Pandas

Data Boot Camp

Lesson 4.1

The Big Picture

Excel Unit
Excel
VBA
Unit Assessment

Python Unit
Python
Pandas
Matplotlib
APIs
Unit Assessment

Databases Unit
SQL
ETL
SQLite
Unit Assessment

Visualization Unit
Web Scraping
Plotly
Leaflet
Tableau
Unit Assessment

Advanced Topics Unit
R—Stats
Big Data—AWS
Machine Learning
Unit Assessment

Final Project
Project Design
Train Model &
Build Database
*Dashboards
Presentation*

01 02 03 04 05 06

Boot Camp Pointers

If you're struggling with this module, remember that you have the following resources to help you:

Learning Assistants

Office Hours

Tutoring Services

Module 4

This Week: Pandas

This Week: Pandas

By the end of this week, you'll know how to:

Read an external CSV file into a DataFrame.

Determine data types of row values in a DataFrame.

Format and retrieve data from columns of a DataFrame.

Merge, filter, slice, and sort a DataFrame.

Apply the `groupby()` function to a DataFrame.

Use multiple methods to perform a function on a DataFrame.

Perform mathematical calculations on columns of a DataFrame or Series.

This Week's Challenge

Using the skills learned throughout the week, help a mock school board with their investigation by adjusting specific data.

Career Connection

How will you use this module's content in your career?

Module 4

How to Succeed This Week

Pro Tip:

New syntax may not always be easy to remember, but don't worry! The documentation you need is just a click away.

Module 4

Today's Agenda

Today's Agenda

By completing today's activities, you'll learn the following skills:

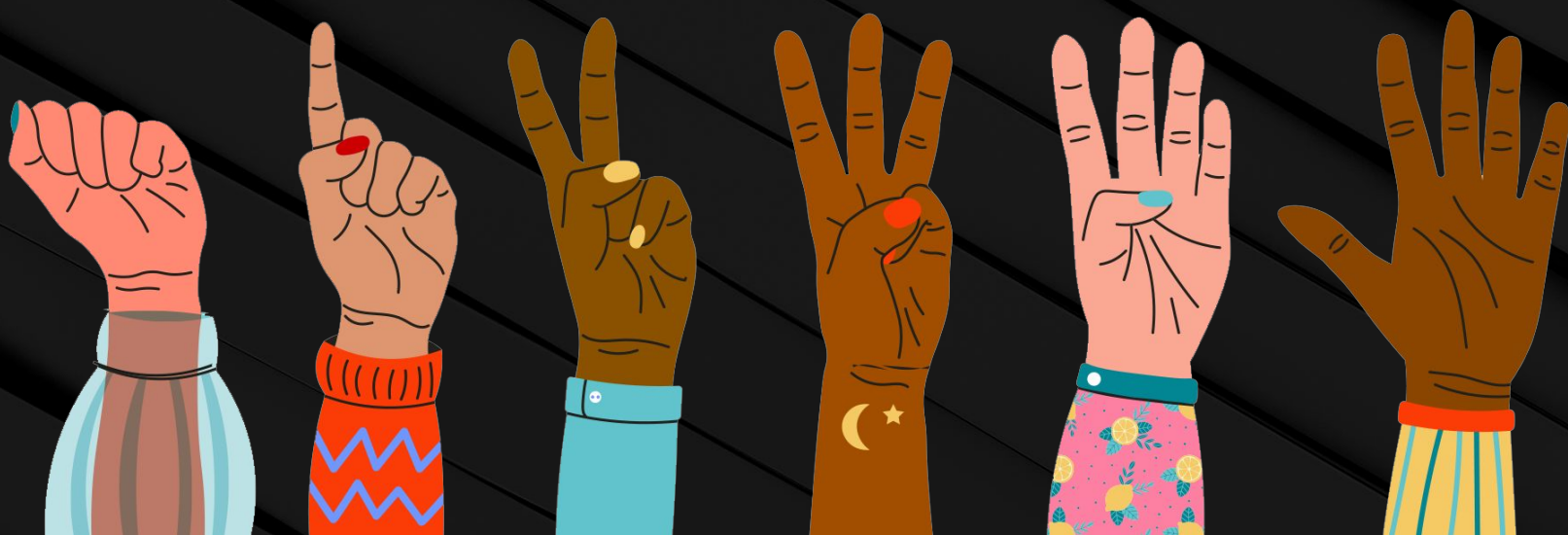| 01	Use a variety of Pandas methods and functions on a DataFrame

02	Merge DataFrames

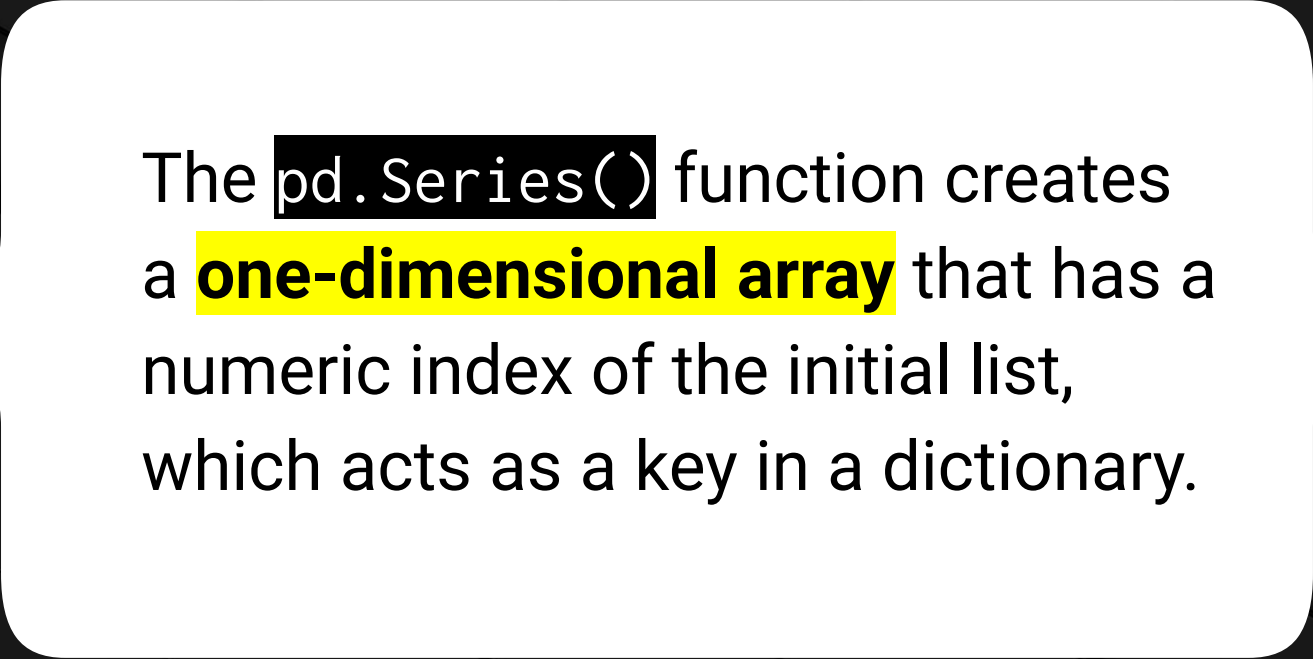| 03	Format DataFrame columns

Make sure you've downloaded any relevant class files!

FIST TO FIVE:

## How comfortable do you feel with this topic?

The `pd.Series()` function creates a **one-dimensional array** that has a numeric index of the initial list, which acts as a key in a dictionary.

Each Column in a DataFrame Is a Series

A `DataFrame` is a two-dimensional, labeled data structure, like a dictionary, with rows and columns of potentially different data types such as strings, integers, and floats, where data is aligned in a table, much like a spreadsheet.

DataFrames
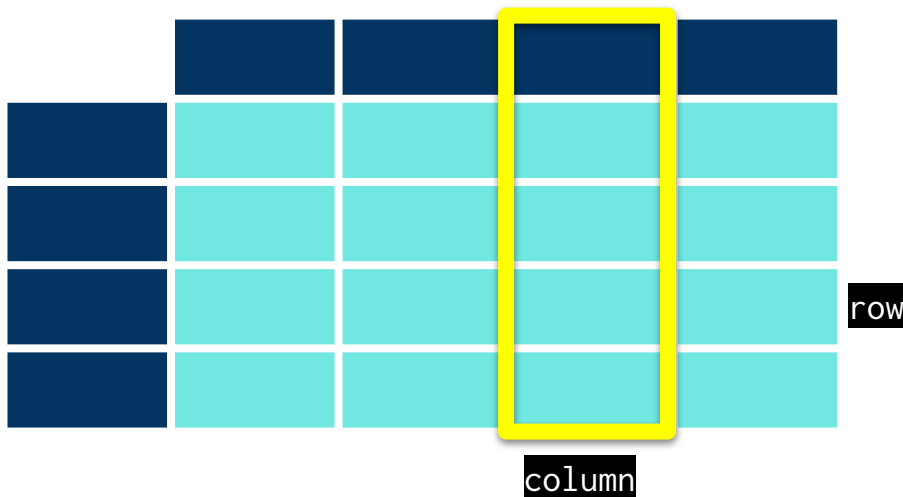
Series

What is the Pandas syntax
for creating a Series?

Pandas Syntax for Creating a Series

01 Import Pandas library

First, import Pandas library running `import pandas as pd`.

This method of import allows Pandas functions/methods to be called using the variable pd.

02 Create a Series

To create a Series, simply run `pd.Series()` function and place a list within the parentheses. Note that the index for the values within the Series will be the numeric index of the initial list.

There Are Multiple Ways to Create DataFrames

Create `DataFrames` from scratch

One of many different ways to create DataFrames from scratch is to use the `pd.DataFrame()` function and provide it with a list of dictionaries. Each dictionary will represent a new row where the keys become column headers, and the values will be placed inside the table.

Provide a dictionary of lists

Another way to use `pd.DataFrame()` is to provide a dictionary of lists. The keys of the dictionary will be the column headers, and the listed values will be placed into their respective rows.

Instructor Demonstration

Creating DataFrames

DataFrame Functions

`head()` takes a DataFrame and shows only its first five rows of data.

This number can be increased or decreased by placing an integer within the parentheses.
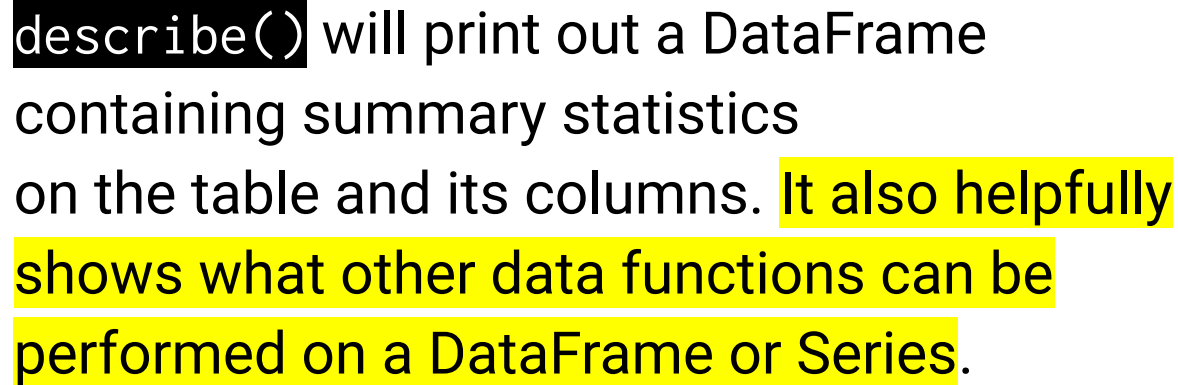
Built-in Function: `head()`

The `head()` method is helpful because it allows the programmer to look at a minified version of a much larger table, thus allowing them to make informed changes without having to search through the entire dataset.

In [3]:
```python
# Use Pandas to read data
data_file_df = pd.read_csv(data_file)
data_file_df.head()
```

Out[3]:

	id	First Name	Last Name	Gender	Amount
0	1	Todd	Lopez	M	8067.7
1	2	Joshua	White	M	7330.1
2	3	Mary	Lewis	F	16335.0
3	4	Emily	Burns	F	12460.8
4	5	Christina	Romero	F	15271.9

`describe()` will print out a DataFrame containing summary statistics on the table and its columns. It also helpfully shows what other data functions can be performed on a DataFrame or Series.

Built-in Function: `describe()`

The `describe()` method will print out a DataFrame containing some analytic information on the table and its columns. It also helpfully shows what other data functions can be performed on a DataFrame or Series.

```
In [4]:  # Display a statistical overview of the DataFrame
         data_file_df.describe()
```

Out[4]:

	id	Amount
count	1000.000000	1000.000000
mean	500.500000	10051.323600
std	288.819436	5831.230806
min	1.000000	3.400000
25%	250.750000	4854.875000
50%	500.500000	10318.050000
75%	750.250000	15117.425000
max	1000.000000	19987.400000

DataFrame Functions: Working With a Single Column

- Most data functions can also be performed on a Series by referencing a single column within the whole DataFrame, like referencing a key within a dictionary.

- In a DataFrame, we place the column header (i.e., "key") in brackets, and the output contains the values under that column header.

```
In [5]:  # Reference a single column within a DataFrame
         data_file_df["Amount"].head()

Out[5]:  0       8067.7
         1       7330.1
         2      16335.0
         3      12460.8
         4      15271.9
         Name: Amount, dtype: float64
```

DataFrame Functions: Working With Multiple Columns

Multiple columns can be referenced, too, by placing all of the desired column headers within a pair of double brackets. To retrieve data from two or more columns, two sets of brackets must be used or Pandas will return an error.

In [6]:
```python
# Reference multiple columns within a DataFrame
data_file_df[["Amount", "Gender"]].head()
```

Out[6]:

	Amount	Gender
0	8067.7	M
1	7330.1	M
2	16335.0	F
3	12460.8	F
4	15271.9	F

`unique()` looks into a Series and **returns all distinct values**.

Built-In Function: `unique()`

There are situations where it is helpful to list out all of the unique values stored within a column. This is precisely what the `unique()` function does, by looking into a Series and returning all of the different values within.

```
In [9]:  # The unique method shows every element of the series that appears only once
         unique = data_file_df["Last Name"].unique()
         unique

Out[9]: array(['Lopez', 'White', 'Lewis', 'Burns', 'Romero', 'Andrews', 'Baker',
               'Diaz', 'Burke', 'Richards', 'Hansen', 'Tucker', 'Wheeler',
               'Turner', 'Reynolds', 'Carpenter', 'Scott', 'Ryan', 'Marshall',
               'Fernandez', 'Olson', 'Riley', 'Woods', 'Wells', 'Gutierrez',
               'Harvey', 'Ruiz', 'Lee', 'Welch', 'Cooper', 'Nichols', 'Murray',
               'Gomez', 'Green', 'Jacobs', 'Griffin', 'Perry', 'Dunn', 'Gardner',
               'Gray', 'Walker', 'Harris', 'Lawrence', 'Black', 'Simpson', 'Sims',
               'Weaver', 'Carr', 'Owens', 'Stephens', 'Butler', 'Matthews', 'Cox',
               'Brooks', 'Austin', 'Moore', 'Hunter', 'Cunningham', 'Lane',
               'Montgomery', 'Vasquez', 'Freeman', 'Hernandez', 'Alexander',
               'Pierce', 'Mcdonald', 'Kelly', 'Foster', 'Bell', 'Johnson',
               'Bowman', 'Porter', 'Wood', 'Reid', 'Willis', 'Bishop',
               'Washington', 'Gonzales', 'Davis', 'Martinez', 'Martin', 'Long',
               'Howell', 'Hawkins', 'Knight', 'Price', 'Day', 'Bailey', 'Flores',
               'Young', 'Evans', 'Cruz', 'Chavez', 'Barnes', 'Coleman', 'Burton',
               'Clark', 'Carter', 'Franklin', 'Ellis', 'Miller', 'Allen', 'Mason',
               'Patterson', 'Stevens', 'Kim', 'Kelley', 'Robinson', 'Hughes',
               'Morgan', 'Dean', 'Stewart', 'Murphy', 'Fox', 'Simmons',
               'Thompson', 'Fuller', 'Peterson', 'Hanson', 'Wright', 'Reed',
               'Graham', 'Parker', 'Boyd', 'Taylor', 'Greene', 'George', 'Mills',
               'Duncan', 'Hill', 'Jordan', 'Stanley', 'Hall', 'James', 'Stone',
               'Warren', 'Fowler', 'Williamson', 'Lynch', 'Harper', 'Little',
               'Nguyen', 'Morrison', 'Ramirez', 'Howard', 'Watkins', 'Robertson',
               'Powell', 'Sanchez', 'Sanders', 'Grant', 'Ross', 'Mitchell',
               'Henderson', 'Rose', 'Perez', 'Berry', 'Watson', 'Gordon',
               'Morales', 'Arnold', 'Morris', 'Crawford', 'Smith', 'Medina',
               'Alvarez', 'Collins', 'Rodriguez', 'Mccoy', 'Bennett',
               'Richardson', 'Chapman', 'Johnston', 'Gilbert', 'Ford', 'Russell',
               'Nelson', 'Castillo', 'Cole', 'Rice', 'Payne', 'Frazier', 'Webb',
               'Armstrong', 'Wilson', 'Garza', 'Garrett', 'Spencer', 'Peters',
               'Sullivan', 'Brown', 'Williams', 'Gonzalez', 'Palmer', 'Fields',
               'Snyder', 'Jackson', 'Edwards', 'Anderson', 'Cook', 'Ramos',
               'Harrison', 'Lawson', 'Banks', 'Wallace', 'Ortiz', 'Gibson',
               'Reyes', 'Shaw', 'Ward', 'Perkins', 'Bradley', 'Rivera', 'Jenkins',
               'Hart', 'Phillips', 'Garcia', 'Fisher', 'King', 'Larson', 'Hunt',
               'Jones', 'Hudson', 'Myers', 'Hayes', 'Dixon', 'Schmidt', 'Moreno',
               'Rogers', 'Thomas', 'Meyer', 'Daniels', 'Bryant', 'Henry',
               'Campbell', 'Ferguson', 'Oliver', 'Ray', 'Carroll', 'Wagner',
               'Kennedy', 'Holmes'], dtype=object)
```

Built-in Function: `value_counts()`

Another method that holds similar functionality is `value_counts()`, which not only returns a list of all unique values within a series but also counts how many times a unique value appears.

```
In [10]:  # The value_counts method counts unique values in a column
          count = data_file_df["Gender"].value_counts()
          count

Out[10]:  M    515
          F    485
          Name: Gender, dtype: int64
```

Instructor Demonstration

DataFrame Functions

DataFrame Functions

Questions?

Activity: Training Grounds

In this activity, you will take a large DataFrame consisting of 200 rows, analyze it using some data functions, and then add a new column.

Suggested Time:

15 minutes

Reading CSVs and Column Manipulation

If you are given data that is in an .xlsx or .csv format, how do you create a DataFrame?

Create a DataFrame

01

You can import the `xlsx` file using using `pd.read_excel()`.

02

And you can import the `csv` file using `pd.read_csv()`.

Instructor Demonstration

Reading CSVs

Activity: GoodReads CSV

In this activity, you will take a large CSV of books, read it into Jupyter Notebook using Pandas, and clean up the columns.

Suggested Time:

15 minutes

Activity: GoodReads CSV

Two things to keep in mind while working on this activity
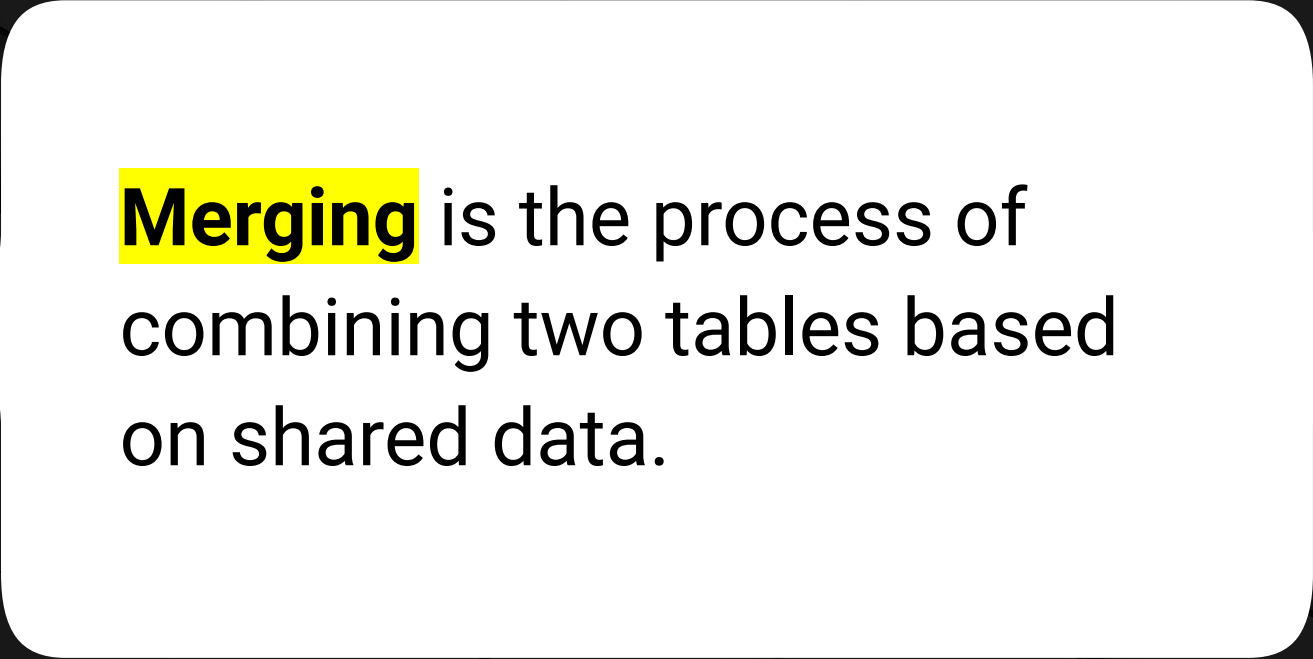
01

The initial CSV file is encoded using UTF-8, so it should be read using this encoding as well to ensure there are no strange characters hidden within the dataset.

02

There are a lot of columns that are being modified within this code, so it is useful to get all the columns in an array using the `.columns` attribute. This helps to make sure that all references are made accurately so as to avoid any potential errors.

Merging DataFrames

Merging is the process of combining two tables based on shared data.

Merging DataFrames

Sometimes an analyst will receive data split across multiple tables and sources

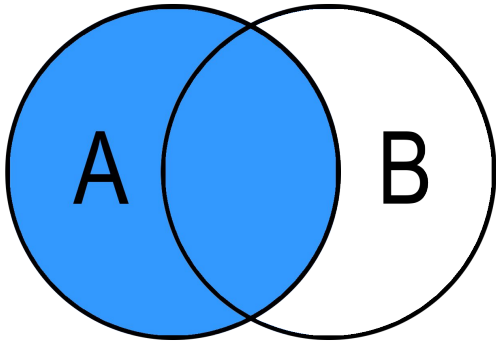Working across multiple tables is error prone and confusing

Shared data can be an identical column in both tables or a shared index

In Pandas, we can merge separate DataFrames using the `pd.merge()` method

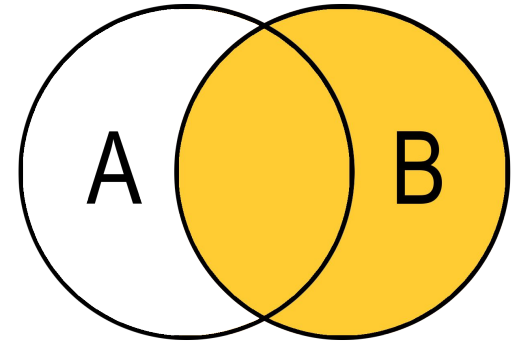# Joins Are VERY Important in Data Science!

When merging data tables, **joins** tell the program what data to keep.
Besides outer joins, there are three other common joins:

Left Outer Join:
All rows from table A,
even if they do not
exist in table B
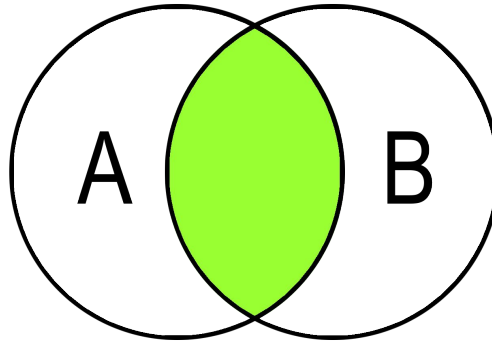
Inner Join:
Fetch the results that
exist in both tables

Right Outer Join:
All rows from table B,
even if they do not
exist in table A

Merging DataFrames: Inner Join

Inner joins are the default means through which DataFrames are combined using the `pd.merge()` method and will only return data whose values match. Rows that do not include matching data will be dropped from the combined DataFrame.

```
In [3]:  # Create DataFrames
         raw_data_items = {
             "customer_id": [403, 112, 543, 999, 654],
             "item": ["soda", "chips", "TV", "Laptop", "Cooler"],
             "cost": [3.00, 4.50, 600, 900, 150]
         }
         items_df = pd.DataFrame(raw_data_items, columns=[
                               "customer_id", "item", "cost"])

         items_df
```

Out[3]:

	customer_id	item	cost
0	403	soda	3.0
1	112	chips	4.5
2	543	TV	600.0
3	999	Laptop	900.0
4	654	Cooler	150.0
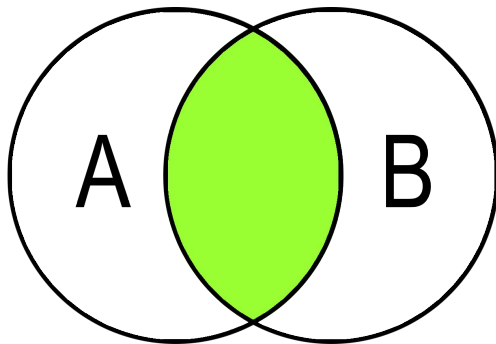
Inner Join:
Fetch the results that
exist in both tables

Merging DataFrames: Outer Join

Outer joins will combine the DataFrames regardless of whether any of the rows match and must be declared as a parameter within the `pd.merge()` method using the syntax `how="outer"`.

```
In [5]:   # Merge two dataframes using an outer join
          merge_df = pd.merge(info_df, items_df, on="customer_id", how="outer")
          merge_df
```

Out[5]:

	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

Any rows that do not include matching data will have the values within replaced with NaN instead.

Merging DataFrames: Right and Left Joins

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

In [6]:
```
# Merge two dataframes using a left join
merge_df = pd.merge(info_df, items_df, on="customer_id", how="left")
merge_df
```

Out[6]:

	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

In [7]:
```
# Merge two dataframes using a right join
merge_df = pd.merge(info_df, items_df, on="customer_id", how="right")
merge_df
```

Out[7]:

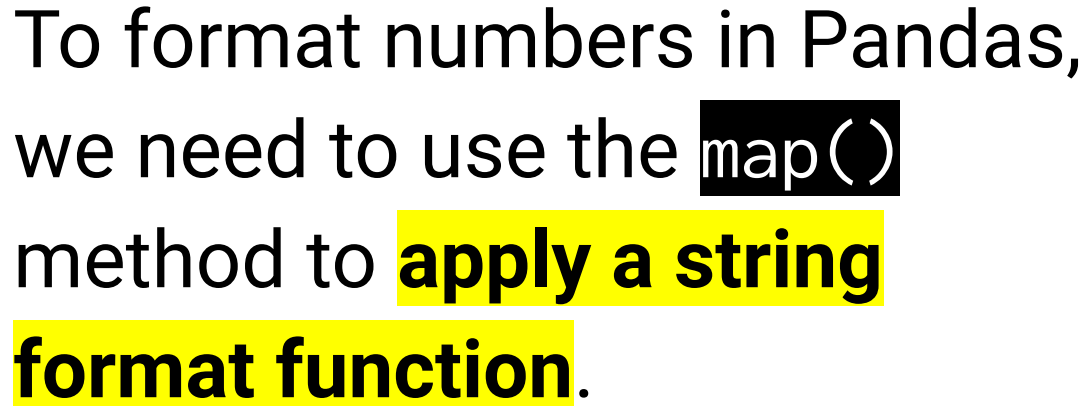|	customer_id	name	email	item	cost
0	112	John	jman@gmail	chips	4.5
1	403	Kelly	kelly@aol.com	soda	3.0
2	999	Sam	sports@school.edu	Laptop	900.0
3	543	April	April@yahoo.com	TV	600.0
4	654	NaN	NaN	Cooler	150.0

Instructor Demonstration

Merging Data
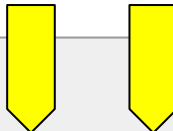
Questions?

Formatting and Mapping

To format numbers in Pandas, we need to use the `map()` method to **apply a string format function**.

Formatting Syntax

The formatting syntax used for mapping is, in a word, confusing. It uses strings containing curly brackets in order to determine how to style columns and this can make it rather difficult to understand at first glance.

```python
# Use Map to format all the columns
file_df["avg_cost"] = file_df["avg_cost"].map("${:.2f}".format)
file_df["population"] =
file_df["population"].map("${:,}".format)
file_df["other"] = file_df["other"].map("${:.2f}".format)
file_df.head()
```
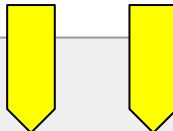
Formatting Syntax

A somewhat easy way to understand mapping strings is that it is almost akin to concatenating strings. Whatever is outside of the curly brackets is added before/after the initial value which is modified by whatever is contained within the curly brackets.

```python
# Use Map to format all the columns
file_df["avg_cost"] = file_df["avg_cost"].map("${:.2f}".format)
file_df["population"] =
file_df["population"].map("${:,}".format)
file_df["other"] = file_df["other"].map("${:.2f}".format)
file_df.head()
```
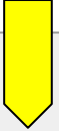
Formatting Syntax

So, to convert values into a typical dollar format, one would use `"${:.2f}"`.

This places a dollar sign before the value which has been rounded to two decimal points.

```python
# Use Map to format all the columns
file_df["avg_cost"] = file_df["avg_cost"].map("${:.2f}".format)
file_df["population"] =
file_df["population"].map("${:,}".format)
file_df["other"] = file_df["other"].map("${:.2f}".format)
file_df.head()
```

Formatting Syntax

Using `"{:,}"` will split a number up so that it uses comma notation.

For example: the value `2000` would become `2,000` using this format string.

```python
# Use Map to format all the columns
file_df["avg_cost"] = file_df["avg_cost"].map("${:.2f}".format)
file_df["population"] =
file_df["population"].map("${:,}".format)
file_df["other"] = file_df["other"].map("${:.2f}".format)
file_df.head()
```

Formatting Syntax

Format mapping only really works once and will return errors if the same code is run multiple times without restarting the kernel. Because of this, formatting is usually applied near the end of an application.

```
# Mapping has changed the datatypes of the columns to strings
file_df.dtypes
```

```
id                  int64
city                object
avg_cost            object
population          object
other               object
dtype: object       object
```

Format mapping also can change the datatype of a column. As such, all calculations should be handled before modifying the formatting.

Instructor Demonstration

Formatting and Mapping

Questions?

Activity: Formatting and Mapping

In this activity, you will read sales data into a DataFrame and are asked to format the columns that are `int64` or `float64` data types with comma notation, a dollar sign, and to two decimal places.

Suggested Time:

15 minutes