Pandas

Thomas Schwarz, SJ

Obtaining Data

- Create data frames:
  - From a python or np object
  - From a file:
    - pd.read_csv('name_of_my_file')
    - pd.read_json()
  - From a database (e.g. sqlite which comes with python)
  - From an API

Obtaining Data

- Download 'earthquakes.csv' and open it up
  - `earthquakes = pd.read_csv('earthquakes.csv')`
- Download the quakes database
  - ```
    with sqlite3.connect('quakes.db') as connection:
        tsunamis = pd.read_sql('SELECT * FROM tsunamis',
                               connection)
    ```
    ```

APIs

- Websites offer APIs to interact with them
  - Better for them then web-scraping
  - Go to https://earthquake.usgs.gov/fdsnws/event/1/
- To make requests, we need the requests module in Python
  - When we make the request, we want to ensure that the website delivered the data
    - Look for a status code of 200

APIs

```
api = 'https://earthquake.usgs.gov/fdsnws/event/1/
query?
format=geojson&starttime=2022-01-01&endtime=2022-02-
01&minmagnitude=5'
response = requests.get(api)
print(response.status_code)
```

- Now we need to extract the json data

```
earthquake_json = response.json()
```

APIs

- We need to understand the json data
  - earthquake_json.keys( ) gives us the json keys
    - Look for metadata and features
    - Features looks good, but we just look at the first element

```
earthquake_json['features'][4]
{'type': 'Feature', 'properties': {'mag': 5.4, 'place': 'Kermadec Islands region',
'time': 1643628412598, 'updated': 1645261166008, 'tz': None, 'url': 'https://
earthquake.usgs.gov/earthquakes/eventpage/us7000ggl1', 'detail': 'https://
earthquake.usgs.gov/fdsnws/event/1/query?eventid=us7000ggl1&format=geojson', 'felt':
None, 'cdi': None, 'mmi': 2.885, 'alert': 'green', 'status': 'reviewed', 'tsunami': 0,
'sig': 449, 'net': 'us', 'code': '7000ggl1', 'ids': ',us7000ggl1,', 'sources': ',us,',
'types': ',losspager,origin,phase-data,shakemap,', 'nst': None, 'dmin': 0.942, 'rms':
0.81, 'gap': 51, 'magType': 'mb', 'type': 'earthquake', 'title': 'M 5.4 - Kermadec
Islands region'}, 'geometry': {'type': 'Point', 'coordinates': [-176.9102, -29.5983,
10]}, 'id': 'us7000ggl1'}
```

APIs

- This is a dictionary
  - The interesting part is in properties, so we extract all those as a list

```
earthquake_properties = [ quake['properties'] for
                quake in earthquake_json['features'] ]
```

  - Now we convert this into a dataframe
  - `df = pd.DataFrame(earthquake_properties)`

Data Frame Inspection

- Inspect the data frame
  - Shape: df.shape
  - Column names df.columns
  - Beginning: df.head( ), df.tail( )
  - Description: df.describe( )

Data Frame Inspection

- We can count values:
  - `df.alert.value_counts( )`
  - Find min / max:
  - `df.mag.max()`
  - and much more

Indexing

- Selecting columns:
  - Use brackets:
    - `df['mag']`
  - Use the name (as long as there is no clash)
    - `df.mag`
  - Use the first one for more than one column

```
df[['mag', 'alert', 'title']]
```

Indexing

- Selecting rows:
  - We can select rows by slicing
    - `df[35:39]`
  - We can combine row and column selection
    - `df[['alert', 'time', 'mag']][35:39]`

Indexing

- Pandas indexes with two methods
  - `loc[ ]` and `iloc[ ]`
  - Notice the square brackets:
    - “loc” for location, “iloc” for integer location

Indexing

- Examples:
  - Selecting a column
    - `df.loc[:,'mag']`
    - `df.loc[:,['mag', 'title']`
    - `df.loc[10:15,['mag', 'title']]`

Indexing

- Examples:
  - Selecting a column by index
    - `df.iloc[10:15,[5,6,7]]`
    - `df.iloc[20:25, 6:10]`

Filtering

- Pandas (like numpy) uses boolean masks
  - ```
    df.loc[df.mag>6,
           ['alert', 'mag', 'magType', 'title']
          ]
    ```
  - ```
    df.loc[(df.tsunami==1) & (df.alert == 'green'),
           ['alert', 'mag', 'type', 'tsunami']
          ]
    ```

- Recall that you need the logical operators & | ~

Filtering

- Example using functions:
  - `df.loc[ df.title.str.contains('Alaska'), ]`

Adding and removing data

- We create a new column by defining it
  - Broadcasting instance: Value is applied to all rows
    - `df['source'] = 'USGS API'`
  - We create a sub-data-frame by boolean condition
    - ```
      tsunami = df[df.tsunami==1]
      nontsunami = df[df.tsunami==0]
      tsunami.shape
      nontsunami.shape
      ```
      ```

Adding and Removing Data

- We can use append or concatenate to combine data frames
  - ```
    pd.concat([tsunami, nontsunami]).shape
    ```

```
tsunami.append(nontsunami).shape
```
```

Adding and Removing Data

- To remove a column, we can just use del
  - `del df['source']`

Evaluating Pandas Data Frames

- Use groupby to combine rows with the same values and then use summary statistics
    - Example: Mean of magnitudes of earthquakes that month depending on whether they created a Tsunami

```
df.groupby('tsunami')['mag'].mean()

tsunami
0    5.326282
1    5.870000
```

Combining Pandas Data Frames

- We can use the join function to combine different data
  - Needs to have a common index
  - Example:
    - Create a data-frame from a dictionary

```
df1 = pd.DataFrame( {'A': [1,2,3], 'B': [0,5,10]})
```

- This gives

```
df1
   A   B
0  1   0
1  2   5
2  3  10
```

Combining Pandas Data Frames

- Example (continued)
  - Create a second data-frame

```
df2 = pd.DataFrame( {'A': [1,2,4,], 'C': [11, 13, 15] })
```

```
df2
   A   C
0  1  11
1  2  13
2  4  15
```

Combining Pandas Data Frames

- Reset the indices to column A
  - Use `inplace=True` to make changes to the data-frames

```
df1.set_index('A', inplace = True)


df1
    B
A
1   0
2   5
3  10
```

Combining Pandas Data Frames

- Do the same to the other data-frame
- Now we combine
  - "outer" means that we combine all rows, even those that do not match

```
df1.join(df2, how='outer')
```

```
     B     C
A
1   0.0  11.0
2   5.0  13.0
3  10.0   NaN
4   NaN  15.0
```