Introduction to Pandas DataFrames

In [69]:
```
import pandas as pd
import numpy as np
```

Introduction

This documentation is intended to give an introduction to using DataFrames in Python based Pandas data analysis and manipulation framework. This document was created using a Jupyter notebook and all code examples can be manipulated live in a Jupyter notebook.

Prerequisites

- basic Python
- basic data analysis knowledge

For More Information

- Jupyter (https://jupyter.org)
- Pandas (https://pandas.pydata.org)
- Pandas DateFrame (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html)

This documentation files are available in Github (https://github.com/kuikala/DataSamples).

What are DataFrames ?

The Pandas framework lets data developers to do everything from basic to complex analytics on data sets of any size in Python. Pandas abstracts complex data analysis workflows into the Python environment allowing developers to create reusable code to drive analytics applications.

Data analysis is all about working with tables of data. Developers need to be able to view, manipulate, merge, sort, subset, etc on the data, but all these tasks start with tables of data. The Pandas library data structure used to hold tables of data are DataFrames.

Why use DataFrames ?

You may be asking yourself, why not use Python lists, dictionaries, arrays or some custom data structure to hold my data ? DataFrames bring simplicity to managing tables of data and includes powerful functionality that would be complex to write against standard Python objects.

Creating DataFrames

Let's start by creating a simple data frame with data on the ten largest cities in the United States.

First we create some basic Python Lists and create a DataFrame from them:

In [70]:
```
city_population_data = {
     'city': ['New York', 'Los Angeles', 'Chicago', 'Houston',
              'Phoenix', 'Philadelphia', 'San Antonio', 'San Diego', 'Dalla
     'state': ['NY', 'CA', 'IL', 'TX', 'AZ', 'PA', 'TX', 'CA', 'TX', 'CA'],
     'population': [8336817, 3979576, 2693976, 2320268, 1680992, 1584064, 1
}

city_population_df = pd.DataFrame(data=city_population_data)
```

Viewing the data is simple:

In [71]:
```
city_population_df
```

Out[71]:

	city	state	population
0	New York	NY	8336817
1	Los Angeles	CA	3979576
2	Chicago	IL	2693976
3	Houston	TX	2320268
4	Phoenix	AZ	1680992
5	Philadelphia	PA	1584064
6	San Antonio	TX	1547253
7	San Diego	CA	1423851
8	Dallas	TX	1343573
9	San Jose	CA	1021795

We can also look at the first few rows:

In [72]:
```
city_population_df.head(3)
```

Out[72]:

	city	state	population
0	New York	NY	8336817
1	Los Angeles	CA	3979576
2	Chicago	IL	2693976

Or just look at specific columns:

In [73]:
```
city_population_df[['city','state']]
```

Out[73]:

	city	state
0	New York	NY
1	Los Angeles	CA
2	Chicago	IL
3	Houston	TX
4	Phoenix	AZ
5	Philadelphia	PA
6	San Antonio	TX
7	San Diego	CA
8	Dallas	TX
9	San Jose	CA

We can filter the list to just those cities with more than 2 million residents:

In [74]:
```
city_population_df[city_population_df.population > 2000000]
```

Out[74]:

	city	state	population
0	New York	NY	8336817
1	Los Angeles	CA	3979576
2	Chicago	IL	2693976
3	Houston	TX	2320268

or just those cities in California:

In [75]:
```
city_population_df[city_population_df.state=='CA']
```

Out[75]:

	city	state	population
1	Los Angeles	CA	3979576
7	San Diego	CA	1423851
9	San Jose	CA	1021795

or combine those and get the cities in California with over 2 million people ?

In [76]:
```
city_population_df[(city_population_df.population > 2000000) &
                   (city_population_df.state=='CA')]
```

Out[76]:

	city	state	population
1	Los Angeles	CA	3979576

DataFrames allow you to slice and dice the data in nearly anyway you can imagine.

We can also sort the data, by state and population:

In [77]:
```
city_population_df.sort_values(by=['state','population'], ascending=[True,
```

Out[77]:

	city	state	population
4	Phoenix	AZ	1680992
1	Los Angeles	CA	3979576
7	San Diego	CA	1423851
9	San Jose	CA	1021795
2	Chicago	IL	2693976
0	New York	NY	8336817
5	Philadelphia	PA	1584064
3	Houston	TX	2320268
6	San Antonio	TX	1547253
8	Dallas	TX	1343573

We can combine the filtering and sorting to narrow down the data down to just those cities in Texas and California from smallest to largest:

In [78]:
```
city_population_df[(city_population_df.state=='TX') |
                   (city_population_df.state=='CA')].sort_values(by='popula
```

Out[78]:

	city	state	population
9	San Jose	CA	1021795
8	Dallas	TX	1343573
7	San Diego	CA	1423851
6	San Antonio	TX	1547253
3	Houston	TX	2320268
1	Los Angeles	CA	3979576

We can also do aggregate operations on data, for instance, how many people are in the top ten cities and how many people on average:

In [79]:
```
city_population_df['population'].sum()
```

Out[79]:
```
25932165
```

In [80]:
```
city_population_df['population'].mean()
```

Out[80]:
```
2593216.5
```

We can also count the number of rows of data that came back, so how many of the top ten cities are in Texas or California:

In [81]:
```
(city_population_df[(city_population_df.state=='TX') | (city_population_df.
```

Out[81]:
```
6
```

This is amazing, what is next ?

Now that we have reviewed some of the basics, the next steps is to dive into more detail of the functionality starting with all the ways that DataFrames can be created.