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 [1]:

```
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 (https://docs.google.com/spreadsheets/d/1IlorV2-Oh9Da1JAZ7weVw86PQrQydSMp-ydVMH135iI/edit?usp=sharing)):

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 [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'm using it to keep the same order as in the picture above)

In [3]:

```
df
```

Out[3]:

	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 [4]:

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

In [5]:

```
df
```

Out[5]:

	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 [6]:

```
df.columns
```

Out[6]:

```
Index(['Population', 'GDP', 'Surface Area', 'HDI', 'Continent'], dty
pe='object')
```

In [7]:

```
df.index
```

Out[7]:

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

In [8]:

```
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: 336.0+ bytes
```

In [9]:

```
df.size
```

Out[9]:

```
35
```

In [10]:

```
df.shape
```

Out[10]:

```
(7, 5)
```

In [12]:

```
df.describe()
```

Out[12]:

	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 [13]:

```
df.dtypes
```

Out[13]:

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

In [14]:

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

Out[14]:

```
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 [15]:

```
df
```

Out[15]:

	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 [16]:

```
df.loc['Canada'] # let you select individual rows - by index
```

Out[16]:

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

In [17]:

```
df.iloc[-1] # let you select the last row - by sequential position
```

Out[17]:

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

In [18]:

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

Out[18]:

```
Canada            35.467
France            63.951
Germany           80.940
Italy             60.665
Japan            127.061
United Kingdom    64.511
United States    318.523
Name: Population, 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 [ ]:

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

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

In [ ]:

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

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

In [ ]:

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

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 [19]:

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

Out[19]:

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

In [20]:

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

Out[20]:

	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 [21]:

```
df.loc['France': 'Italy', 'Population'] # only population column
```

Out[21]:

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

In [22]:

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

Out[22]:

	Population	GDP
France	63.951	2833687
Germany	80.940	3874437
Italy	60.665	2167744

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

In [23]:

```
df
```

Out[23]:

	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 [24]:

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

Out[24]:

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

In [25]:

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

Out[25]:

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

In [26]:

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

Out[26]:

	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 [27]:

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

Out[27]:

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

In [28]:

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

Out[28]:

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

In [ ]:

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

In [ ]:

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

> **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 [29]:

```
df
```

Out[29]:

	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 [30]:

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

Out[30]:

```
Canada           False
France           False
Germany           True
Italy            False
Japan             True
United Kingdom   False
United States     True
Name: Population, dtype: bool
```

In [31]:

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

Out[31]:

	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 [32]:

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

Out[32]:

```
Germany           80.940
Japan            127.061
United States    318.523
Name: Population, dtype: float64
```

In [33]:

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

Out[33]:

	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 [47]:

```
df.drop('Canada') # removes rows
```

Out[47]:

	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 [35]:

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

Out[35]:

	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 [36]:

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

Out[36]:

	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 [37]:

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

Out[37]:

	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 [38]:

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

Out[38]:

	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 [39]:

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

Out[39]:

	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 [40]:

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

Out[40]:

	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 [41]:

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

Out[41]:

	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 [42]:

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

Out[42]:

	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 [43]:

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

Out[43]:

	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 [44]:

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

Out[44]:

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

In [45]:

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

Out[45]:

	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 [46]:

```
df[['GDP', 'HDI']] + crisis # subtract 1 million from GDP and 0.3 from HDI
```

Out[46]:

	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 [49]:

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

In [52]:

```
langs
```

Out[52]:

```
France      French
Germany     German
Italy      Italian
Name: Language, dtype: object
```

In [50]:

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

In [51]:

```
df
```

Out[51]:

	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 [53]:

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

In [54]:

```
df
```

Out[54]:

	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 [55]:

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

Out[55]:

	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 [ ]:

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

In [ ]:

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

Dropping columns

In [ ]:

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

Adding values

In [57]:

```
df.append(pd.Series({
    'Population': 3,
    'GDP': 5
}, name='China'))
```

Out[57]:

	Population	GDP	Surface Area	HDI	Continent	Language
Canada	35.467	1785387.0	9984670.0	0.913	America	English
France	63.951	2833687.0	640679.0	0.888	Europe	English
Germany	80.940	3874437.0	357114.0	0.916	Europe	English
Italy	60.665	2167744.0	301336.0	0.873	Europe	English
Japan	127.061	4602367.0	377930.0	0.891	Asia	English
United Kingdom	64.511	2950039.0	242495.0	0.907	Europe	English
United States	318.523	17348075.0	9525067.0	0.915	America	English
China	3.000	5.0	NaN	NaN	NaN	NaN

Append returns a new `DataFrame` :

In [58]:

```
df
```

Out[58]:

	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

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

In [59]:

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

In [ ]:

```
df
```

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

In [ ]:

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

In [ ]:

```
df
```

More radical index changes

In [ ]:

```
df.reset_index()
```

In [ ]:

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

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 [ ]:

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

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

In [60]:

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

Out[60]:

```
Canada            50339.385908
France            44310.284437
Germany           47868.013343
Italy             35733.025633
Japan             36221.712406
United Kingdom    45729.239975
United States     54464.120330
China                      NaN
dtype: float64
```

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

In [61]:

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

In [62]:

```
df
```

Out[62]:

	Population	GDP	Surface Area	HDI	Continent	Language	GDP Per Capita
Canada	3.546700e+01	1785387.0	9984670.0	0.913	America	English	50339.385908
France	6.395100e+01	2833687.0	640679.0	0.888	Europe	English	44310.284437
Germany	8.094000e+01	3874437.0	357114.0	0.916	Europe	English	47868.013343
Italy	6.066500e+01	2167744.0	301336.0	0.873	Europe	English	35733.025633
Japan	1.270610e+02	4602367.0	377930.0	0.891	Asia	English	36221.712406
United Kingdom	6.451100e+01	2950039.0	242495.0	0.907	Europe	English	45729.239975
United States	3.185230e+02	17348075.0	9525067.0	0.915	America	English	54464.120330
China	1.400000e+09	NaN	NaN	NaN	Asia	NaN	NaN

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 [63]:

```
df.head()
```

Out[63]:

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

In [64]:

```
df.describe()
```

Out[64]:

	Population	GDP	Surface Area	HDI	GDP Per Capita
count	8.000000e+00	7.000000e+00	7.000000e+00	7.000000	7.000000
mean	1.750001e+08	5.080248e+06	3.061327e+06	0.900429	44952.254576
std	4.949747e+08	5.494020e+06	4.576187e+06	0.016592	6954.983875
min	3.546700e+01	1.785387e+06	2.424950e+05	0.873000	35733.025633
25%	6.312950e+01	2.500716e+06	3.292250e+05	0.889500	40265.998421
50%	7.272550e+01	2.950039e+06	3.779300e+05	0.907000	45729.239975
75%	1.749265e+02	4.238402e+06	5.082873e+06	0.914000	49103.699626
max	1.400000e+09	1.734808e+07	9.984670e+06	0.916000	54464.120330

In [65]:

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

In [ ]:

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

In [ ]:

```
population.sum()
```

In [ ]:

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

In [ ]:

```
population.mean()
```

In [ ]:

```
population.std()
```

In [ ]:

```
population.median()
```

In [ ]:

```
population.describe()
```

In [ ]:

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

In [ ]:

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