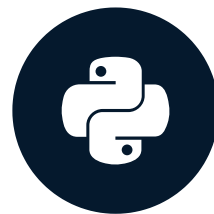


Using `merge_ordered()`

JOINING DATA WITH PANDAS



Aaren Stubberfield
Instructor

merge_ordered()

Left Table

A	B	C
A3	B3	C3
A2	B2	C2
A1	B1	C1



Right Table

C	D
C4	D4
C2	D2
C1	D1

=

Result Table

A	B	C	D
A1	B1	C1	D1
A2	B2	C2	D2
A3	B3	C3	
		C4	D4

Sorted

Method comparison

`.merge()` method:

- Column(s) to join on
 - `on`, `left_on`, and `right_on`
- Type of join
 - `how` (*left, right, inner, outer*) `{{@}}`
 - **default** inner
- Overlapping column names
 - suffixes
- Calling the method
 - `df1.merge(df2)`

`merge_ordered()` method:

- Column(s) to join on
 - `on`, `left_on`, and `right_on`
- Type of join
 - `how` (*left, right, inner, outer*)
 - **default** outer
- Overlapping column names
 - suffixes
- Calling the function
 - `pd.merge_ordered(df1, df2)`

Financial dataset



¹ Photo by Markus Spiske on Unsplash

Stock data

Table Name: `appl`

	date	<u>close</u>
0	2007-02-01	12.087143
1	2007-03-01	13.272857
2	2007-04-01	14.257143
3	2007-05-01	17.312857
4	2007-06-01	17.434286

Mac

Table Name: `mcd`

	date	<u>close</u>
0	2007-01-01	44.349998
1	2007-02-01	43.689999
2	2007-03-01	45.049999
3	2007-04-01	48.279999
4	2007-05-01	50.549999

Merging stock data

```
import pandas as pd  
pd.merge_ordered(aapl, mcd, on='date', suffixes=('_aapl', '_mcd'))
```

	date	close_aapl	close_mcd
0	2007-01-01	NaN	44.349998
1	2007-02-01	12.087143	43.689999
2	2007-03-01	13.272857	45.049999
3	2007-04-01	14.257143	48.279999
4	2007-05-01	17.312857	50.549999
5	2007-06-01	17.434286	NaN

Conten

Forward fill

Before

A	B
A1	B1
A2	
A3	B3
A4	
A5	B5

After

A	B
A1	B1
A2	B1
A3	B3
A4	B3
A5	B5

Fills missing
with
previous
value

Common method

Forward fill example

```
pd.merge_ordered(appl, mcd, on='date',  
                 suffixes=('_aapl', '_mcd'),  
                 fill_method='ffill')
```

	date	close_aapl	close_mcd
0	2007-01-01	NaN	44.349998
1	2007-02-01	12.087143	43.689999
2	2007-03-01	13.272857	45.049999
3	2007-04-01	14.257143	48.279999
4	2007-05-01	17.312857	50.549999
5	2007-06-01	17.434286	50.549999

```
pd.merge_ordered(appl, mcd, on='date',  
                 suffixes=('_aapl', '_mcd'))
```

	date	close_AAPL	close_mcd
0	2007-01-01	NaN	44.349998
1	2007-02-01	12.087143	43.689999
2	2007-03-01	13.272857	45.049999
3	2007-04-01	14.257143	48.279999
4	2007-05-01	17.312857	50.549999
5	2007-06-01	17.434286	NaN

When to use merge_ordered()?

- Ordered data / time series
- Filling in missing values

Exercise

merge_ordered() caution, multiple columns

When using `merge_ordered()` to merge on multiple columns, the order is important when you combine it with the forward fill feature. The function sorts the merge on columns in the order provided. In this exercise, we will merge GDP and population data from the World Bank for the Australia and Sweden, using the order of the merge on columns. The frequency of the series are different, the GDP values are quarterly, and the population is yearly. Use the forward fill feature to fill in the missing values.

Nice! When you merge on `date` first, the table is sorted by `date` then `country`. When forward fill is applied, Sweden's population value in January is used to fill in the missing values for both Australia and the Sweden for the remainder of the year. This is not what you want. The fill forward is using unintended data to fill in the missing values.

However, when you merge on `country` first, the table is sorted by `country` then `date`, so the forward fill is applied appropriately in this situation.

Use `merge_ordered()` on `gdp` and `pop`, merging on columns `date` and `country` with the fill feature, save to `ctry_date`.

Perform the same merge, but join on `country` and `date` with the fill feature, saving this as `date_ctry`.

✓

+100 XP

PRESS ENTER TO

Continue

script.py

Light Mode

```
1 # Merge gdp and pop on country and date with fill
2 date_ctry = pd.merge_ordered(gdp, pop, on = ['country', 'date'],
3                               fill_method = 'ffill')
4
5 # Print date_ctry
6 print(date_ctry)
```

↶

Run Code

Submit Answer

IPython Shell

Slides

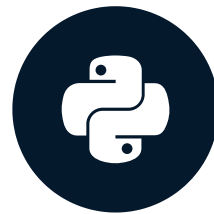
	date	country	gdp	series_code_x	pop	series_code_y
0	1990-01-01	Australia	158051.132	NYGDPMKTPSAKD	17065100	SP.POP.TOTL
1	1990-01-01	Sweden	79837.846	NYGDPMKTPSAKD	8558835	SP.POP.TOTL
2	1990-04-01	Australia	158263.582	NYGDPMKTPSAKD	8558835	SP.POP.TOTL
3	1990-04-01	Sweden	80582.286	NYGDPMKTPSAKD	8558835	SP.POP.TOTL
4	1990-07-01	Australia	157329.279	NYGDPMKTPSAKD	8558835	SP.POP.TOTL
5	1990-07-01	Sweden	79974.360	NYGDPMKTPSAKD	8558835	SP.POP.TOTL
6	1990-09-01	Australia	158240.678	NYGDPMKTPSAKD	8558835	SP.POP.TOTL

Let's practice!

JOINING DATA WITH PANDAS

Using `merge_asof()`

JOINING DATA WITH PANDAS



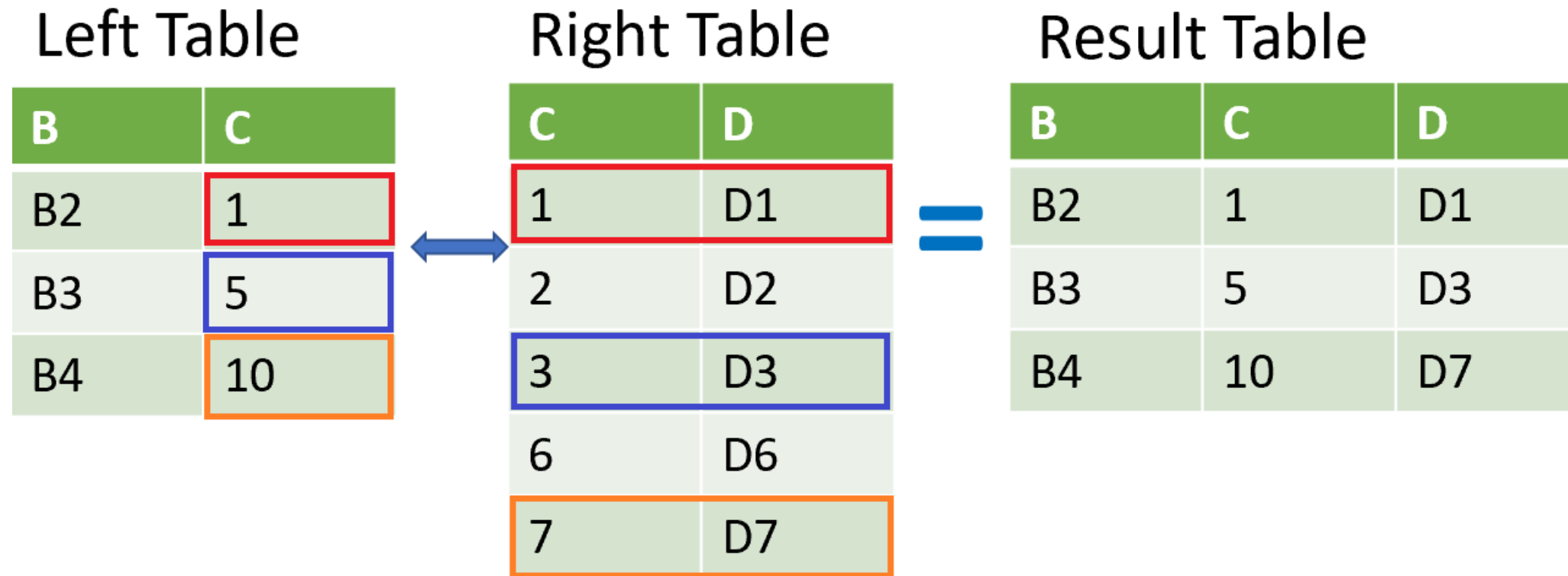
Aaren Stubberfield
Instructor

Using merge_asof()

Left Table			Right Table			Result Table		
B	C		C	D	=	B	C	D
B2	1	↔	1	D1	=	B2	1	D1
B3	5		2	D2		B3	5	D3
B4	10		3	D3		B4	10	D7
			6	D6				
			7	D7				

- Similar to a `merge_ordered()` left join
 - Similar features as `merge_ordered()`
- Match on the nearest key column and not exact matches.
 - Merged "on" columns must be sorted.

Using merge_asof()



- Similar to a `merge_ordered()` left join
 - Similar features as `merge_ordered()`
- Match on the nearest key column and not exact matches.
 - Merged "on" columns must be sorted.

Datasets

Table Name: visa

	date_time	close
0	2017-11-17 16:00:00	110.32
1	2017-11-17 17:00:00	110.24
2	2017-11-17 18:00:00	110.065
3	2017-11-17 19:00:00	110.04
4	2017-11-17 20:00:00	110.0
5	2017-11-17 21:00:00	109.9966
6	2017-11-17 22:00:00	109.82

Table Name: ibm

	date_time	close
0	2017-11-17 15:35:12	149.3
1	2017-11-17 15:40:34	149.13
2	2017-11-17 15:45:50	148.98
3	2017-11-17 15:50:20	148.99
4	2017-11-17 15:55:10	149.11
5	2017-11-17 16:00:03	149.25
6	2017-11-17 16:05:06	149.5175
7	2017-11-17 16:10:12	149.57
8	2017-11-17 16:15:30	149.59
9	2017-11-17 16:20:32	149.82
10	2017-11-17 16:25:47	149.96

merge_asof() example

```
pd.merge_asof(visa, ibm, on='date_time',  
              suffixes=('_visa', '_ibm'))
```

Table Name: `ibm`

	date_time	close_visa	close_ibm
0	2017-11-17 16:00:00	110.32	149.11
1	2017-11-17 17:00:00	110.24	149.83
2	2017-11-17 18:00:00	110.065	149.59
3	2017-11-17 19:00:00	110.04	149.505
4	2017-11-17 20:00:00	110.0	149.42
5	2017-11-17 21:00:00	109.9966	149.26
6	2017-11-17 22:00:00	109.82	148.97

	date_time	close
0	2017-11-17 15:35:12	149.3
1	2017-11-17 15:40:34	149.13
2	2017-11-17 15:45:50	148.98
3	2017-11-17 15:50:20	148.99
4	2017-11-17 15:55:10	149.11
5	2017-11-17 16:00:03	149.25
6	2017-11-17 16:05:06	149.5175
7	2017-11-17 16:10:12	149.57
8	2017-11-17 16:15:30	149.59
9	2017-11-17 16:20:32	149.82
10	2017-11-17 16:25:47	149.96

merge_asof() example with direction

```
pd.merge_asof(visa, ibm, on=['date_time'],  
              suffixes=('_visa', '_ibm'),  
              direction='forward')
```

	date_time	close_visa	close_ibm
0	<u>2017-11-17 16:00:00</u>	<u>110.32</u>	<u>149.25</u>
1	2017-11-17 17:00:00	110.24	149.6184
2	2017-11-17 18:00:00	110.065	149.59
3	2017-11-17 19:00:00	110.04	149.505
4	2017-11-17 20:00:00	110.0	149.42
5	2017-11-17 21:00:00	109.9966	149.26
6	2017-11-17 22:00:00	109.82	148.97

Table Name: `ibm`

	date_time	close
0	2017-11-17 15:35:12	149.3
1	2017-11-17 15:40:34	149.13
2	2017-11-17 15:45:50	148.98
3	2017-11-17 15:50:20	148.99
4	2017-11-17 15:55:10	149.11
5	<u>2017-11-17 16:00:03</u>	<u>149.25</u>
6	2017-11-17 16:05:06	149.5175
7	2017-11-17 16:10:12	149.57
8	2017-11-17 16:15:30	149.59
9	2017-11-17 16:20:32	149.82
10	2017-11-17 16:25:47	149.96

When to use merge_asof()

- Data sampled from a process
- Developing a training set (no data leakage)

merge_asof()

It can be used to do fuzzy matching of dates between tables. ✓

Has an argument that can be set to 'forward' to select the first row in the right table whose key column is greater than or equal to the left's. ✓

After matching two tables, if there are missing values at the top of the table from the right table, this function can fill them in. ✓

both

This function can be used when working with ordered or time-series data. ✓

This function can set the suffix for overlapping column names. ✓

.merge_ordered

It allows for a right join during the merge. ✓

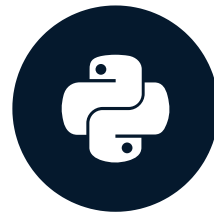
If it cannot match the rows of the tables exactly, it can use forward fill to interpolate the missing data. ✓

Let's practice!

JOINING DATA WITH PANDAS

Selecting data with .query()

JOINING DATA WITH PANDAS



Aaren Stubberfield
Instructor

The .query() method

```
.query('SOME SELECTION STATEMENT')
```

- Accepts an input string
 - Input string used to determine what rows are returned
 - Input string similar to statement after WHERE clause in SQL statement
 - Prior knowledge of SQL is not necessary

Querying on a single condition

This table is `stocks`

```
stocks.query('nike >= 90')
```

	date	<u>disney</u>	<u>nike</u>
0	2019-07-01	143.009995	86.029999
1	2019-08-01	137.259995	84.5
2	2019-09-01	130.320007	93.919998
3	2019-10-01	129.919998	89.550003
4	2019-11-01	151.580002	93.489998
5	2019-12-01	144.630005	101.309998
6	2020-01-01	138.309998	96.300003
7	2020-02-01	117.650002	89.379997
8	2020-03-01	96.599998	82.739998
9	2020-04-01	99.580002	84.629997

	date	<u>disney</u>	<u>nike</u>
2	2019-09-01	130.320007	93.919998
4	2019-11-01	151.580002	93.489998
5	2019-12-01	144.630005	101.309998
6	2020-01-01	138.309998	96.300003

Querying on a multiple conditions, "and", "or"

This table is `stocks`

	date	disney	nike
0	2019-07-01	143.009995	86.029999
1	2019-08-01	137.259995	84.5
2	2019-09-01	130.320007	93.919998
3	2019-10-01	129.919998	89.550003
4	2019-11-01	151.580002	93.489998
5	2019-12-01	144.630005	101.309998
6	2020-01-01	138.309998	96.300003
7	2020-02-01	117.650002	89.379997
8	2020-03-01	96.599998	82.739998
9	2020-04-01	99.580002	84.629997

```
stocks.query('nike > 90 and disney < 140')
```

	date	disney	nike
2	2019-09-01	130.320007	93.919998
6	2020-01-01	138.309998	96.300003

```
stocks.query('nike > 96 or disney < 98')
```

	date	disney	nike
5	2019-12-01	144.630005	101.309998
6	2020-01-01	138.309998	96.300003
28	2020-03-01	96.599998	82.739998

Updated dataset

This table is `stocks_long`

	date	stock	close
0	2019-07-01	disney	143.009995
1	2019-08-01	disney	137.259995
2	2019-09-01	disney	130.320007
3	2019-10-01	disney	129.919998
4	2019-11-01	disney	151.580002
5	2019-07-01	nike	86.029999
6	2019-08-01	nike	84.5
7	2019-09-01	nike	93.919998
8	2019-10-01	nike	89.550003
9	2019-11-01	nike	93.489998

Using .query() to select text

```
stocks_long.query('stock=="disney" or (stock=="nike" and close < 90)')
```

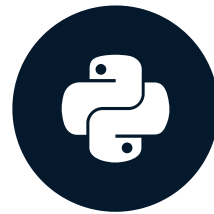
	date	stock	close
0	2019-07-01	disney	143.009995
1	2019-08-01	disney	137.259995
2	2019-09-01	disney	130.320007
3	2019-10-01	disney	129.919998
4	2019-11-01	disney	151.580002
5	2019-07-01	nike	86.029999
6	2019-08-01	nike	84.5
8	2019-10-01	nike	89.550003

Let's practice!

JOINING DATA WITH PANDAS

Reshaping data with `.melt()`

JOINING DATA WITH PANDAS



Aaren Stubberfield
Instructor

Wide versus long data

Wide Format



	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

Long Format



	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150

What does the `.melt()` method do?

- The melt method will allow us to unpivot our dataset

	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150



	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150

Dataset in wide format

This table is called `social_fin`

	financial	company	2019	2018	2017	2016
0	total_revenue	twitter	3459329	3042359	2443299	2529619
1	gross_profit	twitter	2322288	2077362	1582057	1597379
2	net_income	twitter	1465659	1205596	-108063	-456873
3	total_revenue	facebook	70697000	55838000	40653000	27638000
4	gross_profit	facebook	57927000	46483000	35199000	23849000
5	net_income	facebook	18485000	22112000	15934000	10217000

Example of .melt()

```
social_fin_tall = social_fin.melt(id_vars=['financial', 'company'])  
print(social_fin_tall.head(10))
```

	financial	company	variable	value
0	total_revenue	twitter	2019	3459329
1	gross_profit	twitter	2019	2322288
2	net_income	twitter	2019	1465659
3	total_revenue	facebook	2019	70697000
4	gross_profit	facebook	2019	57927000
5	net_income	facebook	2019	18485000
6	total_revenue	twitter	2018	3042359
7	gross_profit	twitter	2018	2077362
8	net_income	twitter	2018	1205596
9	total_revenue	facebook	2018	55838000

Melting with value_vars

```
social_fin_tall = social_fin.melt(id_vars=['financial', 'company'],  
                                value_vars=['2018', '2017'])  
  
print(social_fin_tall.head(9))
```

	financial	company	variable	value
0	total_revenue	twitter	2018	3042359
1	gross_profit	twitter	2018	2077362
2	net_income	twitter	2018	1205596
3	total_revenue	facebook	2018	55838000
4	gross_profit	facebook	2018	46483000
5	net_income	facebook	2018	22112000
6	total_revenue	twitter	2017	2443299
7	gross_profit	twitter	2017	1582057
8	net_income	twitter	2017	-108063

Melting with column names

```
social_fin_tall = social_fin.melt(id_vars=['financial', 'company'],  
                                value_vars=['2018', '2017'],  
                                var_name='year', value_name='dollars')  
  
print(social_fin_tall.head(8))
```

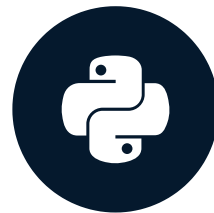
	financial	company	year	dollars
0	total_revenue	twitter	2018	3042359
1	gross_profit	twitter	2018	2077362
2	net_income	twitter	2018	1205596
3	total_revenue	facebook	2018	55838000
4	gross_profit	facebook	2018	46483000
5	net_income	facebook	2018	22112000
6	total_revenue	twitter	2017	2443299
7	gross_profit	twitter	2017	1582057

Let's practice!

JOINING DATA WITH PANDAS

Course wrap-up

JOINING DATA WITH PANDAS



Aaren Stubberfield
Instructor

You're this high performance race car now



¹ Photo by jae park from Pexels

Data merging basics

- Inner join using `.merge()`
- One-to-one and one-to-many relationships
- Merging multiple tables

Merging tables with different join types

- Inner join using `.merge()`
- One-to-one and one-to-many relationships
- Merging multiple tables
- Left, right, and outer joins
- Merging a table to itself and merging on indexes

Advanced merging and concatenating

- Inner join using `.merge()`
- One-to-one and one-to-many relationships
- Merging multiple tables
- Left, right, and outer joins
- Merging a table to itself and merging on indexes
- Filtering joins
 - semi and anti joins
- Combining data vertically with `.concat()`
- Verify data integrity

Merging ordered and time-series data

- Inner join using `.merge()`
- One-to-one and one-to-one relationships
- Merging multiple tables
- Left, right, and outer joins
- Merging a table to itself and merging on indexes
- Filtering joins
 - semi and anti joins
- Combining data vertically with `.concat()`
- Verify data integrity
- Ordered data
 - `merge_ordered()` and `merge_asof()`
- Manipulating data with `.melt()`

Thank you!
JOINING DATA WITH PANDAS