AI Bootcamp

Combining DataFrames with Pandas

Module 5 Day 1

Class Objectives

By the end of class, you will be able to:

1. Understand the difference between merging, joining, and concatenating.
2. Concatenate DataFrames vertically and horizontally.
3. Join DataFrames on indices.
4. Merge DataFrames vertically and horizontally.
5. Handle duplicate entries during merging.

Instructor **Demonstration**

Combining Data

Concatenation

Concatenation is the process of combining DataFrames across rows or columns.

Combining Tables Using concat

States Populations table

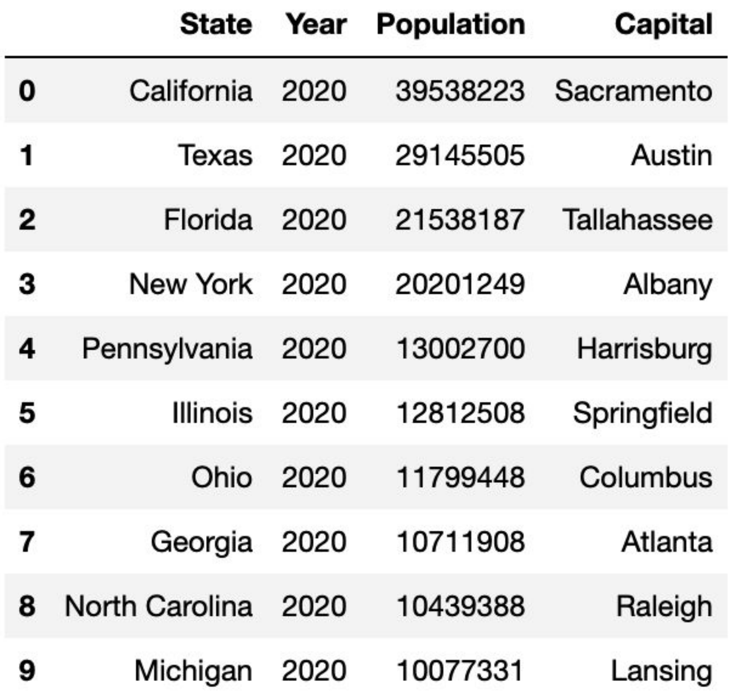
	State	Year	Population
0	California	2020	39538223
1	Texas	2020	29145505
2	Florida	2020	21538187
3	New York	2020	20201249
4	Pennsylvania	2020	13002700
5	Illinois	2020	12812508
6	Ohio	2020	11799448
7	Georgia	2020	10711908
8	North Carolina	2020	10439388
9	Michigan	2020	10077331

States Capitals table

	State	Capital
0	California	Sacramento
1	Texas	Austin
2	Florida	Tallahassee
3	New York	Albany
4	Pennsylvania	Harrisburg
5	Illinois	Springfield
6	Ohio	Columbus
7	Georgia	Atlanta
8	North Carolina	Raleigh
9	Michigan	Lansing

If we want to add the “Capitals” column from the “States Populations” table, we’ll need to combine the two tables along rows and specify which row from the “States Capitals table” to add to the “States Populations” table

```
states_pop_capitals_concat = pd.concat([states_population, states_capitals['Capital']], axis=1)
```



	State	Year	Population	Capital
0	California	2020	39538223	Sacramento
1	Texas	2020	29145505	Austin
2	Florida	2020	21538187	Tallahassee
3	New York	2020	20201249	Albany
4	Pennsylvania	2020	13002700	Harrisburg
5	Illinois	2020	12812508	Springfield
6	Ohio	2020	11799448	Columbus
7	Georgia	2020	10711908	Atlanta
8	North Carolina	2020	10439388	Raleigh
9	Michigan	2020	10077331	Lansing

Joining

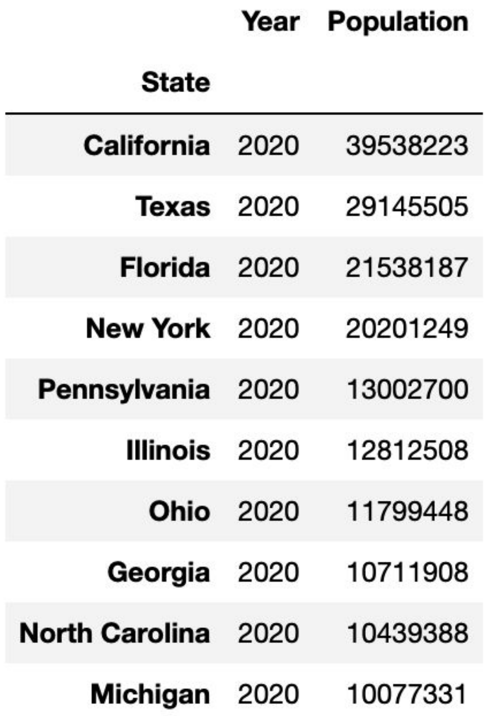
Joining is used to combine DataFrames across columns on a common index.

Combining Tables Using `join`

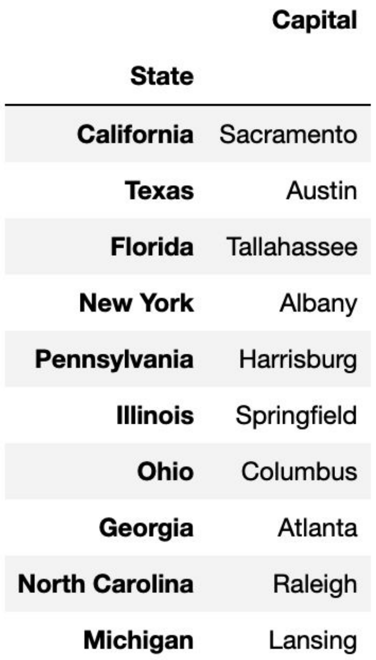
	State	Year	Population
0	California	2020	39538223
1	Texas	2020	29145505
2	Florida	2020	21538187
3	New York	2020	20201249
4	Pennsylvania	2020	13002700
5	Illinois	2020	12812508
6	Ohio	2020	11799448
7	Georgia	2020	10711908
8	North Carolina	2020	10439388
9	Michigan	2020	10077331

To combine the two tables on the “State” column, we have to set the “State” column in both tables as the index.

	State	Capital
0	California	Sacramento
1	Texas	Austin
2	Florida	Tallahassee
3	New York	Albany
4	Pennsylvania	Harrisburg
5	Illinois	Springfield
6	Ohio	Columbus
7	Georgia	Atlanta
8	North Carolina	Raleigh
9	Michigan	Lansing



	Year	Population
State		
California	2020	39538223
Texas	2020	29145505
Florida	2020	21538187
New York	2020	20201249
Pennsylvania	2020	13002700
Illinois	2020	12812508
Ohio	2020	11799448
Georgia	2020	10711908
North Carolina	2020	10439388
Michigan	2020	10077331

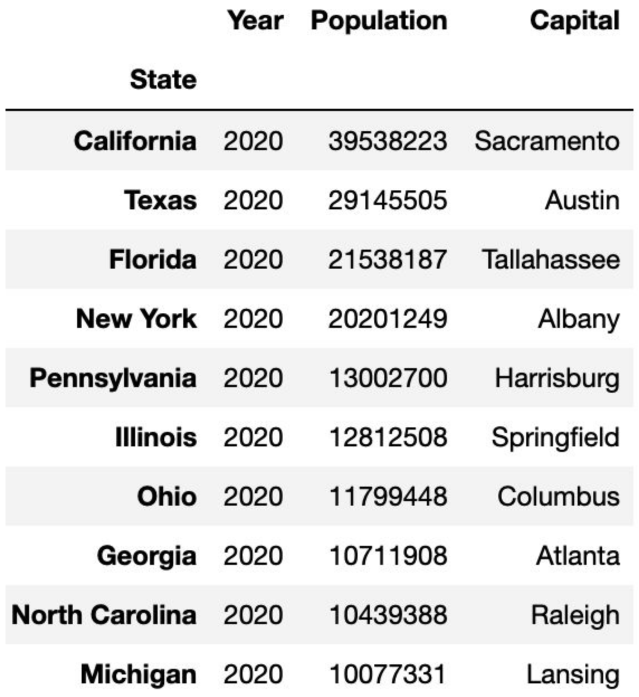


	Capital
State	
California	Sacramento
Texas	Austin
Florida	Tallahassee
New York	Albany
Pennsylvania	Harrisburg
Illinois	Springfield
Ohio	Columbus
Georgia	Atlanta
North Carolina	Raleigh
Michigan	Lansing

To set the index we use:

```
states_population.set_index('State', inplace=True)

states_capitals.set_index('State', inplace=True)
```



	Year	Population	Capital
State			
California	2020	39538223	Sacramento
Texas	2020	29145505	Austin
Florida	2020	21538187	Tallahassee
New York	2020	20201249	Albany
Pennsylvania	2020	13002700	Harrisburg
Illinois	2020	12812508	Springfield
Ohio	2020	11799448	Columbus
Georgia	2020	10711908	Atlanta
North Carolina	2020	10439388	Raleigh
Michigan	2020	10077331	Lansing

Merging

Merging is used to combine DataFrames across columns using a common column or index.

Combining Tables Using `merge`

	State	Year	Population
0	California	2020	39538223
1	Texas	2020	29145505
2	Florida	2020	21538187
3	New York	2020	20201249
4	Pennsylvania	2020	13002700
5	Illinois	2020	12812508
6	Ohio	2020	11799448
7	Georgia	2020	10711908
8	North Carolina	2020	10439388
9	Michigan	2020	10077331

To combine the two DataFrames, we specify the “State” column to perform the merge.

	State	Capital
0	California	Sacramento
1	Texas	Austin
2	Florida	Tallahassee
3	New York	Albany
4	Pennsylvania	Harrisburg
5	Illinois	Springfield
6	Ohio	Columbus
7	Georgia	Atlanta
8	North Carolina	Raleigh
9	Michigan	Lansing

	State	Year	Population	Capital
0	California	2020	39538223	Sacramento
1	Texas	2020	29145505	Austin
2	Florida	2020	21538187	Tallahassee
3	New York	2020	20201249	Albany
4	Pennsylvania	2020	13002700	Harrisburg
5	Illinois	2020	12812508	Springfield
6	Ohio	2020	11799448	Columbus
7	Georgia	2020	10711908	Atlanta
8	North Carolina	2020	10439388	Raleigh
9	Michigan	2020	10077331	Lansing

Instructor **Demonstration**

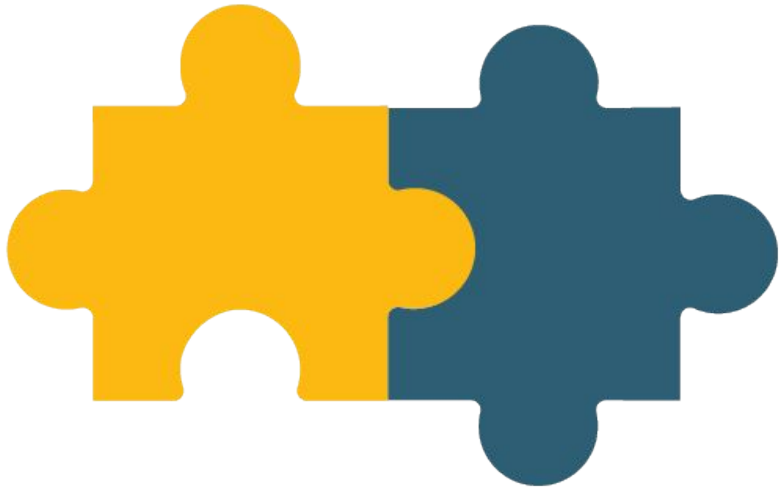
Concatenating DataFrames

Concatenation

Pandas has a **`concat`** function that can be used to combine Dataframes.

When combining DataFrames across **columns**, the columns from one DataFrame are placed **adjacent** to columns from another DataFrame.

When combining DataFrames across **rows**, the rows from one DataFrame are placed **below** the rows from another DataFrame.



Concatenation

The `concat` function accepts the following arguments:

- A list of DataFrames to be joined
- The `axis` to join on (by column or row)
- The `join` operation (inner vs. outer)

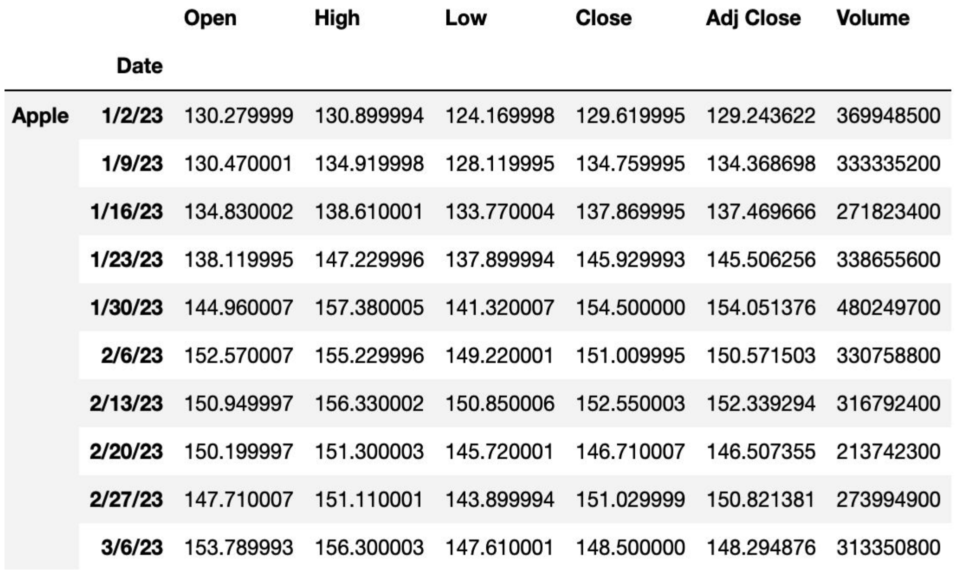
The `concat` function creates a new DataFrame that includes data from all datasets that were joined. The amount of data returned will depend on the type of `join` performed when concatenating.

Concatenating DataFrames: Creating an Index

The **concat** function also allows you to add a list of string values as part of the table index based on the column values.

```
# Join by rows and add the stock ticker as the key.
joined_data_rows = pd.concat([apple_data, goog_data, meta_data], axis="rows",
                             join="inner", keys=['Apple','Google', 'Meta'] )

joined_data_rows.head(10)
```



		Open	High	Low	Close	Adj Close	Volume
	Date						
Apple	**1/2/23**	130.279999	130.899994	124.169998	129.619995	129.243622	369948500
	1/9/23	130.470001	134.919998	128.119995	134.759995	134.368698	333335200
	1/16/23	134.830002	138.610001	133.770004	137.869995	137.469666	271823400
	1/23/23	138.119995	147.229996	137.899994	145.929993	145.506256	338655600
	1/30/23	144.960007	157.380005	141.320007	154.500000	154.051376	480249700
	2/6/23	152.570007	155.229996	149.220001	151.009995	150.571503	330758800
	2/13/23	150.949997	156.330002	150.850006	152.550003	152.339294	316792400
	2/20/23	150.199997	151.300003	145.720001	146.710007	146.507355	213742300
	2/27/23	147.710007	151.110001	143.899994	151.029999	150.821381	273994900
	3/6/23	153.789993	156.300003	147.610001	148.500000	148.294876	313350800

Activity:

Concatenating Country Products

In this activity, you will practice concatenating DataFrames related to country products.

Suggested Time:
15 Minutes

Time’s up!

Let’s review

Questions?

Instructor **Demonstration**

Joining DataFrames

Joining

The `join` function will join DataFrames on the index rather than columns.

By default, `join` will attempt to perform a left join on indices but won't directly merge DataFrames. All the columns, including those with matching names, are retained in the resulting DataFrame.

If the DataFrames being combined have columns with the same name, we can add text to these using the `lusffix` and `rsuffix` parameters to distinguish between them.

Or, we can add a suffix to all the columns prior to joining by using the `add_suffix` method.

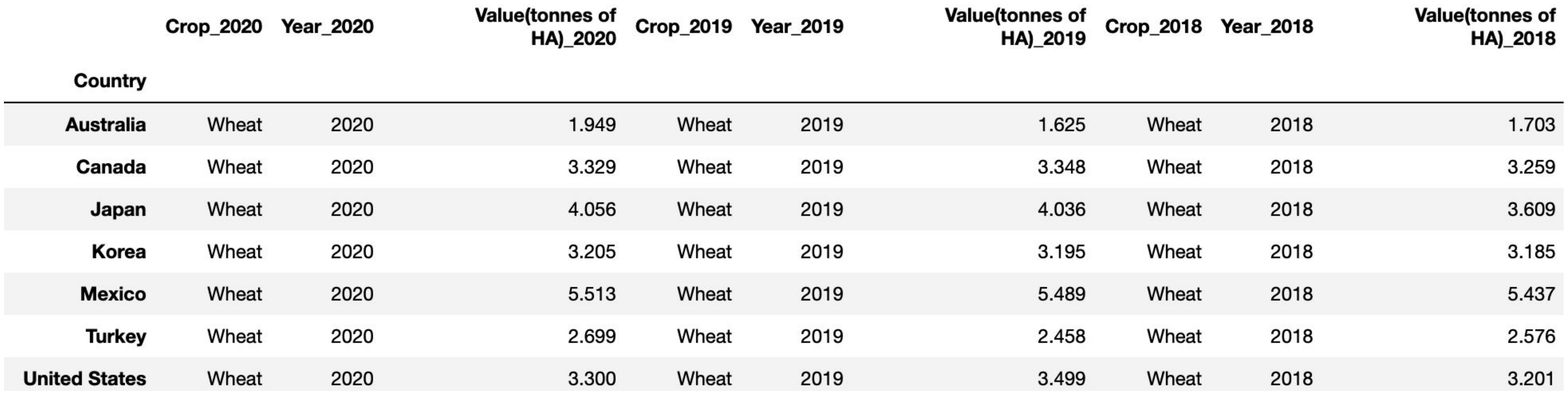
Joining DataFrames

```python
# Join the 2018 and 2019 wheat data where the left suffix is 2019 and right suffix is 2018.
wheat_2018_19_data = wheat_2019_df.join(wheat_2018_df, lsuffix='_2019', rsuffix='_2018')
wheat_2018_19_data
```

	Crop_2019	Year_2019	Value(tonnes of HA)_2019	Crop_2018	Year_2018	Value(tonnes of HA)_2018
Country						
Australia	Wheat	2019	1.625	Wheat	2018	1.703
Canada	Wheat	2019	3.348	Wheat	2018	3.259
Japan	Wheat	2019	4.036	Wheat	2018	3.609
Korea	Wheat	2019	3.195	Wheat	2018	3.185
Mexico	Wheat	2019	5.489	Wheat	2018	5.437
Turkey	Wheat	2019	2.458	Wheat	2018	2.576
United States	Wheat	2019	3.499	Wheat	2018	3.201

Joining DataFrames

```
# Join the 2018 and 2019 wheat data with the 2020 wheat data and add the suffix '_2020' to the 2020 data.
all_wheat_data = wheat_2020_df.add_suffix('_2020').join(wheat_2018_19_data)
all_wheat_data
```



	Crop_2020	Year_2020	Value(tonnes of HA)_2020	Crop_2019	Year_2019	Value(tonnes of HA)_2019	Crop_2018	Year_2018	Value(tonnes of HA)_2018
Country									
Australia	Wheat	2020	1.949	Wheat	2019	1.625	Wheat	2018	1.703
Canada	Wheat	2020	3.329	Wheat	2019	3.348	Wheat	2018	3.259
Japan	Wheat	2020	4.056	Wheat	2019	4.036	Wheat	2018	3.609
Korea	Wheat	2020	3.205	Wheat	2019	3.195	Wheat	2018	3.185
Mexico	Wheat	2020	5.513	Wheat	2019	5.489	Wheat	2018	5.437
Turkey	Wheat	2020	2.699	Wheat	2019	2.458	Wheat	2018	2.576
United States	Wheat	2020	3.300	Wheat	2019	3.499	Wheat	2018	3.201

Activity:

Joining Alternative Energy Data

In this activity, you will practice combining DataFrames using the `join` function and appending this data to a combined DataFrame. The data of different years is joined to provide a comprehensive view of the data over time.

Suggested Time:
15 Minutes

Time’s up!

Let’s review

Questions?

Break

15 mins

Instructor **Demonstration**

Merging DataFrames

Merging DataFrames

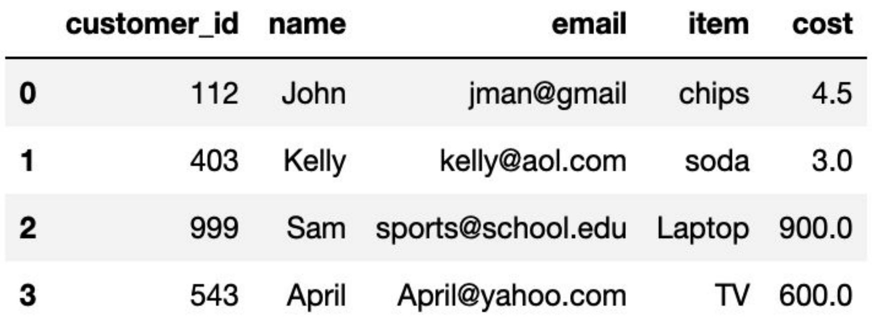
What’s Merging?

- Sometimes, an analyst will receive data split across multiple tables and sources.
- Working across multiple tables is error-prone and confusing.
- **Merging** is the process of combining two tables based on shared data.
- Shared data can be an identical column in both tables or a shared index.
- In pandas, we can merge separate DataFrames by using the `pd.merge()` method.

Merging DataFrames: Inner Joins

An inner join is the default method for combining DataFrames when using `pd.merge()`. It only returns data with matching values. Rows that do not include matching data will be dropped from the combined DataFrame.

```
# Merge two DataFrames. An inner join is used by default.
merge_df = pd.merge(info_df, items_df, on="customer_id")
merge_df
```



	customer_id	name	email	item	cost
0	112	John	jman@gmail	chips	4.5
1	403	Kelly	kelly@aol.com	soda	3.0
2	999	Sam	sports@school.edu	Laptop	900.0
3	543	April	April@yahoo.com	TV	600.0

Merging DataFrames: Outer Joins

Outer joins combine the DataFrames whether or not the rows match. They must be declared as a parameter within the **`pd.merge()`** method by using the syntax **`how="outer"`**.

```
# Merge two DataFrames using an outer join
merge_df = pd.merge(info_df, items_df, on="customer_id", how="outer")
merge_df
```

	customer_id	name	email	item	cost
0	112	John	jman@gmail	chips	4.5
1	403	Kelly	kelly@aol.com	soda	3.0
2	999	Sam	sports@school.edu	Laptop	900.0
3	543	April	April@yahoo.com	TV	600.0
4	123	Bobbo	HeyImBobbo@msn.com	NaN	NaN
5	654	NaN	NaN	Cooler	150.0

Merging DataFrames: Left Joins

These joins protect the data contained within one DataFrame, like an outer join does, while also dropping the rows with null data from the other DataFrame

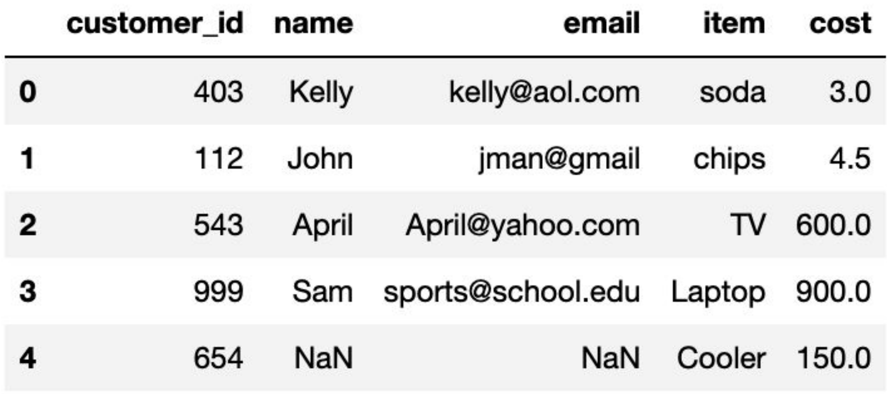
```
# Merge two DataFrames using a left join
merge_df = pd.merge(info_df, items_df, on="customer_id", how="left")
merge_df
```

	customer_id	name	email	item	cost
0	112	John	jman@gmail	chips	4.5
1	403	Kelly	kelly@aol.com	soda	3.0
2	999	Sam	sports@school.edu	Laptop	900.0
3	543	April	April@yahoo.com	TV	600.0
4	123	Bobbo	HeyImBobbo@msn.com	NaN	NaN

Merging DataFrames: Right Joins

These joins protect the data contained within one DataFrame, like an outer join does, while also dropping the rows with null data from the other DataFrame

```
# Merge two DataFrames using a right join
merge_df = pd.merge(info_df, items_df, on="customer_id", how="right")
merge_df
```



	customer_id	name	email	item	cost
0	403	Kelly	kelly@aol.com	soda	3.0
1	112	John	jman@gmail	chips	4.5
2	543	April	April@yahoo.com	TV	600.0
3	999	Sam	sports@school.edu	Laptop	900.0
4	654	NaN	NaN	Cooler	150.0

Activity:
Census Merging

In this activity, you will merge the two Census datasets that we created in the last class and then do a calculation and sort the values.

Suggested Time:
15 Minutes

Time's up!

Let's review

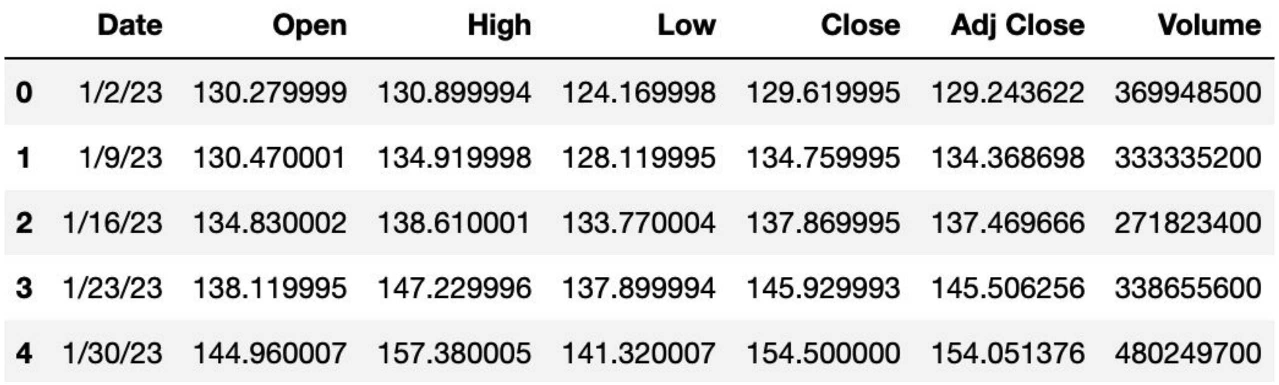
Questions?

Instructor **Demonstration**

Merging Duplicate Columns

Merging Duplicate Columns

When we merge DataFrames, there will often be columns with the same name, like stock data.



	Date	Open	High	Low	Close	Adj Close	Volume
0	1/2/23	130.279999	130.899994	124.169998	129.619995	129.243622	369948500
1	1/9/23	130.470001	134.919998	128.119995	134.759995	134.368698	333335200
2	1/16/23	134.830002	138.610001	133.770004	137.869995	137.469666	271823400
3	1/23/23	138.119995	147.229996	137.899994	145.929993	145.506256	338655600
4	1/30/23	144.960007	157.380005	141.320007	154.500000	154.051376	480249700

Merging Duplicate Columns

The best option is to merge the DataFrames on the “Date” column using the default “inner” join to prevent losing information.

```
# Merge Apple stock with Google stock on the date using pd.merge().
merged_apple_google = pd.merge(apple_data, google_data, on="Date")
merged_apple_google.head(10)
```

	Date	Open_x	High_x	Low_x	Close_x	Adj Close_x	Volume_x	Open_y	High_y	Low_y	Close_y	Adj Close_y	Volume_y
0	1/2/23	130.279999	130.899994	124.169998	129.619995	129.243622	369948500	89.830002	91.550003	85.570000	88.160004	88.160004	97533700
1	1/9/23	130.470001	134.919998	128.119995	134.759995	134.368698	333335200	89.195000	92.980003	86.699997	92.800003	92.800003	113236000
2	1/16/23	134.830002	138.610001	133.770004	137.869995	137.469666	271823400	92.779999	99.419998	90.839996	99.279999	99.279999	124989900
3	1/23/23	138.119995	147.229996	137.899994	145.929993	145.506256	338655600	99.129997	101.580002	95.262001	100.709999	100.709999	143746600
4	1/30/23	144.960007	157.380005	141.320007	154.500000	154.051376	480249700	98.745003	108.820000	97.519997	105.220001	105.220001	156510500

Merging Duplicate Columns

```
# Rename the columns.
merged_apple_google =
merged_apple_google.rename(columns={"Open_x":"Apple_Open","High_x":"Apple_High",
                       "Low_x": "Apple_Low", "Close_x": "Apple_Close",
                       "Adj Close_x": "Apple_Adj_Close", "Volume_x":"Apple_Volume",
                       "Open_y": "Google_Open","High_y": "Google_High",
                       "Low_y": "Google_Low", "Close_y": "Google_Close",
                       "Adj Close_y": "Google_Adj_Close", "Volume_y": "Google_Volume"})

merged_apple_google.head(10)
```

	Date	Apple_Open	Apple_High	Apple_Low	Apple_Close	Apple_Adj_Close	Apple_Volume	Google_Open	Google_High	Google_Low	Google_Close
0	1/2/23	130.279999	130.899994	124.169998	129.619995	129.243622	369948500	89.830002	91.550003	85.570000	88.160004
1	1/9/23	130.470001	134.919998	128.119995	134.759995	134.368698	333335200	89.195000	92.980003	86.699997	92.800003
2	1/16/23	134.830002	138.610001	133.770004	137.869995	137.469666	271823400	92.779999	99.419998	90.839996	99.279999
3	1/23/23	138.119995	147.229996	137.899994	145.929993	145.506256	338655600	99.129997	101.580002	95.262001	100.709999
4	1/30/23	144.960007	157.380005	141.320007	154.500000	154.051376	480249700	98.745003	108.820000	97.519997	105.220001

Activity:

Merging Crop Production

In this activity, you will merge G20 crop datasets from 2018, 2019, and 2020. Then clean the merged DataFrames by removing or renaming duplicate columns.

Suggested Time:

15 Minutes

Time’s up!

Let’s review

Questions?

Let’s **recap**

Recap

After today's lesson you are able to:

1. Understand the difference between merging, joining, and concatenating.
2. Concatenate DataFrames vertically and horizontally.
3. Join DataFrames on indices.
4. Merge DataFrames vertically and horizontally.
5. Handle duplicate entries during merging.

Next

In the next lesson, you'll learn about the concept of grouping data, the application of aggregations on grouped data, the `agg( ) function`, custom Python functions to transform grouped data, multi-index aggregations, and the concept of binning.

Questions?

The End