Introduction to JSON

STREAMLINED DATA INGESTION WITH PANDAS

Amany Mahfouz
Instructor

Javascript Object Notation (JSON)

- Common web data format
- Not tabular
  - Records don't have to all have the same set of attributes
- Data organized into collections of objects
- Objects are collections of attribute-value pairs
- Nested JSON: objects within objects

Reading JSON Data

- `read_json()`
  - Takes a string path to JSON *or* JSON data as a string
  - Specify data types with `dtype` keyword argument
  - `orient` keyword argument to flag uncommon JSON data layouts
    - possible values in `pandas` documentation

Data Orientation

- JSON data isn't tabular
  - `pandas` guesses how to arrange it in a table
- `pandas` can automatically handle common orientations

Record Orientation

- Most common JSON arrangement


```
[
  {
      "age_adjusted_death_rate": "7.6",
      "death_rate": "6.2",
      "deaths": "32",
      "leading_cause": "Accidents Except Drug Posioning (V01-X39, X43, X45-X59, Y85-Y86)",
      "race_ethnicity": "Asian and Pacific Islander",
      "sex": "F",
      "year": "2007"
  },
  {
      "age_adjusted_death_rate": "8.1",
      "death_rate": "8.3",
      "deaths": "87",
      ...
```

Column Orientation

- More space-efficient than record-oriented JSON

```
{
  "age_adjusted_death_rate": {
      "0": "7.6",
      "1": "8.1",
      "2": "7.1",
      "3": ".",
      "4": ".",
      "5": "7.3",
      "6": "13",
      "7": "20.6",
      "8": "17.4",
      "9": ".",
      "10": ".",
      "11": "19.8",
      ...
```

Specifying Orientation

- Split oriented data - `nyc_death_causes.json`


```
{
  "columns": [
      "age_adjusted_death_rate",
      "death_rate",
      "deaths",
      "leading_cause",
      "race_ethnicity",
      "sex",
      "year"
  ],
  "index": [...],
  "data": [
      [
          "7.6",
```

Specifying Orientation

```
import pandas as pd

death_causes = pd.read_json("nyc_death_causes.json",
                            orient="split")

print(death_causes.head())
```

```
  age_adjusted_death_rate death_rate deaths             leading_cause             race_ethnicity se
0                     7.6        6.2     32  Accidents Except Drug...  Asian and Pacific Islander   F
1                     8.1        8.3     87  Accidents Except Drug...          Black Non-Hispanic   F
2                     7.1        6.1     71  Accidents Except Drug...                    Hispanic   F
3                       .          .      .  Accidents Except Drug...          Not Stated/Unknown   F
4                       .          .      .  Accidents Except Drug...       Other Race/ Ethnicity   F

[5 rows x 7 columns]
```

Let's practice!

STREAMLINED DATA INGESTION WITH PANDAS

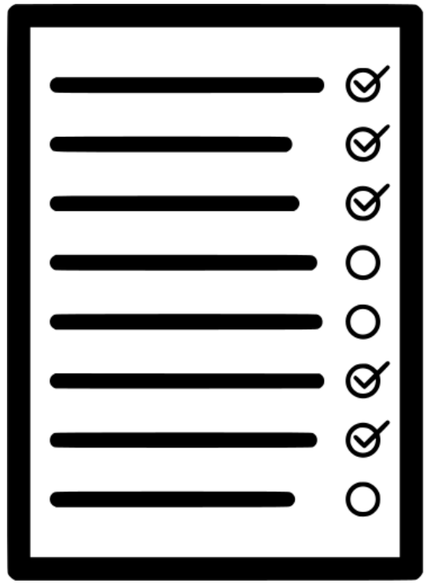
Introduction to APIs

STREAMLINED DATA INGESTION WITH PANDAS

Amany Mahfouz
Instructor

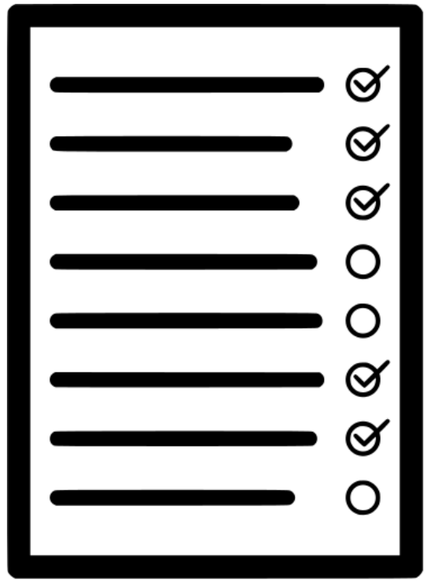
Application Programming Interfaces

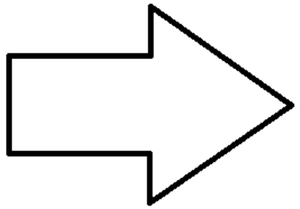
- Defines how a application communicates with other programs
- Way to get data from an application without knowing database details



Application Programming Interfaces

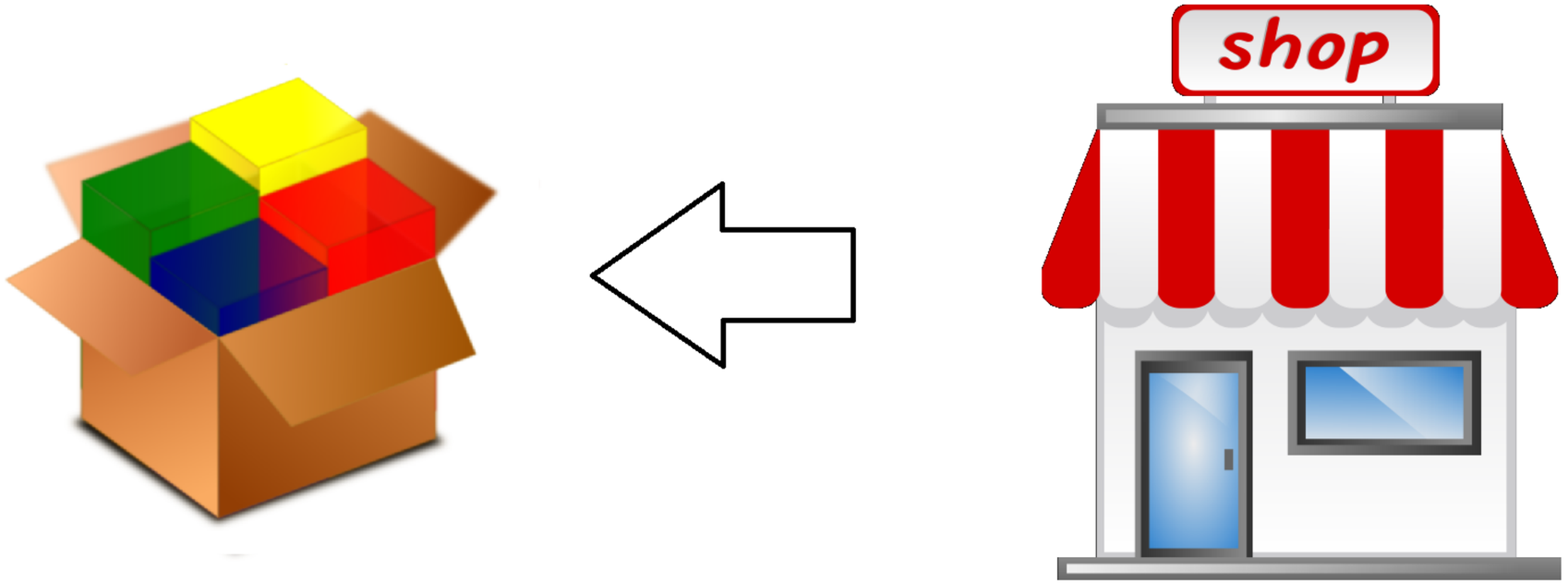
- Defines how a application communicates with other programs
- Way to get data from an application without knowing database details





Application Programming Interfaces

- Defines how a application communicates with other programs
- Way to get data from an application without knowing database details



Requests

- Send and get data from websites
- Not tied to a particular API
- `requests.get()` to get data from a URL

requests.get()

- `requests.get(url_string)` to get data from a URL
- Keyword arguments
  - `params` keyword: takes a dictionary of parameters and values to customize API request
  - `headers` keyword: takes a dictionary, can be used to provide user authentication to API
- Result: a `response` object, containing data and metadata
  - `response.json()` will return just the JSON data

response.json() and pandas

- `response.json()` returns a dictionary
- `read_json()` expects strings, not dictionaries
- Load the response JSON to a data frame with `pd.DataFrame()`
  - `read_json()` will give an error!

Yelp Business Search API

Yelp Business Search API

Request

```
GET https://api.yelp.com/v3/businesses/search
```

Parameters

These parameters should be in the query string.

Name	Type	Description
term	string	Optional. Search term, for example "food" or "restaurants". The term may also be business names, such as "Starbucks". If term is not included the endpoint will default to searching across businesses from a small number of popular categories.
location	string	Required if either latitude or longitude is not provided. This string indicates the geographic area to be used when searching for businesses. Examples: "New York City", "NYC", "350 5th Ave, New York, NY 10118". Businesses returned in the response may not be strictly within the specified location.
latitude	decimal	Required if location is not provided. Latitude of the location you want to search nearby.

Yelp Business Search API

Request

```
GET https://api.yelp.com/v3/businesses/search
```

Parameters

These parameters should be in the query string.

Name	Type	Description
term	string	Optional. Search term, for example "food" or "restaurants". The term may also be business names, such as "Starbucks". If term is not included the endpoint will default to searching across businesses from a small number of popular categories.
location	string	Required if either latitude or longitude is not provided. This string indicates the geographic area to be used when searching for businesses. Examples: "New York City", "NYC", "350 5th Ave, New York, NY 10118". Businesses returned in the response may not be strictly within the specified location.
latitude	decimal	Required if location is not provided. Latitude of the location you want to search nearby.

Yelp Business Search API

Request

```
GET https://api.yelp.com/v3/businesses/search
```

Parameters

These parameters should be in the query string.

Name	Type	Description
term	string	Optional. Search term, for example "food" or "restaurants". The term may also be business names, such as "Starbucks". If term is not included the endpoint will default to searching across businesses from a small number of popular categories.
location	string	Required if either latitude or longitude is not provided. This string indicates the geographic area to be used when searching for businesses. Examples: "New York City", "NYC", "350 5th Ave, New York, NY 10118". Businesses returned in the response may not be strictly within the specified location.
latitude	decimal	Required if location is not provided. Latitude of the location you want to search nearby.

Yelp Business Search API

Request

```
GET https://api.yelp.com/v3/businesses/search
```

Parameters

These parameters should be in the query string.

Name	Type	Description
term	string	Optional. Search term, for example "food" or "restaurants". The term may also be business names, such as "Starbucks". If term is not included the endpoint will default to searching across businesses from a small number of popular categories.
location	string	Required if either latitude or longitude is not provided. This string indicates the geographic area to be used when searching for businesses. Examples: "New York City", "NYC", "350 5th Ave, New York, NY 10118". Businesses returned in the response may not be strictly within the specified location.
latitude	decimal	Required if location is not provided. Latitude of the location you want to search nearby.

Yelp Business Search API

Response Body


```
{
  "total": 8228,
  "businesses": [
    {
      "rating": 4,
      "price": "$",
      "phone": "+14152520800",
      "id": "E8RJkjfdcwgtyoPMjQ_Olg",
      "alias": "four-barrel-coffee-san-francisco",
      "is_closed": false,
      "categories": [
        {
          "alias": "coffee",
          "title": "Coffee & Tea"
        }
      ],
      "review_count": 1738,
      "name": "Four Barrel Coffee",
      "url": "https://www.yelp.com/biz/four-barrel-coffee-san-francisco",
```

Making Requests

```
import requests
import pandas as pd

api_url = "https://api.yelp.com/v3/businesses/search"

# Set up parameter dictionary according to documentation
params = {"term": "bookstore",
          "location": "San Francisco"}

# Set up header dictionary w/ API key according to documentation
headers = {"Authorization": "Bearer {}".format(api_key)}

# Call the API
response = requests.get(api_url,
                        params=params,
                        headers=headers)
```

Parsing Responses

```
# Isolate the JSON data from the response object
data = response.json()
print(data)
```

```
{'businesses': [{'id': '_rbF2ooLcMRA7Kh8neIr4g', 'alias': 'city-lights-bookstore-san-francisco', 'nar
```

```
# Load businesses data to a data frame
bookstores = pd.DataFrame(data["businesses"])
print(bookstores.head(2))
```

```
                              alias        ...
0   city-lights-bookstore-san-francisco        ...      https://www.yelp.com/biz/city-lights-bookstor
1  alexander-book-company-san-francisco        ...      https://www.yelp.com/biz/alexander-book-compa

[2 rows x 16 columns]
```

Let's practice!

STREAMLINED DATA INGESTION WITH PANDAS

Working with nested JSONs

STREAMLINED DATA INGESTION WITH PANDAS

Amany Mahfouz
Instructor

Nested JSONs

- JSONs contain objects with attribute-value pairs
- A JSON is nested when the value itself is an object

Response Body


```
{
  "total": 8228,
  "businesses": [
    {
      "rating": 4,
      "price": "$",
      "phone": "+14152520800",
      "id": "E8RJkjfdcwgtyoPMjQ_Olg",
      "alias": "four-barrel-coffee-san-francisco",
      "is_closed": false,
      "categories": [
        {
          "alias": "coffee",
          "title": "Coffee & Tea"
        }
      ],
      "review_count": 1738,
      "name": "Four Barrel Coffee",
      "url": "https://www.yelp.com/biz/four-barrel-coffee-san-francisco",
      "coordinates": {
        "latitude": 37.7670169511878,
        "longitude": -122.42184275
      },
      "image_url": "http://s3-media2.fl.yelpcdn.com/bphoto/MmgtASP3l_t4tPCL1iAsCg/o.jpg",
      "location": {
        "city": "San Francisco",
        "country": "US",
```

Response Body


```
{
  "total": 8228,
  "businesses": [
    {
      "rating": 4,
      "price": "$",
      "phone": "+14152520800",
      "id": "E8RJkjfdcwgtyoPMjQ_Olg",
      "alias": "four-barrel-coffee-san-francisco",
      "is_closed": false,
      "categories": [
        {
          "alias": "coffee",
          "title": "Coffee & Tea"
        }
      ],
      "review_count": 1738,
      "name": "Four Barrel Coffee",
      "url": "https://www.yelp.com/biz/four-barrel-coffee-san-francisco",
      "coordinates": {
        "latitude": 37.7670169511878,
        "longitude": -122.42184275
      },
      "image_url": "http://s3-media2.fl.yelpcdn.com/bphoto/MmgtASP3l_t4tPCL1iAsCg/o.jpg",
      "location": {
        "city": "San Francisco",
        "country": "US",
```

Response Body


```
{
  "total": 8228,
  "businesses": [
    {
      "rating": 4,
      "price": "$",
      "phone": "+14152520800",
      "id": "E8RJkjfdcwgtyoPMjQ_Olg",
      "alias": "four-barrel-coffee-san-francisco",
      "is_closed": false,
      "categories": [
        {
          "alias": "coffee",
          "title": "Coffee & Tea"
        }
      ],
      "review_count": 1738,
      "name": "Four Barrel Coffee",
      "url": "https://www.yelp.com/biz/four-barrel-coffee-san-francisco",
      "coordinates": {
        "latitude": 37.7670169511878,
        "longitude": -122.42184275
      },
      "image_url": "http://s3-media2.fl.yelpcdn.com/bphoto/MmgtASP3l_t4tPCL1iAsCg/o.jpg",
      "location": {
        "city": "San Francisco",
        "country": "US",
```

Response Body


```
{
  "total": 8228,
  "businesses": [
    {
      "rating": 4,
      "price": "$",
      "phone": "+14152520800",
      "id": "E8RJkjfdcwgtyoPMjQ_Olg",
      "alias": "four-barrel-coffee-san-francisco",
      "is_closed": false,
      "categories": [
        {
          "alias": "coffee",
          "title": "Coffee & Tea"
        }
      ],
      "review_count": 1738,
      "name": "Four Barrel Coffee",
      "url": "https://www.yelp.com/biz/four-barrel-coffee-san-francisco",
      "coordinates": {
        "latitude": 37.7670169511878,
        "longitude": -122.42184275
      },
      "image_url": "http://s3-media2.fl.yelpcdn.com/bphoto/MmgtASP3l_t4tPCL1iAsCg/o.jpg",
      "location": {
        "city": "San Francisco",
        "country": "US",
```

```
# Print columns containing nested data
print(bookstores[["categories", "coordinates", "location"]].head(3))
```

```
                                          categories  \
0   [{'alias': 'bookstores', 'title': 'Bookstores'}]
1  [{'alias': 'bookstores', 'title': 'Bookstores'...
2   [{'alias': 'bookstores', 'title': 'Bookstores'}]

                                         coordinates  \
0  {'latitude': 37.7975997924805, 'longitude': -1...
1  {'latitude': 37.7885846793652, 'longitude': -1...
2  {'latitude': 37.7589836120605, 'longitude': -1...

                                            location
0  {'address1': '261 Columbus Ave', 'address2': '...
1  {'address1': '50 2nd St', 'address2': '', 'add...
2  {'address1': '866 Valencia St', 'address2': ''...
```

pandas.io.json

- `pandas.io.json` submodule has tools for reading and writing JSON
  - Needs its own `import` statement
- `json_normalize()`
  - Takes a dictionary/list of dictionaries (like `pd.DataFrame()` does)
  - Returns a flattened data frame
  - Default flattened column name pattern: `attribute.nestedattribute`
  - Choose a different separator with the `sep` argument

Loading Nested JSON Data

```
import pandas as pd
import requests
from pandas.io.json import json_normalize

# Set up headers, parameters, and API endpoint
api_url = "https://api.yelp.com/v3/businesses/search"
headers = {"Authorization": "Bearer {}".format(api_key)}
params = {"term": "bookstore",
          "location": "San Francisco"}

# Make the API call and extract the JSON data
response = requests.get(api_url,
                        headers=headers,
                        params=params)
data = response.json()
```

```
# Flatten data and load to data frame, with _ separators
bookstores = json_normalize(data["businesses"], sep="_")
print(list(bookstores))
```

```
['alias',
 'categories',
 'coordinates_latitude',
 'coordinates_longitude',
 ...
 'location_address1',
 'location_address2',
 'location_address3',
 'location_city',
 'location_country',
 'location_display_address',
 'location_state',
 'location_zip_code',
 ...
 'url']
```

Deeply Nested Data

```
print(bookstores.categories.head())
```

```
0     [{'alias': 'bookstores', 'title': 'Bookstores'}]
1    [{'alias': 'bookstores', 'title': 'Bookstores'...
2     [{'alias': 'bookstores', 'title': 'Bookstores'}]
3     [{'alias': 'bookstores', 'title': 'Bookstores'}]
4    [{'alias': 'bookstores', 'title': 'Bookstores'...
Name: categories, dtype: object
```

Deeply Nested Data

- `json_normalize()`
  - `record_path` : string/list of string attributes to nested data
  - `meta` : list of other attributes to load to data frame
  - `meta_prefix` : string to prefix to meta column names

Deeply Nested Data

```
# Flatten categories data, bring in business details
df = json_normalize(data["businesses"],
                    sep="_",
                    record_path="categories",
                    meta=["name",
                          "alias",
                          "rating",
                          ["coordinates", "latitude"],
                          ["coordinates", "longitude"]],
                    meta_prefix="biz_")
```

```
print(df.head(4))
```

```
        alias                title                biz_name  \
0  bookstores           Bookstores   City Lights Bookstore
1  bookstores           Bookstores  Alexander Book Company
2  stationery  Cards & Stationery  Alexander Book Company
3  bookstores           Bookstores       Borderlands Books

                          biz_alias  biz_rating  biz_coordinates_latitude  \
0   city-lights-bookstore-san-francisco         4.5                 37.797600
1  alexander-book-company-san-francisco         4.5                 37.788585
2  alexander-book-company-san-francisco         4.5                 37.788585
3       borderlands-books-san-francisco         5.0                 37.758984

   biz_coordinates_longitude
0                -122.406578
1                -122.400631
2                -122.400631
3                -122.421638
```

Let's practice!

STREAMLINED DATA INGESTION WITH PANDAS

Combining multiple datasets

STREAMLINED DATA INGESTION WITH PANDAS

Amany Mahfouz
Instructor

Appending

- Use case: adding rows from one data frame to another
- `append()`
  - Data frame method
  - Syntax: `df1.append(df2)`
  - Set `ignore_index` to `True` to renumber rows

Appending

```
# Get first 20 bookstore results
params = {"term": "bookstore",
          "location": "San Francisco"}
first_results = requests.get(api_url,
                             headers=headers
                             params=params).json()

first_20_bookstores = json_normalize(first_results["businesses"],
                                     sep="_")

print(first_20_bookstores.shape)
```

```
(20, 24)
```

```
# Get the next 20 bookstores
params["offset"] = 20
next_results = requests.get(api_url,
                            headers=headers
                            params=params).json()

next_20_bookstores = json_normalize(next_results["businesses"],
                                    sep="_")

print(next_20_bookstores.shape)
```

```
(20, 24)
```

```
# Put bookstore datasets together, renumber rows
bookstores = first_20_bookstores.append(next_20_bookstores,
                                        ignore_index=True)

print(bookstores.name)
```

```
0                              City Lights Bookstore
1                           Alexander Book Company
2                                Borderlands Books
3                                  Alley Cat Books
4                                  Dog Eared Books
...                                            ...
35                                  Forest Books
36               San Francisco Center For The Book
37                          KingSpoke - Book Store
38                           Eastwind Books & Arts
39                                     My Favorite
Name: name, dtype: object
```

Merging

- Use case: combining datasets to add related columns
- Datasets have key column(s) with common values
- `merge()` : `pandas` version of a SQL join

Merging

- `merge()`
  - Both a `pandas` function and a data frame method
- `df.merge()` arguments
  - Second data frame to merge
  - Columns to merge on
    - `on` if names are the same in both data frames
    - `left_on` and `right_on` if key names differ
    - Key columns should be the same data type

```
call_counts.head()
```

```
  created_date  call_counts
0   01/01/2018         4597
1   01/02/2018         4362
2   01/03/2018         3045
3   01/04/2018         3374
4   01/05/2018         4333
```

```
weather.head()
```

```
         date    tmax  tmin
0  12/01/2017      52    42
1  12/02/2017      48    39
2  12/03/2017      48    42
3  12/04/2017      51    40
4  12/05/2017      61    50
```

Merging

```
# Merge weather into call counts on date columns
merged = call_counts.merge(weather,
                           left_on="created_date",
                           right_on="date")

print(merged.head())
```

```
  created_date  call_counts        date  tmax  tmin
0   01/01/2018         4597  01/01/2018    19     7
1   01/02/2018         4362  01/02/2018    26    13
2   01/03/2018         3045  01/03/2018    30    16
3   01/04/2018         3374  01/04/2018    29    19
4   01/05/2018         4333  01/05/2018    19     9
```

Merging

```
   created_date  call_counts        date  tmax  tmin
0    01/01/2018         4597  01/01/2018    19     7
1    01/02/2018         4362  01/02/2018    26    13
2    01/03/2018         3045  01/03/2018    30    16
3    01/04/2018         3374  01/04/2018    29    19
4    01/05/2018         4333  01/05/2018    19     9
```

- Default `merge()` behavior: return only values that are in both datasets
- One record for each value match between data frames
  - Multiple matches = multiple records

Let's practice!

STREAMLINED DATA INGESTION WITH PANDAS

Wrap-up

STREAMLINED DATA INGESTION WITH PANDAS

Amany Mahfouz
Instructor

Recap

Chapters 1 and 2

- `read_csv()` and `read_excel()`
- Setting data types, choosing data to load, handling missing data and errors

Recap

Chapter 3

- `read_sql()` and `sqlalchemy`
- SQL `SELECT`, `WHERE`, aggregate functions and joins

Recap

Chapter 4

- `read_json()`, `json_normalize()`, and `requests`
- Working with APIs and nested JSONs
- Appending and merging datasets

Where to next?

- Learn more about data wrangling in `pandas`
  - Working with indexes, transforming values, dropping rows and columns
  - Reshaping data by merging, melting, pivoting
  - **Data Manipulation with Python Skill Track**

Where to next?

- Explore a variety of analysis topics
  - Descriptive statistics (e.g. medians, means, standard deviation)
  - Inferential statistics (hypothesis testing, correlation, regression)
  - **Exploratory Data Analysis in Python**
  - **Introduction to Linear Modeling in Python**

Where to next?

- Learn data visualization techniques
  - `seaborn` and `matplotlib` libraries
  - **Introduction to Data Visualization with Python**
  - **Introduction to Matplotlib**

Where to next?

- Wrangle data as part of a fuller data science workflow
  - **Analyzing Police Activity with pandas**
  - **Analyzing US Census Data in Python**
  - **Analyzing Social Media Data in Python**

Congrats!

STREAMLINED DATA INGESTION WITH PANDAS