# Pandas - `DataFrame`s

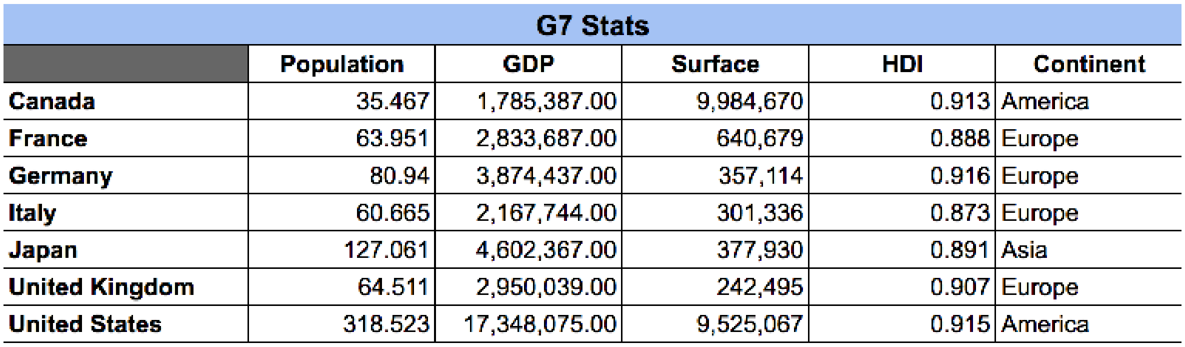
Probably the most important data structure of pandas is the `DataFrame`. It's a tabular structure tightly integrated with `Series`.

## Hands on!

In [147…

```python
import numpy as np
import pandas as pd
```

We'll keep our analysis of G7 countries and looking now at DataFrames. As said, a DataFrame looks a lot like a table (as the one you can appreciate here):



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 |

Creating `DataFrame`s manually can be tedious. 99% of the time you'll be pulling the data from a Database, a csv file or the web. But still, you can create a DataFrame by specifying the columns and values:

In [148…

```python
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'm using it to keep the same order as in the picture above)*

In [149…

```
df
```

Out[149…

| | 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 |

`DataFrame`s also have indexes. As you can see in the "table" above, pandas has assigned a numeric, autoincremental index automatically to each "row" in our DataFrame. In our case, we know that each row represents a country, so we'll just reassign the index:

In [150…

```
df.index = [
    'Canada',
    'France',
    'Germany',
    'Italy',
    'Japan',
    'United Kingdom',
    'United States',
]
```

In [151…

```
df
```

Out[151…

| | 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 |
| **United Kingdom** | 64.511 | 2950039 | 242495 | 0.907 | Europe |
| **United States** | 318.523 | 17348075 | 9525067 | 0.915 | America |

In [152…

```
df.columns
```

Out[152…

```
Index(['Population', 'GDP', 'Surface Area', 'HDI', 'Continent'], dtype='objec
t')
```

In [153…

```
df.index
```

Out[153…

```
Index(['Canada', 'France', 'Germany', 'Italy', 'Japan', 'United Kingdom',
       'United States'],
      dtype='object')
```

In [154…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7 entries, Canada to United States
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: 636.0+ bytes
```

In [155…

```
df.size
```

Out[155…

```
35
```

In [156…

```python
df.shape
```

Out[156…

```
(7, 5)
```

In [157…

```python
df.describe()
```

Out[157…

| | 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 |

In [158…

```python
df.dtypes
```

Out[158…

| | 0 |
|---|---|
| **Population** | float64 |
| **GDP** | int64 |
| **Surface Area** | int64 |
| **HDI** | float64 |
| **Continent** | object |

**dtype:** object

In [159…

```python
df.dtypes.value_counts()
```

Out[159…

| | count |
|---|---|
| **float64** | 2 |
| **int64** | 2 |
| **object** | 1 |

**dtype:** int64

# Indexing, Selection and Slicing

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

In [160…

```
df
```

Out[160…

| | 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 |
| **United Kingdom** | 64.511 | 2950039 | 242495 | 0.907 | Europe |
| **United States** | 318.523 | 17348075 | 9525067 | 0.915 | America |

In [161…

```
df.loc['Canada']
```

Out[161…

| | Canada |
|---|---|
| **Population** | 35.467 |
| **GDP** | 1785387 |
| **Surface Area** | 9984670 |
| **HDI** | 0.913 |
| **Continent** | America |

**dtype:** object

In [162…

```
df.iloc[-1]
```

Out[162…

| | United States |
|---|---|
| **Population** | 318.523 |
| **GDP** | 17348075 |
| **Surface Area** | 9525067 |
| **HDI** | 0.915 |
| **Continent** | America |

**dtype:** object

In [163…

```
df['Population']
```

Out[163…

| | Population |
|---|---|
| **Canada** | 35.467 |
| **France** | 63.951 |
| **Germany** | 80.940 |
| **Italy** | 60.665 |
| **Japan** | 127.061 |
| **United Kingdom** | 64.511 |
| **United States** | 318.523 |

**dtype:** float64

Note that the `index` of the returned Series is the same as the DataFrame one. And its `name` is the name of the column. If you're working on a notebook and want to see a more DataFrame-like format you can use the `to_frame` method:

In [164…

```
df['Population'].to_frame()
```

Out[164…

| | Population |
|---|---|
| **Canada** | 35.467 |
| **France** | 63.951 |
| **Germany** | 80.940 |
| **Italy** | 60.665 |
| **Japan** | 127.061 |
| **United Kingdom** | 64.511 |
| **United States** | 318.523 |

Multiple columns can also be selected similarly to `numpy` and `Series`:

In [165…

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

Out[165…

| | Population | GDP |
|---|---|---|
| **Canada** | 35.467 | 1785387 |
| **France** | 63.951 | 2833687 |
| **Germany** | 80.940 | 3874437 |
| **Italy** | 60.665 | 2167744 |
| **Japan** | 127.061 | 4602367 |
| **United Kingdom** | 64.511 | 2950039 |
| **United States** | 318.523 | 17348075 |

In this case, the result is another `DataFrame`. Slicing works differently, it acts at "row level", and can be counter intuitive:

In [166…

```
df[1:3]
```

Out[166…

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

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 [167…

```
df.loc['Italy']
```

Out[167…

| | Italy |
|---|---|
| **Population** | 60.665 |
| **GDP** | 2167744 |
| **Surface Area** | 301336 |
| **HDI** | 0.873 |
| **Continent** | Europe |

**dtype:** object

In [168…

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

Out[168…

| | 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 |

As a second "argument", you can pass the column(s) you'd like to select:

In [169…

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

Out[169…

| | Population |
|---|---|
| **France** | 63.951 |
| **Germany** | 80.940 |
| **Italy** | 60.665 |

**dtype:** float64

In [170…

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

Out[170…

| | Population | GDP |
|---|---|---|
| **France** | 63.951 | 2833687 |
| **Germany** | 80.940 | 3874437 |
| **Italy** | 60.665 | 2167744 |

`iloc` works with the (numeric) "position" of the index:

In [171…

```
df
```

Out[171…

| | 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 |
| **United Kingdom** | 64.511 | 2950039 | 242495 | 0.907 | Europe |
| **United States** | 318.523 | 17348075 | 9525067 | 0.915 | America |

In [172…

```
df.iloc[0]
```

Out[172…

| | Canada |
|---|---|
| **Population** | 35.467 |
| **GDP** | 1785387 |
| **Surface Area** | 9984670 |
| **HDI** | 0.913 |
| **Continent** | America |

**dtype:** object

In [173…

```
df.iloc[-1]
```

Out[173…

| | United States |
|---|---|
| **Population** | 318.523 |
| **GDP** | 17348075 |
| **Surface Area** | 9525067 |
| **HDI** | 0.915 |
| **Continent** | America |

**dtype:** object

In [174…

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

Out[174…

|  | Population | GDP | Surface Area | HDI | Continent |
|---|---|---|---|---|---|
| **Canada** | 35.467 | 1785387 | 9984670 | 0.913 | America |
| **France** | 63.951 | 2833687 | 640679 | 0.888 | Europe |
| **United States** | 318.523 | 17348075 | 9525067 | 0.915 | America |

In [175…

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

Out[175…

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

In [176…

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

Out[176…

|  | HDI |
|---|---|
| **France** | 0.888 |
| **Germany** | 0.916 |

**dtype:** float64

In [177…

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

Out[177…

|  | Population | HDI |
|---|---|---|
| **France** | 63.951 | 0.888 |
| **Germany** | 80.940 | 0.916 |

In [178…

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

Out[178…

|  | GDP | Surface Area |
|---|---|---|
| **France** | 2833687 | 640679 |
| **Germany** | 3874437 | 357114 |

> **RECOMMENDED: Always use `loc` and `iloc` to reduce ambiguity, specially with `DataFrame`s with numeric indexes.**

# Conditional selection (boolean arrays)

We saw conditional selection applied to `Series` and it'll work in the same way for `DataFrame`s. After all, a `DataFrame` is a collection of `Series`:

In [179…

```
df
```

Out[179…

| | 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 |
| **United Kingdom** | 64.511 | 2950039 | 242495 | 0.907 | Europe |
| **United States** | 318.523 | 17348075 | 9525067 | 0.915 | America |

In [180…

```
df['Population'] > 70
```

Out[180…

| | Population |
|---|---|
| **Canada** | False |
| **France** | False |
| **Germany** | True |
| **Italy** | False |
| **Japan** | True |
| **United Kingdom** | False |
| **United States** | True |

**dtype:** bool

In [181…

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

Out[181…

| | Population | GDP | Surface Area | HDI | Continent |
|---|---|---|---|---|---|
| **Germany** | 80.940 | 3874437 | 357114 | 0.916 | Europe |
| **Japan** | 127.061 | 4602367 | 377930 | 0.891 | Asia |
| **United States** | 318.523 | 17348075 | 9525067 | 0.915 | America |

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 [182…

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

Out[182…

| | Population |
|---|---|
| **Germany** | 80.940 |
| **Japan** | 127.061 |
| **United States** | 318.523 |

**dtype:** float64

In [183…

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

Out[183…

| | Population | GDP |
|---|---|---|
| **Germany** | 80.940 | 3874437 |
| **Japan** | 127.061 | 4602367 |
| **United States** | 318.523 | 17348075 |

# Dropping stuff

Opposed to the concept of selection, we have "dropping". Instead of pointing out which values you'd like to *select* you could point which ones you'd like to `drop`:

In [184…

```
df.drop('Canada')
```

Out[184…

| | 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 |
| **United Kingdom** | 64.511 | 2950039 | 242495 | 0.907 | Europe |
| **United States** | 318.523 | 17348075 | 9525067 | 0.915 | America |

In [185…

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

Out[185…

| | 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 |
| **United Kingdom** | 64.511 | 2950039 | 242495 | 0.907 | Europe |
| **United States** | 318.523 | 17348075 | 9525067 | 0.915 | America |

In [186…

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

Out[186…

| | GDP | Surface Area | Continent |
|---|---|---|---|
| **Canada** | 1785387 | 9984670 | America |
| **France** | 2833687 | 640679 | Europe |
| **Germany** | 3874437 | 357114 | Europe |
| **Italy** | 2167744 | 301336 | Europe |
| **Japan** | 4602367 | 377930 | Asia |
| **United Kingdom** | 2950039 | 242495 | Europe |
| **United States** | 17348075 | 9525067 | America |

In [187…

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

Out[187…

| | 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 |
| **United Kingdom** | 64.511 | 2950039 | 242495 | 0.907 | Europe |
| **United States** | 318.523 | 17348075 | 9525067 | 0.915 | America |

In [188…

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

Out[188…

| | GDP | Surface Area | Continent |
|---|---|---|---|
| **Canada** | 1785387 | 9984670 | America |
| **France** | 2833687 | 640679 | Europe |
| **Germany** | 3874437 | 357114 | Europe |
| **Italy** | 2167744 | 301336 | Europe |
| **Japan** | 4602367 | 377930 | Asia |
| **United Kingdom** | 2950039 | 242495 | Europe |
| **United States** | 17348075 | 9525067 | America |

In [189…

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

Out[189…

| | GDP | Surface Area | Continent |
|---|---|---|---|
| **Canada** | 1785387 | 9984670 | America |
| **France** | 2833687 | 640679 | Europe |
| **Germany** | 3874437 | 357114 | Europe |
| **Italy** | 2167744 | 301336 | Europe |
| **Japan** | 4602367 | 377930 | Asia |
| **United Kingdom** | 2950039 | 242495 | Europe |
| **United States** | 17348075 | 9525067 | America |

In [190…

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

Out[190…

| | GDP | Surface Area | Continent |
|---|---|---|---|
| **Canada** | 1785387 | 9984670 | America |
| **France** | 2833687 | 640679 | Europe |
| **Germany** | 3874437 | 357114 | Europe |
| **Italy** | 2167744 | 301336 | Europe |
| **Japan** | 4602367 | 377930 | Asia |
| **United Kingdom** | 2950039 | 242495 | Europe |
| **United States** | 17348075 | 9525067 | America |

In [191…

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

Out[191…

| | 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 |
| **United Kingdom** | 64.511 | 2950039 | 242495 | 0.907 | Europe |
| **United States** | 318.523 | 17348075 | 9525067 | 0.915 | America |

All these `drop` methods return a new `DataFrame`. If you'd like to modify it "in place", you can use the `inplace` attribute (there's an example below).

# Operations

In [192…

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

Out[192…

| | Population | GDP |
|---|---|---|
| **Canada** | 35.467 | 1785387 |
| **France** | 63.951 | 2833687 |
| **Germany** | 80.940 | 3874437 |
| **Italy** | 60.665 | 2167744 |
| **Japan** | 127.061 | 4602367 |
| **United Kingdom** | 64.511 | 2950039 |
| **United States** | 318.523 | 17348075 |

In [193…

```
df[['Population', 'GDP']] / 100
```

Out[193…

|  | 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 |
| **United Kingdom** | 0.64511 | 29500.39 |
| **United States** | 3.18523 | 173480.75 |

**Operations with Series** work at a column level, broadcasting down the rows (which can be counter intuitive).

In [194…

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

Out[194…

|  | 0 |
|---|---|
| **GDP** | -1000000.0 |
| **HDI** | -0.3 |

**dtype:** float64

In [195…

```python
df[['GDP', 'HDI']]
```

Out[195…

|  | GDP | HDI |
|---|---|---|
| **Canada** | 1785387 | 0.913 |
| **France** | 2833687 | 0.888 |
| **Germany** | 3874437 | 0.916 |
| **Italy** | 2167744 | 0.873 |
| **Japan** | 4602367 | 0.891 |
| **United Kingdom** | 2950039 | 0.907 |
| **United States** | 17348075 | 0.915 |

In [196…

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

Out[196…

| | 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 |
| **United Kingdom** | 1950039.0 | 0.607 |
| **United States** | 16348075.0 | 0.615 |

## Modifying DataFrames

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

### Adding a new column

In [197…

```
langs = pd.Series(
    ['French', 'German', 'Italian'],
    index=['France', 'Germany', 'Italy'],
    name='Language'
)
```

In [198…

```
langs
```

Out[198…

| | Language |
|---|---|
| **France** | French |
| **Germany** | German |
| **Italy** | Italian |

**dtype:** object

In [199…

```
df['Language'] = langs
```

In [200…

```
df
```

Out[200…

| | Population | GDP | Surface Area | HDI | Continent | Language |
|---|---|---|---|---|---|---|
| **Canada** | 35.467 | 1785387 | 9984670 | 0.913 | America | NaN |
| **France** | 63.951 | 2833687 | 640679 | 0.888 | Europe | French |
| **Germany** | 80.940 | 3874437 | 357114 | 0.916 | Europe | German |
| **Italy** | 60.665 | 2167744 | 301336 | 0.873 | Europe | Italian |
| **Japan** | 127.061 | 4602367 | 377930 | 0.891 | Asia | NaN |
| **United Kingdom** | 64.511 | 2950039 | 242495 | 0.907 | Europe | NaN |
| **United States** | 318.523 | 17348075 | 9525067 | 0.915 | America | NaN |

---

## Replacing values per column

In [201…

```
df['Language'] = 'English'
```

In [202…

```
df
```

Out[202…

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

---

## Renaming Columns

In [203…

```
df.rename(
    columns={
        'HDI': 'Human Development Index',
        'Anual Popcorn Consumption': 'APC'
    }, index={
        'United States': 'USA',
        'United Kingdom': 'UK',
        'Argentina': 'AR'
    })
```

Out[203…

| | Population | GDP | Surface Area | Human Development Index | Continent | Language |
|---|---|---|---|---|---|---|
| **Canada** | 35.467 | 1785387 | 9984670 | 0.913 | America | English |
| **France** | 63.951 | 2833687 | 640679 | 0.888 | Europe | English |
| **Germany** | 80.940 | 3874437 | 357114 | 0.916 | Europe | English |
| **Italy** | 60.665 | 2167744 | 301336 | 0.873 | Europe | English |
| **Japan** | 127.061 | 4602367 | 377930 | 0.891 | Asia | English |
| **UK** | 64.511 | 2950039 | 242495 | 0.907 | Europe | English |
| **USA** | 318.523 | 17348075 | 9525067 | 0.915 | America | English |

In [204…

```
df.rename(index=str.upper)
```

Out[204…

| | Population | GDP | Surface Area | HDI | Continent | Language |
|---|---|---|---|---|---|---|
| **CANADA** | 35.467 | 1785387 | 9984670 | 0.913 | America | English |
| **FRANCE** | 63.951 | 2833687 | 640679 | 0.888 | Europe | English |
| **GERMANY** | 80.940 | 3874437 | 357114 | 0.916 | Europe | English |
| **ITALY** | 60.665 | 2167744 | 301336 | 0.873 | Europe | English |
| **JAPAN** | 127.061 | 4602367 | 377930 | 0.891 | Asia | English |
| **UNITED KINGDOM** | 64.511 | 2950039 | 242495 | 0.907 | Europe | English |
| **UNITED STATES** | 318.523 | 17348075 | 9525067 | 0.915 | America | English |

In [205…

```
df.rename(index=lambda x: x.lower())
```

Out[205…

| | Population | GDP | Surface Area | HDI | Continent | Language |
|---|---|---|---|---|---|---|
| **canada** | 35.467 | 1785387 | 9984670 | 0.913 | America | English |
| **france** | 63.951 | 2833687 | 640679 | 0.888 | Europe | English |
| **germany** | 80.940 | 3874437 | 357114 | 0.916 | Europe | English |
| **italy** | 60.665 | 2167744 | 301336 | 0.873 | Europe | English |
| **japan** | 127.061 | 4602367 | 377930 | 0.891 | Asia | English |
| **united kingdom** | 64.511 | 2950039 | 242495 | 0.907 | Europe | English |
| **united states** | 318.523 | 17348075 | 9525067 | 0.915 | America | English |

---

## Dropping columns

In [206…

```
df.drop(columns='Language', inplace=True)
```

---

## Adding values

In [207…

```
df = pd.concat([df, pd.Series({
    'Population': 3,
    'GDP': 5
}, name='China').to_frame().T])
```

Append returns a new `DataFrame`:

In [208…

```
df
```

Out[208…

| | Population | GDP | Surface Area | HDI | Continent |
|---|---|---|---|---|---|
| **Canada** | 35.467 | 1785387 | 9984670.0 | 0.913 | America |
| **France** | 63.951 | 2833687 | 640679.0 | 0.888 | Europe |
| **Germany** | 80.940 | 3874437 | 357114.0 | 0.916 | Europe |
| **Italy** | 60.665 | 2167744 | 301336.0 | 0.873 | Europe |
| **Japan** | 127.061 | 4602367 | 377930.0 | 0.891 | Asia |
| **United Kingdom** | 64.511 | 2950039 | 242495.0 | 0.907 | Europe |
| **United States** | 318.523 | 17348075 | 9525067.0 | 0.915 | America |
| **China** | 3.000 | 5 | NaN | NaN | NaN |

You can directly set the new index and values to the `DataFrame`:

In [209…

```
df.loc['China'] = pd.Series({'Population': 1_400_000_000, 'Continent': 'Asia'}
```

In [210…

```
df
```

Out[210…

| | Population | GDP | Surface Area | HDI | Continent |
|---|---|---|---|---|---|
| **Canada** | 3.546700e+01 | 1785387.0 | 9984670.0 | 0.913 | America |
| **France** | 6.395100e+01 | 2833687.0 | 640679.0 | 0.888 | Europe |
| **Germany** | 8.094000e+01 | 3874437.0 | 357114.0 | 0.916 | Europe |
| **Italy** | 6.066500e+01 | 2167744.0 | 301336.0 | 0.873 | Europe |
| **Japan** | 1.270610e+02 | 4602367.0 | 377930.0 | 0.891 | Asia |
| **United Kingdom** | 6.451100e+01 | 2950039.0 | 242495.0 | 0.907 | Europe |
| **United States** | 3.185230e+02 | 17348075.0 | 9525067.0 | 0.915 | America |
| **China** | 1.400000e+09 | NaN | NaN | NaN | Asia |

We can use `drop` to just remove a row by index:

In [211…

```
df.drop('China', inplace=True)
```

In [212…

```
df
```

Out[212…

| | Population | GDP | Surface Area | HDI | Continent |
|---|---|---|---|---|---|
| **Canada** | 35.467 | 1785387.0 | 9984670.0 | 0.913 | America |
| **France** | 63.951 | 2833687.0 | 640679.0 | 0.888 | Europe |
| **Germany** | 80.940 | 3874437.0 | 357114.0 | 0.916 | Europe |
| **Italy** | 60.665 | 2167744.0 | 301336.0 | 0.873 | Europe |
| **Japan** | 127.061 | 4602367.0 | 377930.0 | 0.891 | Asia |
| **United Kingdom** | 64.511 | 2950039.0 | 242495.0 | 0.907 | Europe |
| **United States** | 318.523 | 17348075.0 | 9525067.0 | 0.915 | America |

## More radical index changes

In [213…

```
df.reset_index()
```

Out[213…

| | index | Population | GDP | Surface Area | HDI | Continent |
|---|---|---|---|---|---|---|
| **0** | Canada | 35.467 | 1785387.0 | 9984670.0 | 0.913 | America |
| **1** | France | 63.951 | 2833687.0 | 640679.0 | 0.888 | Europe |
| **2** | Germany | 80.940 | 3874437.0 | 357114.0 | 0.916 | Europe |
| **3** | Italy | 60.665 | 2167744.0 | 301336.0 | 0.873 | Europe |
| **4** | Japan | 127.061 | 4602367.0 | 377930.0 | 0.891 | Asia |
| **5** | United Kingdom | 64.511 | 2950039.0 | 242495.0 | 0.907 | Europe |
| **6** | United States | 318.523 | 17348075.0 | 9525067.0 | 0.915 | America |

In [214…

```
df.set_index('Population')
```

```
Out[214…
                   GDP  Surface Area    HDI  Continent
Population
   35.467    1785387.0     9984670.0  0.913    America
   63.951    2833687.0      640679.0  0.888     Europe
   80.940    3874437.0      357114.0  0.916     Europe
   60.665    2167744.0      301336.0  0.873     Europe
  127.061    4602367.0      377930.0  0.891       Asia
   64.511    2950039.0      242495.0  0.907     Europe
  318.523   17348075.0     9525067.0  0.915    America
```

## Creating columns from other columns

Altering a DataFrame often involves combining different columns into another. For example, in our Countries analysis, we could try to calculate the "GDP per capita", which is just, `GDP / Population`.

```
In [215…
df[['Population', 'GDP']]
```

```
Out[215…
                Population         GDP
        Canada      35.467   1785387.0
        France      63.951   2833687.0
       Germany      80.940   3874437.0
         Italy      60.665   2167744.0
         Japan     127.061   4602367.0
United Kingdom      64.511   2950039.0
 United States     318.523  17348075.0
```

The regular pandas way of expressing that, is just dividing each series:

```
In [216…
df['GDP'] / df['Population']
```

Out[216…

|  | 0 |
|---|---|
| **Canada** | 50339.385908 |
| **France** | 44310.284437 |
| **Germany** | 47868.013343 |
| **Italy** | 35733.025633 |
| **Japan** | 36221.712406 |
| **United Kingdom** | 45729.239975 |
| **United States** | 54464.120330 |

**dtype:** float64

The result of that operation is just another series that you can add to the original `DataFrame`:

In [217…

```
df['GDP Per Capita'] = df['GDP'] / df['Population']
```

In [218…

```
df
```

Out[218…

|  | Population | GDP | Surface Area | HDI | Continent | GDP Per Capita |
|---|---|---|---|---|---|---|
| **Canada** | 35.467 | 1785387.0 | 9984670.0 | 0.913 | America | 50339.385908 |
| **France** | 63.951 | 2833687.0 | 640679.0 | 0.888 | Europe | 44310.284437 |
| **Germany** | 80.940 | 3874437.0 | 357114.0 | 0.916 | Europe | 47868.013343 |
| **Italy** | 60.665 | 2167744.0 | 301336.0 | 0.873 | Europe | 35733.025633 |
| **Japan** | 127.061 | 4602367.0 | 377930.0 | 0.891 | Asia | 36221.712406 |
| **United Kingdom** | 64.511 | 2950039.0 | 242495.0 | 0.907 | Europe | 45729.239975 |
| **United States** | 318.523 | 17348075.0 | 9525067.0 | 0.915 | America | 54464.120330 |

# Statistical info

You've already seen the `describe` method, which gives you a good "summary" of the `DataFrame`. Let's explore other methods in more detail:

In [219…

```
df.head()
```

Out[219…

|  | Population | GDP | Surface Area | HDI | Continent | GDP Per Capita |
|---|---|---|---|---|---|---|
| **Canada** | 35.467 | 1785387.0 | 9984670.0 | 0.913 | America | 50339.385908 |
| **France** | 63.951 | 2833687.0 | 640679.0 | 0.888 | Europe | 44310.284437 |
| **Germany** | 80.940 | 3874437.0 | 357114.0 | 0.916 | Europe | 47868.013343 |
| **Italy** | 60.665 | 2167744.0 | 301336.0 | 0.873 | Europe | 35733.025633 |
| **Japan** | 127.061 | 4602367.0 | 377930.0 | 0.891 | Asia | 36221.712406 |

In [220…

```
df.describe()
```

Out[220…

|  | Population | GDP | Surface Area | HDI | GDP Per Capita |
|---|---|---|---|---|---|
| **count** | 7.000000 | 7.000000e+00 | 7.000000e+00 | 7.000000 | 7.000000 |
| **mean** | 107.302571 | 5.080248e+06 | 3.061327e+06 | 0.900429 | 44952.254576 |
| **std** | 97.249970 | 5.494020e+06 | 4.576187e+06 | 0.016592 | 6954.983875 |
| **min** | 35.467000 | 1.785387e+06 | 2.424950e+05 | 0.873000 | 35733.025633 |
| **25%** | 62.308000 | 2.500716e+06 | 3.292250e+05 | 0.889500 | 40265.998421 |
| **50%** | 64.511000 | 2.950039e+06 | 3.779300e+05 | 0.907000 | 45729.239975 |
| **75%** | 104.000500 | 4.238402e+06 | 5.082873e+06 | 0.914000 | 49103.699626 |
| **max** | 318.523000 | 1.734808e+07 | 9.984670e+06 | 0.916000 | 54464.120330 |

In [221…

```
population = df['Population']
```

In [222…

```
population.min(), population.max()
```

Out[222…

```
(35.467, 318.523)
```

In [223…

```
population.sum()
```

Out[223…

```
np.float64(751.118)
```

In [224…

```
population.sum() / len(population)
```

Out[224…

```
np.float64(107.30257142857144)
```

In [225…

```
population.mean()
```

Out[225…

```
np.float64(107.30257142857144)
```

In [226…

```
population.std()
```

Out[226…

```
97.24996987121581
```

In [227…

```python
population.median()
```

Out[227…

```
64.511
```

In [228…

```python
population.describe()
```

Out[228…

|  | Population |
|---|---|
| **count** | 7.000000 |
| **mean** | 107.302571 |
| **std** | 97.249970 |
| **min** | 35.467000 |
| **25%** | 62.308000 |
| **50%** | 64.511000 |
| **75%** | 104.000500 |
| **max** | 318.523000 |

**dtype:** float64

In [229…

```python
population.quantile(.25)
```

Out[229…

```
np.float64(62.308)
```

In [230…

```python
population.quantile([.2, .4, .6, .8, 1])
```

Out[230…

|  | Population |
|---|---|
| **0.2** | 61.3222 |
| **0.4** | 64.1750 |
| **0.6** | 74.3684 |
| **0.8** | 117.8368 |
| **1.0** | 318.5230 |

**dtype:** float64