Data Analysis with Python

Intro to Pandas
DataFrames

DataFrame: most important data structure of pandas

- It is a tabular structure tightly integrated with Series
- Use analysis of G7 countries and Series as reference

G7 Stats					
	Population	GDP	Surface	HDI	Continent
Canada	35.467	1,785,387.00	9,984,670	0.913	America
France	63.951	2,833,687.00	640,679	0.888	Europe
Germany	80.94	3,874,437.00	357,114	0.916	Europe
Italy	60.665	2,167,744.00	301,336	0.873	Europe
Japan	127.061	4,602,367.00	377,930	0.891	Asia
United Kingdom	64.511	2,950,039.00	242,495	0.907	Europe
United States	318.523	17,348,075.00	9,525,067	0.915	America

- A DataFrame looks alot like a table
- Creating DataFrames manually can be tedious
- 99% of the time you'll be pulling Data from a Database, a csv file or the web
- You cans till create a DataFrame by specifying the columns and values

```
In [2]:
df = pd.DataFrame({
    'Population': [35.467, 63.951, 80.94, 60.665, 127.061, 64.511, 318.523],
    'GDP': [
        1785387,
        2833687,
        3874437,
        2167744,
        4602367,
        2950039,
        17348075
    ],
    'Surface Area': [
        9984670,
        640679,
        357114,
        301336,
        377930,
        242495,
        9525067
    ],
    'HDI': [
        0.913,
        0.888,
        0.916,
        0.873,
        0.891,
        0.907,
        0.915
    ],
    'Continent': [
        'America',
        'Europe',
        'Europe',
        'Europe',
        'Asia',
        'Europe',
        'America'
    ]
}, columns=['Population', 'GDP', 'Surface Area', 'HDI', 'Continent'])
```

- The *columns* attribute is optional; I just wanted to match the table above

```
In [3]: df
```

	Population	GDP	Surface Area	HDI	Continent
0	35.467	1785387	9984670	0.913	America
1	63.951	2833687	640679	0.888	Europe
2	80.940	3874437	357114	0.916	Europe
3	60.665	2167744	301336	0.873	Europe
4	127.061	4602367	377930	0.891	Asia
5	64.511	2950039	242495	0.907	Europe
6	318.523	17348075	9525067	0.915	America

DataFrames also have indexes.

- As seen in the table above, pandas has assigned a numeric, auto-incremental index to each row in the DataFrame
- In this case, each row represents a country, so will reassign the index

```
In [4]: df.index = [
            'Canada',
            'France',
            'Germany',
            'Italy',
            'Japan',
            'UK',
            'US'
        ]
```

```
df.index = [
        ‘Canada’,
        ‘France’,
        ‘Germany’,
        ‘Italy’,
        ‘Japan’,
        ‘UK’,
        ‘US’
]
```

```
In [5]: df
```

	Population	GDP	Surface Area	HDI	Continent
Canada	35.467	1785387	9984670	0.913	America
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe
Italy	60.665	2167744	301336	0.873	Europe
Japan	127.061	4602367	377930	0.891	Asia
UK	64.511	2950039	242495	0.907	Europe
US	318.523	17348075	9525067	0.915	America

df

- Notice: the indices within the table changed from basic integer indices to the names of G7 countries

```
In [6]: df.index

Index(['Canada', 'France', 'Germany', 'Italy', 'Japan', 'UK', 'US'], dtype='object')
```

df.index

```
In [7]: df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 7 entries, Canada to US
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Population    7 non-null      float64
 1   GDP           7 non-null      int64
 2   Surface Area  7 non-null      int64
 3   HDI           7 non-null      float64
 4   Continent     7 non-null      object
dtypes: float64(2), int64(2), object(1)
memory usage: 280.0+ bytes
```

df.info()

```
In [8]: df.size

35
```

df.size

```
In [10]: df.shape

(7, 5)
```

df.shape

- (rows, cols)

```
In [11]: df.describe()
```

	Population	GDP	Surface Area	HDI
count	7.000000	7.000000e+00	7.000000e+00	7.000000
mean	107.302571	5.080248e+06	3.061327e+06	0.900429
std	97.249970	5.494020e+06	4.576187e+06	0.016592
min	35.467000	1.785387e+06	2.424950e+05	0.873000
25%	62.308000	2.500716e+06	3.292250e+05	0.889500
50%	64.511000	2.950039e+06	3.779300e+05	0.907000
75%	104.000500	4.238402e+06	5.082873e+06	0.914000
max	318.523000	1.734808e+07	9.984670e+06	0.916000

df.describe()

```
In [12]: df.dtypes

Population      float64
GDP               int64
Surface Area      int64
HDI             float64
Continent        object
dtype: object
```

df.dtypes

```
In [13]: df.dtypes.value_counts()

float64    2
int64      2
object     1
dtype: int64
```

df.dtypes.value_counts()

Indexing, Selection and Slicing

Individual columns in the DataFrame can be selected with regular indexing. Each column is represented as a Series.

```
In [14]: # Indexing, Selection and Slicing
         df
```

	Population	GDP	Surface Area	HDI	Continent
Canada	35.467	1785387	9984670	0.913	America
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe
Italy	60.665	2167744	301336	0.873	Europe
Japan	127.061	4602367	377930	0.891	Asia
UK	64.511	2950039	242495	0.907	Europe
US	318.523	17348075	9525067	0.915	America

df

```
In [15]: df.loc['Canada']

Population       35.467
GDP            1785387
Surface Area    9984670
HDI               0.913
Continent       America
Name: Canada, dtype: object
```

df.loc[‘Canada’]

```
In [16]: df.iloc[-1]

Population       318.523
GDP            17348075
Surface Area    9525067
HDI               0.915
Continent       America
Name: US, dtype: object
```

df.iloc[-1]

- Note the difference between *In[15]* and *In[16]*

df.loc[‘...’] vs df.iloc[index/int]

```
In [17]: df['Population']

Canada      35.467
France      63.951
Germany     80.940
Italy       60.665
Japan      127.061
UK          64.511
US         318.523
Name: Population, dtype: float64
```

df[‘Population’]

- Note: The index of the returned Series is the same as the DataFrame one. And its name is the name of the column.

If you are working on a notebook and want to see a more **DataFrame-like format** you can use the *to_frame method*

```
In [18]: df['Population'].to_frame()
```

	Population
Canada	35.467
France	63.951
Germany	80.940
Italy	60.665
Japan	127.061
UK	64.511
US	318.523

df[‘Population’].to_frame()

```
df[['Population', 'GDP']]
```

	Population	GDP
Canada	35.467	1785387
France	63.951	2833687
Germany	80.940	3874437
Italy	60.665	2167744
Japan	127.061	4602367
UK	64.511	2950039
US	318.523	17348075

df[[‘Population’, ‘GDP’]]

- This shows that multiple columns can also be selected similarly to numpy and Series
- In this case, the result is a DataFrame

```
In [20]: df[1:3]
```

	Population	GDP	Surface Area	HDI	Continent
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe

df[1:3]
Slicing works differently. It acts at '**row level**' and can be counter intuitive

- Note: the **upper limit is EXCLUDED**

```
In [21]: df.loc['Italy']

Population      60.665
GDP           2167744
Surface Area   301336
HDI             0.873
Continent      Europe
Name: Italy, dtype: object
```

df.loc['Italy']

Row level selection works better with *loc* and *iloc* which are recommended over regular "direct slicing" (df[:]).

- *loc* selects rows matching the given index

```
In [22]: df.loc['France': 'Italy']
```

	Population	GDP	Surface Area	HDI	Continent
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe
Italy	60.665	2167744	301336	0.873	Europe

df.loc['France': 'Italy']

- Notice the difference between *In[20]* and *In[22]*
    - *In[22]* includes the upper limit while *In[20]* doesn't

```
In [24]: df.loc['France': 'Italy', 'Population']

France     63.951
Germany    80.940
Italy      60.665
Name: Population, dtype: float64
```

df.loc['France': 'Italy', 'Population']

- As a **second argument**, you can pass the **column(s)** you'd like to extract

```
In [25]: df.loc['France': 'Italy', 'Population'].to_frame()
```

	Population
France	63.951
Germany	80.940
Italy	60.665

df.loc['France': 'Italy', 'Population'].to_frame()

- Notice the difference between *In[24]* and *In[25]*

```
In [26]: df.loc['France': 'Italy', ['Population', 'GDP']]
```

	Population	GDP
France	63.951	2833687
Germany	80.940	3874437
Italy	60.665	2167744

df.loc['France': 'Italy', ['Population', 'GDP']]

Generally, if there are multiple columns indexed, then the output will be a dataFrame

- Even without the *.to_frame()*

```
In [29]: df
```

	Population	GDP	Surface Area	HDI	Continent
Canada	35.467	1785387	9984670	0.913	America
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe
Italy	60.665	2167744	301336	0.873	Europe
Japan	127.061	4602367	377930	0.891	Asia
UK	64.511	2950039	242495	0.907	Europe
US	318.523	17348075	9525067	0.915	America

df

```
In [30]: df.iloc[0]

Population       35.467
GDP            1785387
Surface Area    9984670
HDI               0.913
Continent       America
Name: Canada, dtype: object
```

df.iloc[0]

```
In [32]: df.iloc[-1]

Population       318.523
GDP            17348075
Surface Area    9525067
HDI               0.915
Continent       America
Name: US, dtype: object
```

df.iloc[-1]

```
In [33]: df.iloc[[0, 1, -1]]
```

	Population	GDP	Surface Area	HDI	Continent
Canada	35.467	1785387	9984670	0.913	America
France	63.951	2833687	640679	0.888	Europe
US	318.523	17348075	9525067	0.915	America

df.iloc[[0, 1, -1]]

```
In [34]: df.iloc[1:3]
```

	Population	GDP	Surface Area	HDI	Continent
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe

df.iloc[1:3]

- Note: the upper limit index is excluded in the output

```
In [35]: df.iloc[1:3, 3]

France     0.888
Germany    0.916
Name: HDI, dtype: float64
```

df.iloc[1:3, 3]

```
In [36]: df.iloc[1:3, 3].to_frame()
```

	HDI
France	0.888
Germany	0.916

df.iloc[1:3, 3].to_frame()

- Compare the output between *In[35]* and *In[36]*

```
In [37]: df.iloc[1:3, [0,3]]
```

	Population	HDI
France	63.951	0.888
Germany	80.940	0.916

df.iloc[1:3, [0, 3]]

```
In [38]: df.iloc[1:3, 1:3]
```

	GDP	Surface Area
France	2833687	640679
Germany	3874437	357114

df.iloc[1:3, 1:3]

- Note: the upper limit for both the row index slice and column index slice are excluded

NOTE: Always use *iloc* and *loc* to reduce ambiguity, especially for DataFrames with numeric indices

Conditional Selection (boolean arrays)

Recall conditional selection applied to Series. The methods will work the same way for DataFrames.

- This makes sense since a DataFrame is a collection of Series

```
In [39]: df
```

	Population	GDP	Surface Area	HDI	Continent
Canada	35.467	1785387	9984670	0.913	America
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe
Italy	60.665	2167744	301336	0.873	Europe
Japan	127.061	4602367	377930	0.891	Asia
UK	64.511	2950039	242495	0.907	Europe
US	318.523	17348075	9525067	0.915	America

df

```
In [40]: df['Population'] > 70

Canada     False
France     False
Germany     True
Italy      False
Japan       True
UK         False
US          True
Name: Population, dtype: bool
```

```
In [44]: (df['Population'] > 70).to_frame()
```

	Population
Canada	False
France	False
Germany	True
Italy	False
Japan	True
UK	False
US	True

df[‘Population’] > 70

```
In [45]: df.loc[df['Population'] > 70]
```

	Population	GDP	Surface Area	HDI	Continent
Germany	80.940	3874437	357114	0.916	Europe
Japan	127.061	4602367	377930	0.891	Asia
US	318.523	17348075	9525067	0.915	America

df.loc[df['Population'] > 70]

```
In [46]: df.loc[df['Population'] > 70, 'Population']

Germany     80.940
Japan      127.061
US         318.523
Name: Population, dtype: float64

In [48]: df.loc[df['Population'] > 70, 'Population'].to_frame()
```

	Population
Germany	80.940
Japan	127.061
US	318.523

df.loc[df['Population'] > 70, 'Population']

The boolean matching is done at index level, so you can filter by any row, as long as it contains the right indexes.

- Column selection still works as expected

```
In [49]: df.loc[df['Population'] > 70, ['Population', 'GDP']]
```

	Population	GDP
Germany	80.940	3874437
Japan	127.061	4602367
US	318.523	17348075

df.loc[df['Population'] > 70, ['Population', 'GDP']]

Dropping Stuff

Opposed to the concept of selection, we have “dropping”. Instead of pointing out which values you’d like to *select* you could specify which ones you’d like to drop.

```
In [50]: # Dropping Stuff
         df.drop('Canada')
```

	Population	GDP	Surface Area	HDI	Continent
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe
Italy	60.665	2167744	301336	0.873	Europe
Japan	127.061	4602367	377930	0.891	Asia
UK	64.511	2950039	242495	0.907	Europe
US	318.523	17348075	9525067	0.915	America

df.drop(‘Canada’)

```
In [53]: df.drop(['Canada', 'Japan'])
```

	Population	GDP	Surface Area	HDI	Continent
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe
Italy	60.665	2167744	301336	0.873	Europe
UK	64.511	2950039	242495	0.907	Europe
US	318.523	17348075	9525067	0.915	America

df.drop([‘Canada’, ‘Japan’])

- Compare *In[50]* and *In[53]*
  - Notice when you index **multiple rows**, the rows must be within [...]
  - When there’s **one** row, (...) is fine

```
In [54]: df.drop(columns=['Population', 'HDI'])
```

	GDP	Surface Area	Continent
Canada	1785387	9984670	America
France	2833687	640679	Europe
Germany	3874437	357114	Europe
Italy	2167744	301336	Europe
Japan	4602367	377930	Asia
UK	2950039	242495	Europe
US	17348075	9525067	America

df.drop(columns=['Population', 'HDI'])

```
In [57]: df.drop(['Italy', 'Canada'], axis=0)
```

	Population	GDP	Surface Area	HDI	Continent
France	63.951	2833687	640679	0.888	Europe
Germany	80.940	3874437	357114	0.916	Europe
Japan	127.061	4602367	377930	0.891	Asia
UK	64.511	2950039	242495	0.907	Europe
US	318.523	17348075	9525067	0.915	America

df.drop(['Italy', 'Canada'], axis=0)

- drops **rows**

```
In [59]: df.drop(['Population', 'HDI'], axis=1)
```

	GDP	Surface Area	Continent
Canada	1785387	9984670	America
France	2833687	640679	Europe
Germany	3874437	357114	Europe
Italy	2167744	301336	Europe
Japan	4602367	377930	Asia
UK	2950039	242495	Europe
US	17348075	9525067	America

df.drop(['Population', 'HDI'], axis=1)

- Drops **columns**

```
In [60]: df.drop(['Population', 'HDI'], axis='columns')
```

	GDP	Surface Area	Continent
Canada	1785387	9984670	America
France	2833687	640679	Europe
Germany	3874437	357114	Europe
Italy	2167744	301336	Europe
Japan	4602367	377930	Asia
UK	2950039	242495	Europe
US	17348075	9525067	America

df.drop(['Population', 'HDI'], axis='columns')

- This is an equivalent code/ prompt as above (receive same DataFrame output)

```
In [61]: df.drop(['Canada', 'Germany'], axis='rows')
```

	Population	GDP	Surface Area	HDI	Continent
France	63.951	2833687	640679	0.888	Europe
Italy	60.665	2167744	301336	0.873	Europe
Japan	127.061	4602367	377930	0.891	Asia
UK	64.511	2950039	242495	0.907	Europe
US	318.523	17348075	9525067	0.915	America

df.drop(['Canada', 'Germany'], axis='rows')

All these *drop* methods return a **new DataFrame**. If you want to modify it "in place", can use the ***inpace*** attribute.

- Example below

Operations

```
In [62]: df[['Population', 'GDP']]
```

	Population	GDP
Canada	35.467	1785387
France	63.951	2833687
Germany	80.940	3874437
Italy	60.665	2167744
Japan	127.061	4602367
UK	64.511	2950039
US	318.523	17348075

df[[‘Population’, ‘GDP’]]

```
In [63]: df[['Population', 'GDP']] / 100
```

	Population	GDP
Canada	0.35467	17853.87
France	0.63951	28336.87
Germany	0.80940	38744.37
Italy	0.60665	21677.44
Japan	1.27061	46023.67
UK	0.64511	29500.39
US	3.18523	173480.75

df[[‘Population’, ‘GDP’]] / 100

```
In [65]: crisis = pd.Series([-1_000_000, -0.3], index=['GDP', 'HDI'])
         crisis

GDP   -1000000.0
HDI         -0.3
dtype: float64
```

crisis = pd.Series([-1_000_000, -0.3], index=[‘GDP’, ‘HDI’])
crisis

```
In [66]: df[['GDP', 'HDI']]
```

	GDP	HDI
Canada	1785387	0.913
France	2833687	0.888
Germany	3874437	0.916
Italy	2167744	0.873
Japan	4602367	0.891
UK	2950039	0.907
US	17348075	0.915

df[['GDP', 'HDI']]

```
In [67]: df[['GDP', 'HDI']] + crisis
```

	GDP	HDI
Canada	785387.0	0.613
France	1833687.0	0.588
Germany	2874437.0	0.616
Italy	1167744.0	0.573
Japan	3602367.0	0.591
UK	1950039.0	0.607
US	16348075.0	0.615

df[['GDP', 'HDI']] + crisis

- Notice: calculation has been applied to each entry within this DataFrame

Modifying DataFrames

It's simple and intuitive. You can add columns, or replace values for columns without issues.

References

https://www.youtube.com/watch?v=r-uOLxNrNk8

https://github.com/ine-rmotr-curriculum/freecodecamp-intro-to-pandas/blob/master/3%20-%20Pandas%20-%20DataFrames.ipynb

https://docs.google.com/spreadsheets/d/1IlorV2-Oh9Da1JAZ7weVw86PQrQydSMp-ydVMH135il/edit#gid=0