	Fish (frozen)	14.86	1/15/2023	Gaza Strip	meat, fish and eggs	11.50

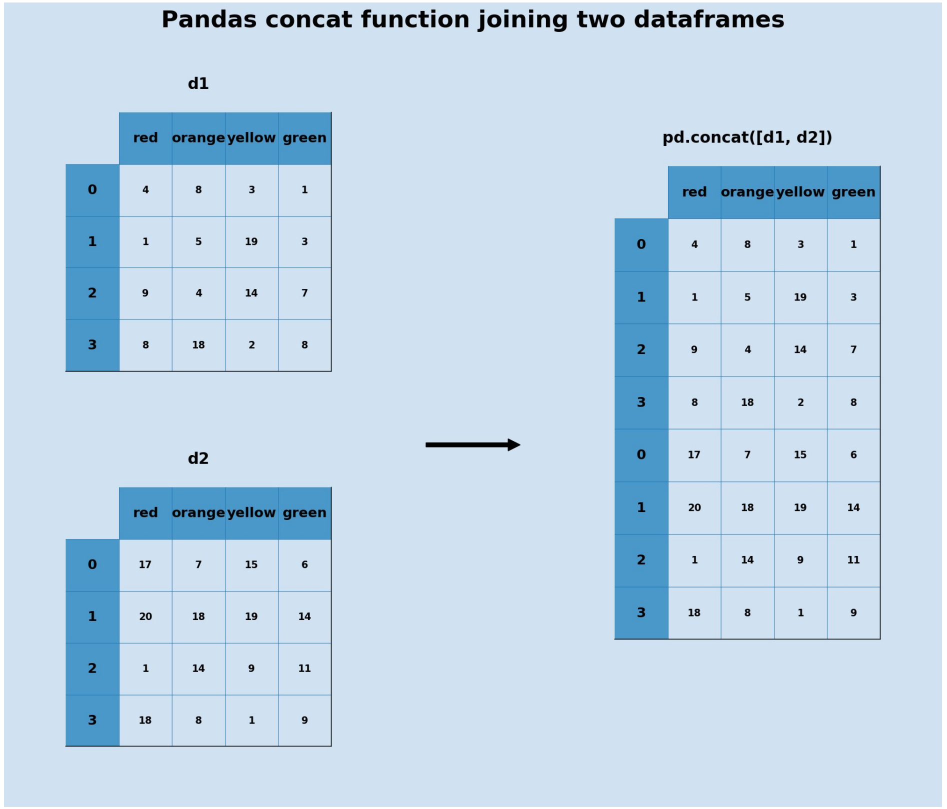
Concatenating dataframes

Concatenate Date

- The **concat** function in Pandas allows you to combine data from multiple DataFrames along either **rows (vertical)** or **columns (horizontal)**.
- When you merge data vertically, you’re stacking rows from one DataFrame on top of another
- Which is often useful when you have similar data split across different files or subsets and want to combine them into a single DataFrame for further analysis
- Use `axis=0` for vertical (row-wise) concatenation.
- Use `axis=1` for horizontal (column-wise) concatenation.

Concatenate Date Vertically

- All DataFrames **should ideally have the same columns**, as concat will align data based on column names. If a column is missing in any DataFrame, `NaN` values will fill those cells.
- By default, concat will **keep the indices from each DataFrame**, though you can reset or ignore the index.
- To concatenate data vertically, use `axis=0` as parameter to `concat` function



Pandas concat function joining two dataframes

d1

	red	orange	yellow	green
0	4	8	3	1
1	1	5	19	3
2	9	4	14	7
3	8	18	2	8

d2

	red	orange	yellow	green
0	17	7	15	6
1	20	18	19	14
2	1	14	9	11
3	18	8	1	9

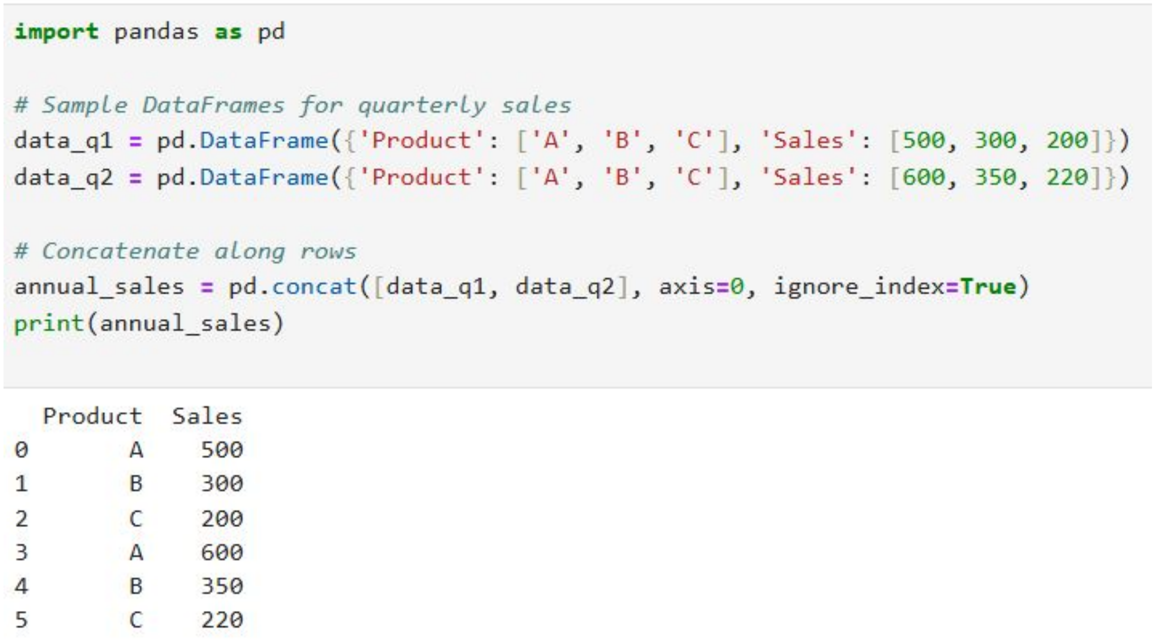
pd.concat([d1, d2])

	red	orange	yellow	green
0	4	8	3	1
1	1	5	19	3
2	9	4	14	7
3	8	18	2	8
0	17	7	15	6
1	20	18	19	14
2	1	14	9	11
3	18	8	1	9

Concatenate Date Vertically Example

- Suppose we have quarterly sales data for the same year stored in separate DataFrames:

This combines **`data_q1`** and **`data_q2`** vertically into a single DataFrame, making it easier to analyze total sales across the year. The **`ignore_index=True`** parameter resets the index to create a continuous sequence in the merged DataFrame

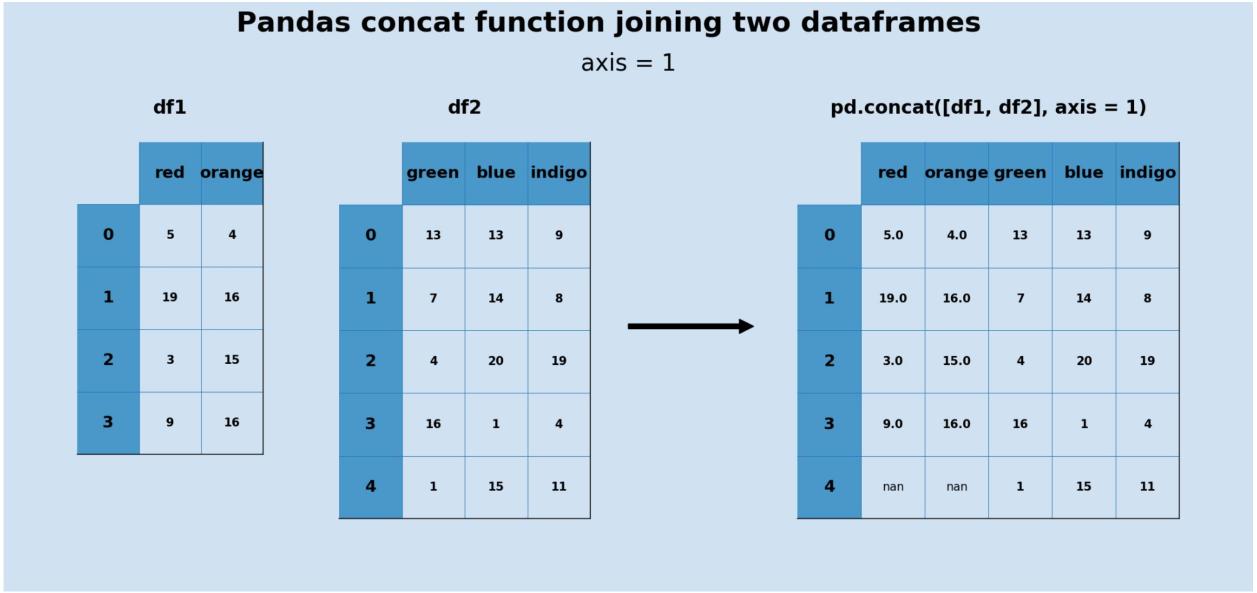


```
import pandas as pd

# Sample DataFrames for quarterly sales
data_q1 = pd.DataFrame({'Product': ['A', 'B', 'C'], 'Sales': [500, 300, 200]})
data_q2 = pd.DataFrame({'Product': ['A', 'B', 'C'], 'Sales': [600, 350, 220]})

# Concatenate along rows
annual_sales = pd.concat([data_q1, data_q2], axis=0, ignore_index=True)
print(annual_sales)
```

```
  Product  Sales
0       A    500
1       B    300
2       C    200
3       A    600
4       B    350
5       C    220
```

Concatenate Date Horizontally

- Merge data horizontally by combining the DataFrames side-by-side, adding columns rather than rows
- This approach is useful when you have related data split across DataFrames with a shared index or key, and you want to bring in additional columns



Pandas concat function joining two dataframes

axis = 1

df1

	red	orange
0	5	4
1	19	16
2	3	15
3	9	16

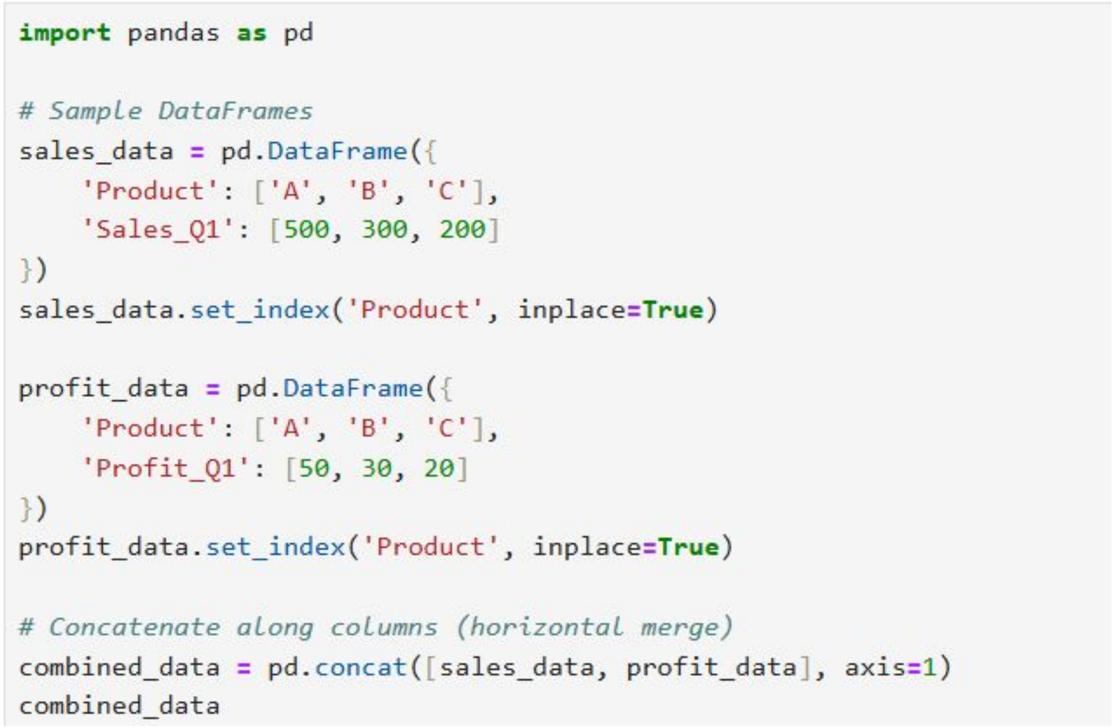
df2

	green	blue	indigo
0	13	13	9
1	7	14	8
2	4	20	19
3	16	1	4
4	1	15	11

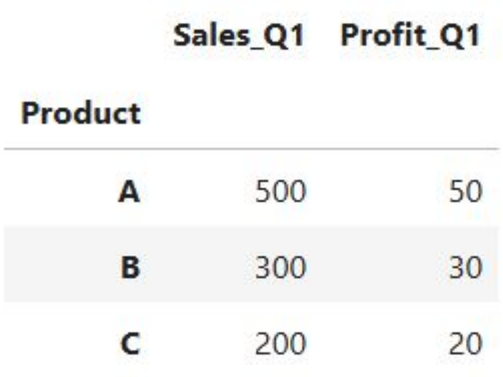
pd.concat([df1, df2], axis = 1)

	red	orange	green	blue	indigo
0	5.0	4.0	13	13	9
1	19.0	16.0	7	14	8
2	3.0	15.0	4	20	19
3	9.0	16.0	16	1	4
4	nan	nan	1	15	11

Concatenate Date Horizontally Example



```
import pandas as pd

# Sample DataFrames
sales_data = pd.DataFrame({
    'Product': ['A', 'B', 'C'],
    'Sales_Q1': [500, 300, 200]
})
sales_data.set_index('Product', inplace=True)

profit_data = pd.DataFrame({
    'Product': ['A', 'B', 'C'],
    'Profit_Q1': [50, 30, 20]
})
profit_data.set_index('Product', inplace=True)

# Concatenate along columns (horizontal merge)
combined_data = pd.concat([sales_data, profit_data], axis=1)
combined_data
```



	Sales_Q1	Profit_Q1
Product		
A	500	50
B	300	30
C	200	20

To consider

- **When the index is different:** By default, concat aligns rows based on the index. If indices don't match, `NaN` values will fill in for missing entries.
- **When the rows are different**: If one DataFrame has rows that the other doesn't, the unmatched rows will have `NaN` values for the missing columns.
- **Duplicate columns**: When combining on `axis=1`, avoid duplicate column names, as concat will append `_x` and `_y` suffixes to duplicate names.

End Of Slides

Thank you.