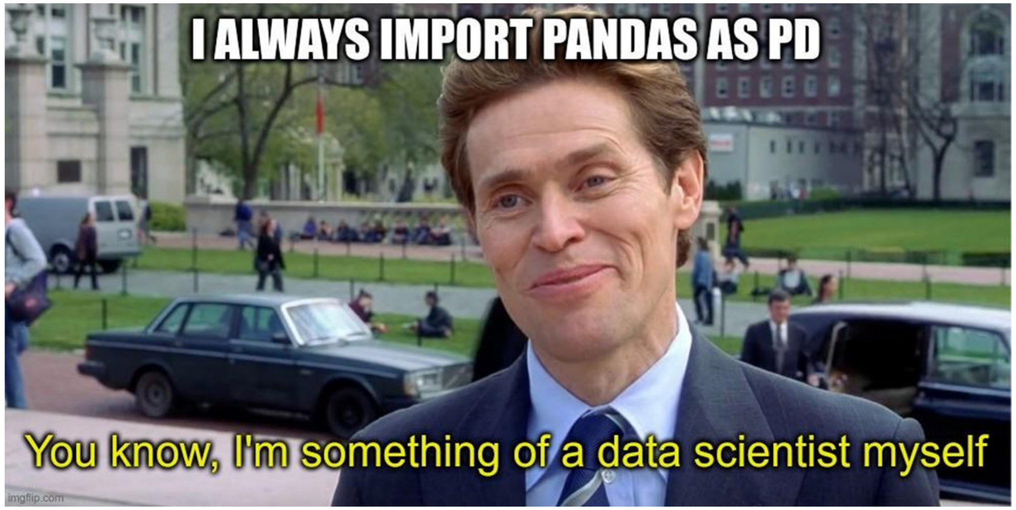
Data and Statistics with Python

So, you want to become a data scientist?

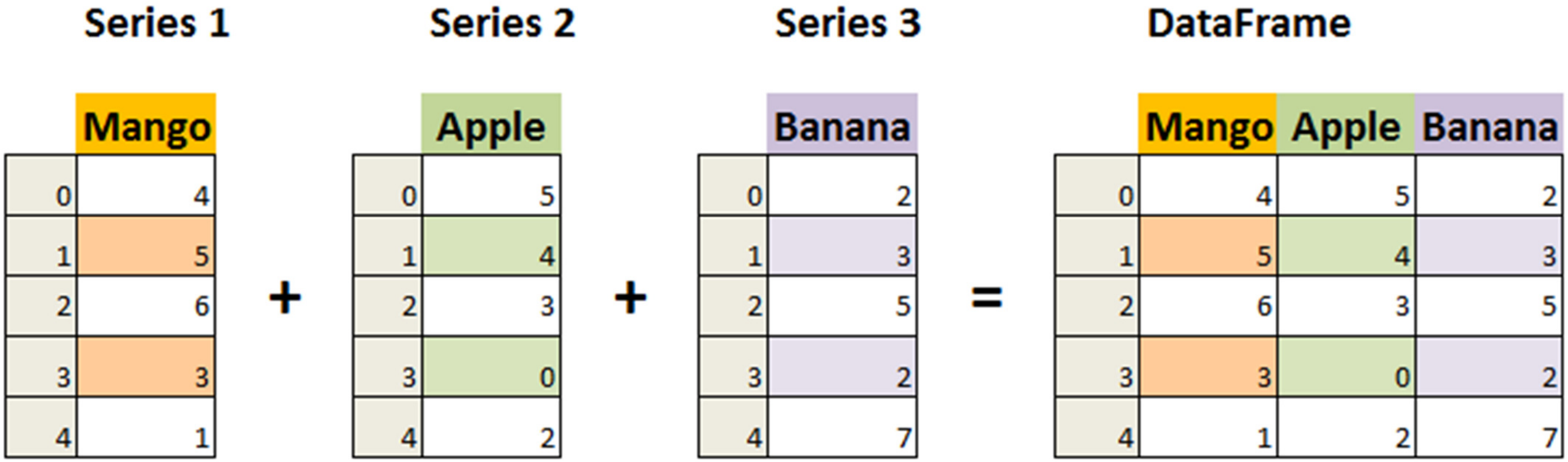

I ALWAYS IMPORT PANDAS AS PD
You know, I'm something of a data scientist myself
imgflip.com

Requirement

```
pip install pandas bokeh
```

Creating Series and DataFrame objects

- Pandas objects can be thought of as enhanced versions of NumPy structured arrays in which the rows and columns are identified with labels rather than simple integer indices.
- Pandas provides a host of useful tools, methods, and functionality on top of the basic data structures, but nearly everything that follows will require an understanding of what these structures are. Thus, before we go any further, let's introduce these three fundamental Pandas data structures: the Series and DataFrame.



Series 1

	Mango
0	4
1	5
2	6
3	3
4	1

+

Series 2

	Apple
0	5
1	4
2	3
3	0
4	2

+

Series 3

	Banana
0	2
1	3
2	5
3	2
4	7

=

DataFrame

	Mango	Apple	Banana
0	4	5	2
1	5	4	3
2	6	3	5
3	3	0	2
4	1	2	7

Series and DataFrame

```
import pandas as pd
import numpy as np
from numpy.random import default_rng

rng = default_rng(12345) #seed for random data
diff_data = rng.normal(0, 1, size=100)  #generate random data
cumulative = np.add.accumulate(diff_data)
data_series = pd.Series(diff_data) #create Series
print(data_series) # print data
data_frame = pd.DataFrame({
    "diffs": data_series,
    "cumulative": data_series.cumsum()
}) #create data frame
print(data_frame)
```

Loading and storing data from DataFrame

- Usually, we create a DataFrame object from the raw data in a Python session.
- In practice, the data will often come from an external source, such as an existing spreadsheet or CSV file, database, or API endpoint.
- For this reason, pandas provides numerous utilities for loading and storing data to file.
- Out of the box, pandas supports loading and storing data from CSV, Excel (xls or xlsx), JSON, SQL, Parquet, and Google BigQuery.
- This makes it very easy to import your data into pandas and then manipulate and analyze this data using Python.

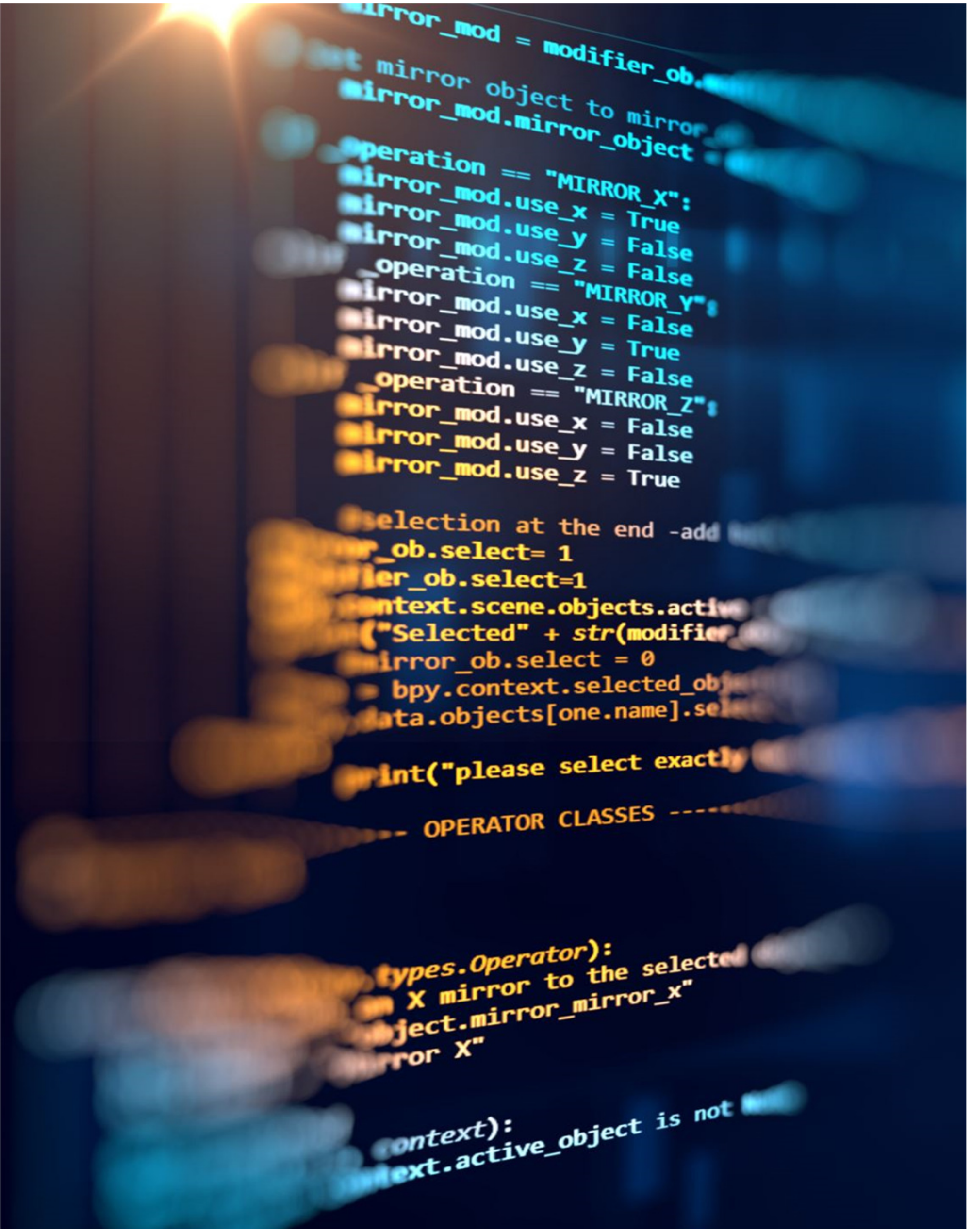
Save and Load

- Save DataFrame to CSV file

```
data_frame.to_csv("sample.csv", index=False)
```

- Load data from CSV file to DataFrame

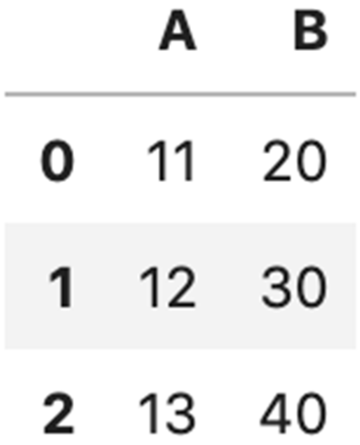
```
df = pd.read_csv("sample.csv", index_col=False)
print(df)
```



```
import pandas as pd
import numpy as np
from numpy.random import default_rng
rng = default_rng(12345)

diffs = rng.normal(0, 1, size=100)
cumulative = np.add.accumulate(diffs)

data_frame = pd.DataFrame({
   "diffs": diffs,
   "cumulative": cumulative
})
print(data_frame)

data_frame.to_csv("sample.csv", index=False)
df = pd.read_csv("sample.csv", index_col=False)
print(df)
```

Manipulate DataFrame

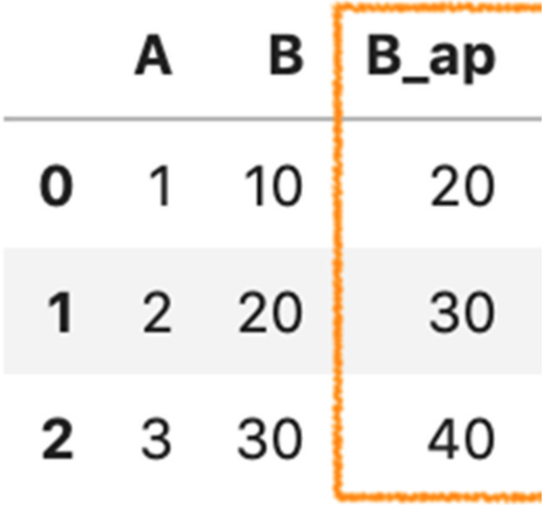
```
df = pd.DataFrame({'A': [1,2,3], 'B': [10,20,30] })
def plus_10(x):
        return x+10
df.apply(plus_10)
```



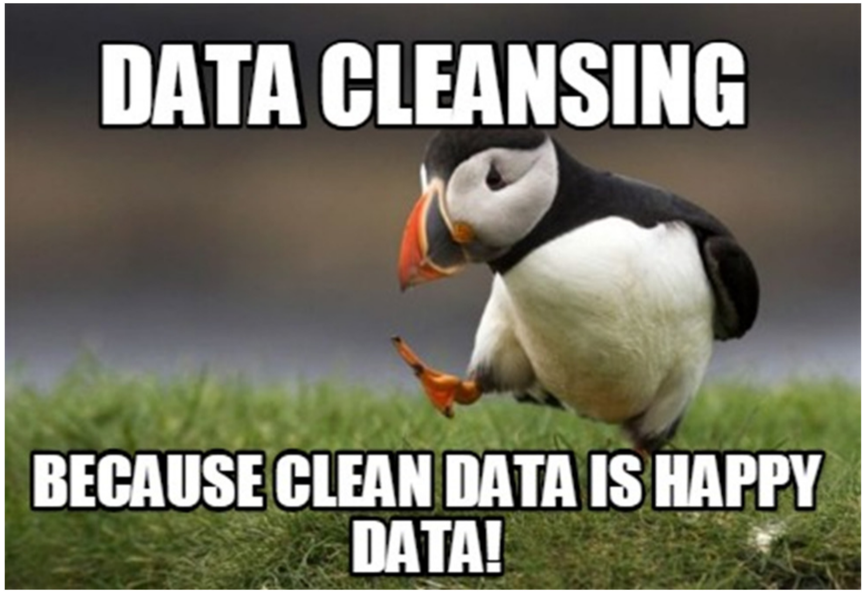
	A	B
0	11	20
1	12	30
2	13	40

Manipulate single column

- df['B_ap'] = df['B'].**apply(plus_10)**



	A	B	B_ap
0	1	10	20
1	2	20	30
2	3	30	40



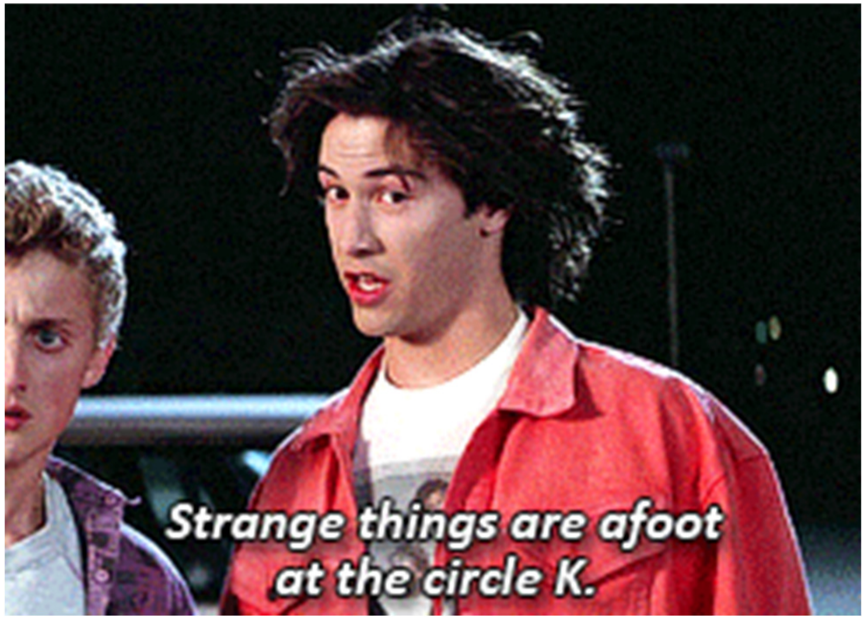
What about null vales?

- df = data_frame.dropna()

NaN, NaT, and None

- If a column is **numeric** and you have a missing value that value will be a NaN. **NaN stands for *Not a Number*.**
- If a column is a **DateTime** and you have a missing value, then that value will be a NaT. **NaT stands for *Not a Time***.

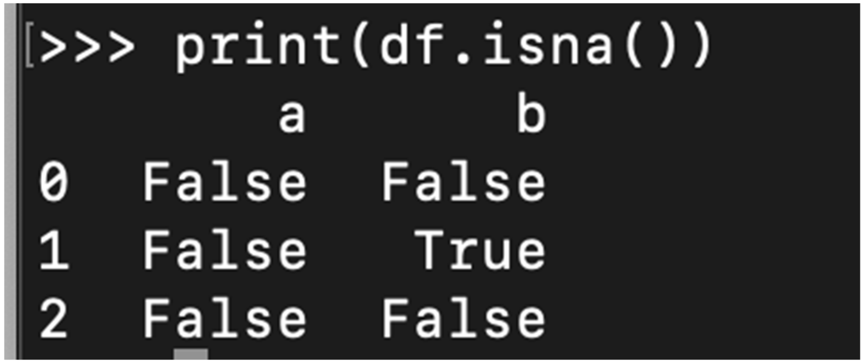
A pandas object dtype column - the dtype for strings as of this writing - can hold None, NaN, NaT or all three at the same time!



What are these NaN values anyway?

- NaN is a NumPy value. np.**NaN**
- NaT is a Pandas value. pd.**NaT**
- None is a vanilla Python value. **None**

Finding missing values



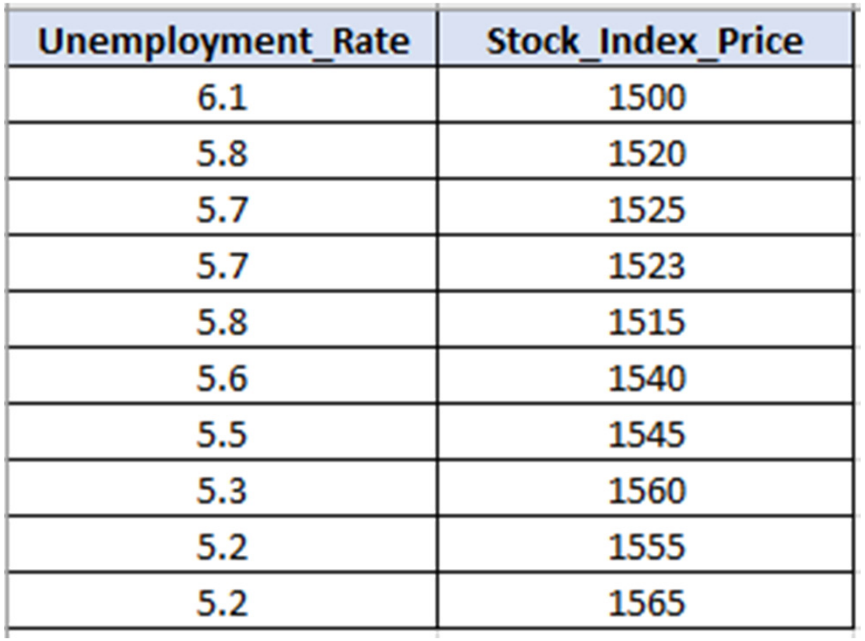
```
[>>> print(df.isna())
       a      b
0  False  False
1  False   True
2  False  False
```

```
import pandas as pd
import numpy as np
#creating the DataFrame
df = pd.DataFrame({'a':[0,1,''],'b':['',None,3]})
print("------The DataFrame is----------")
print(df)
print("--------------------------------")
print(df.isna())
```

Plotting data from a DataFrame

- Step 1: Prepare the data
- Step 2: Create the DataFrame
- Step 3: Plot the DataFrame using Pandas

Step 1: Prepare the data

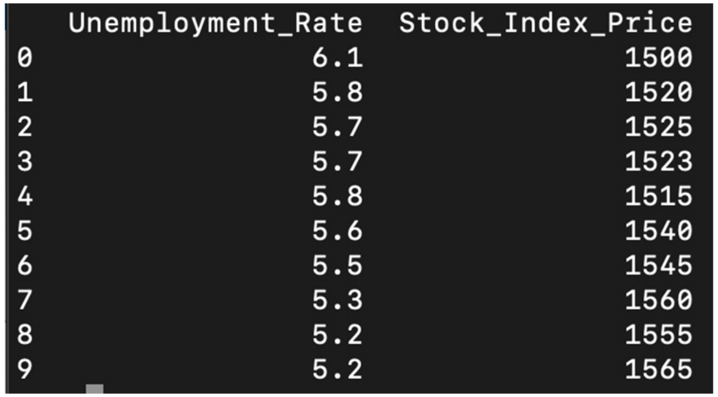


Unemployment_Rate	Stock_Index_Price
6.1	1500
5.8	1520
5.7	1525
5.7	1523
5.8	1515
5.6	1540
5.5	1545
5.3	1560
5.2	1555
5.2	1565

- The following data will be used to create the scatter diagram. This data captures the relationship between two variables related to an economy:

Step 2: Create the DataFrame

```
import pandas as pd

data = {'Unemployment_Rate':
[6.1,5.8,5.7,5.7,5.8,5.6,5.5,5.3,5.2,5.2], 'Stock_Index_Price':
[1500,1520,1525,1523,1515,1540,1545,1560,1555,1565] }


df =
pd.DataFrame(data,columns=['Unemployment_Rate','Stock
_Index_Price'])


print (df)
```



```
   Unemployment_Rate  Stock_Index_Price
0                6.1               1500
1                5.8               1520
2                5.7               1525
3                5.7               1523
4                5.8               1515
5                5.6               1540
6                5.5               1545
7                5.3               1560
8                5.2               1555
9                5.2               1565
```

Step 3: Plot the DataFrame using Pandas

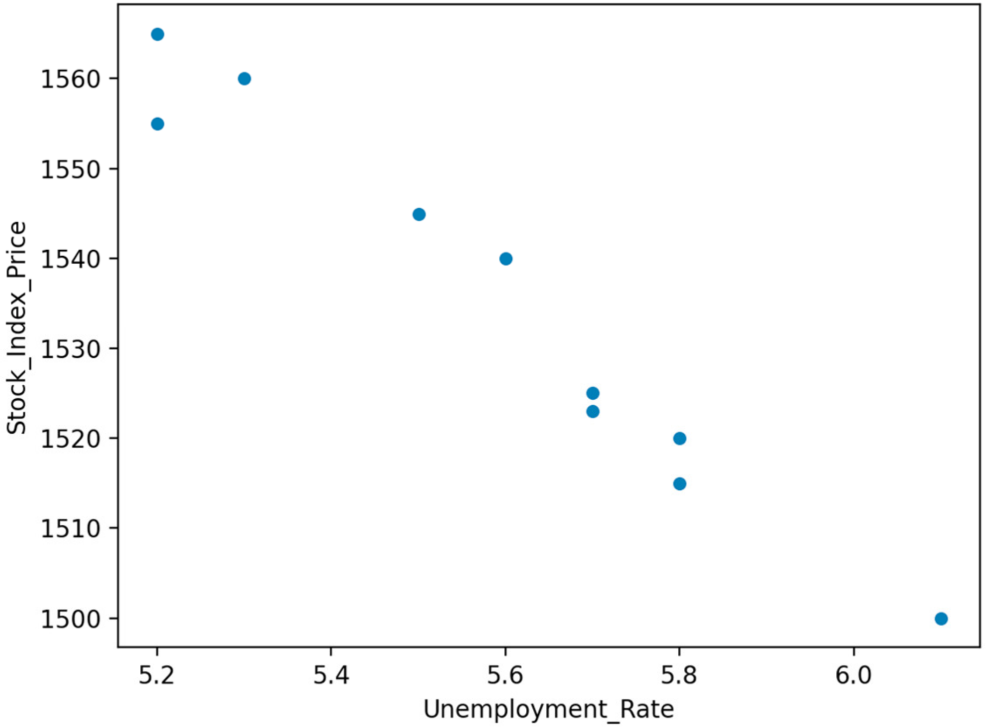
- df.plot(x ='Unemployment_Rate', y='Stock_Index_Price', kind = 'scatter’)

- Notice that you can specify the type of chart by setting **kind = ‘scatter’**
- You’ll also need to add the Matplotlib syntax to show the plot (ensure that the Matplotlib package is install in Python):
- **import matplotlib.pyplot as plt**

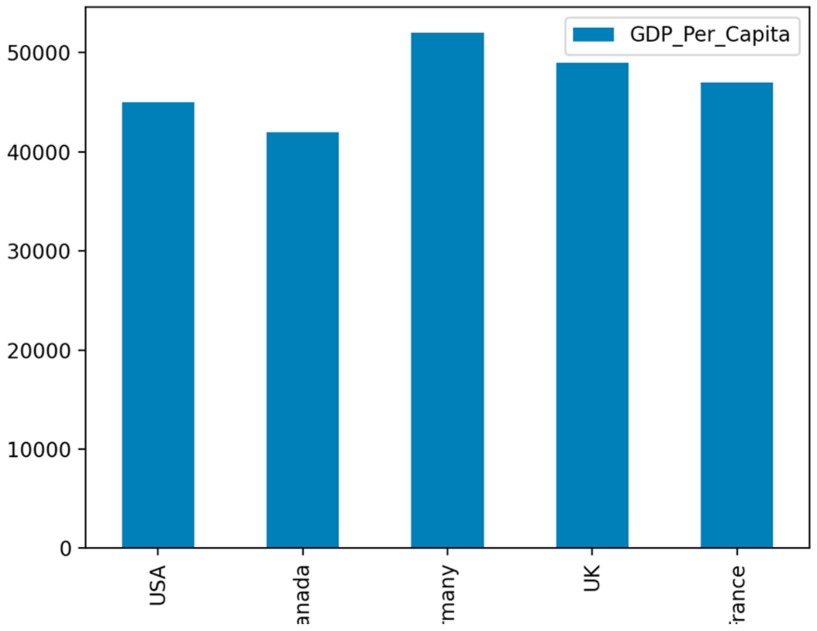
Putting everything together:

```
import pandas as pd
import matplotlib.pyplot as plt
data = {'Unemployment_Rate': [6.1,5.8,5.7,5.7,5.8,5.6,5.5,5.3,5.2,5.2],
'Stock_Index_Price': [1500,1520,1525,1523,1515,1540,1545,1560,1555,1565] }
df = pd.DataFrame(data,columns=['Unemployment_Rate','Stock_Index_Price'])
df.plot(x ='Unemployment_Rate', y='Stock_Index_Price', kind = 'scatter')
plt.show()
```

Output

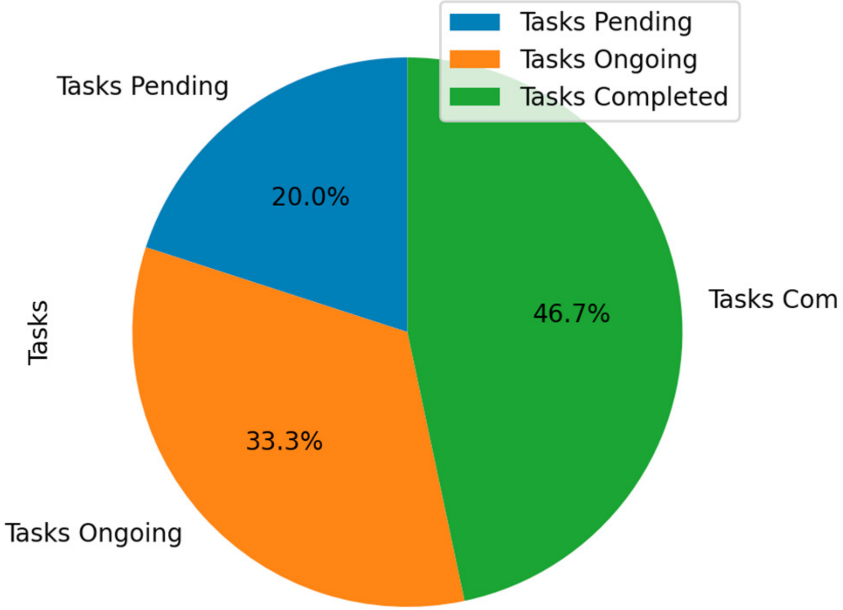


1560
1550
1540
1530
1520
1510
1500
Stock_Index_Price
5.2
5.4
5.6
5.8
6.0
Unemployment_Rate



Plot a Bar Chart using Pandas

```
import pandas as pd

import matplotlib.pyplot as plt

data = {'Country':
['USA','Canada','Germany','UK','France'],
'GDP_Per_Capita': [45000,42000,52000,49000,47000]
}

df = pd.DataFrame(data,columns=['Country','GDP_Per_Capita'])

df.plot(x ='Country', y='GDP_Per_Capita', kind = 'bar’)

plt.show()
```



Plot a Pie Chart using Pandas

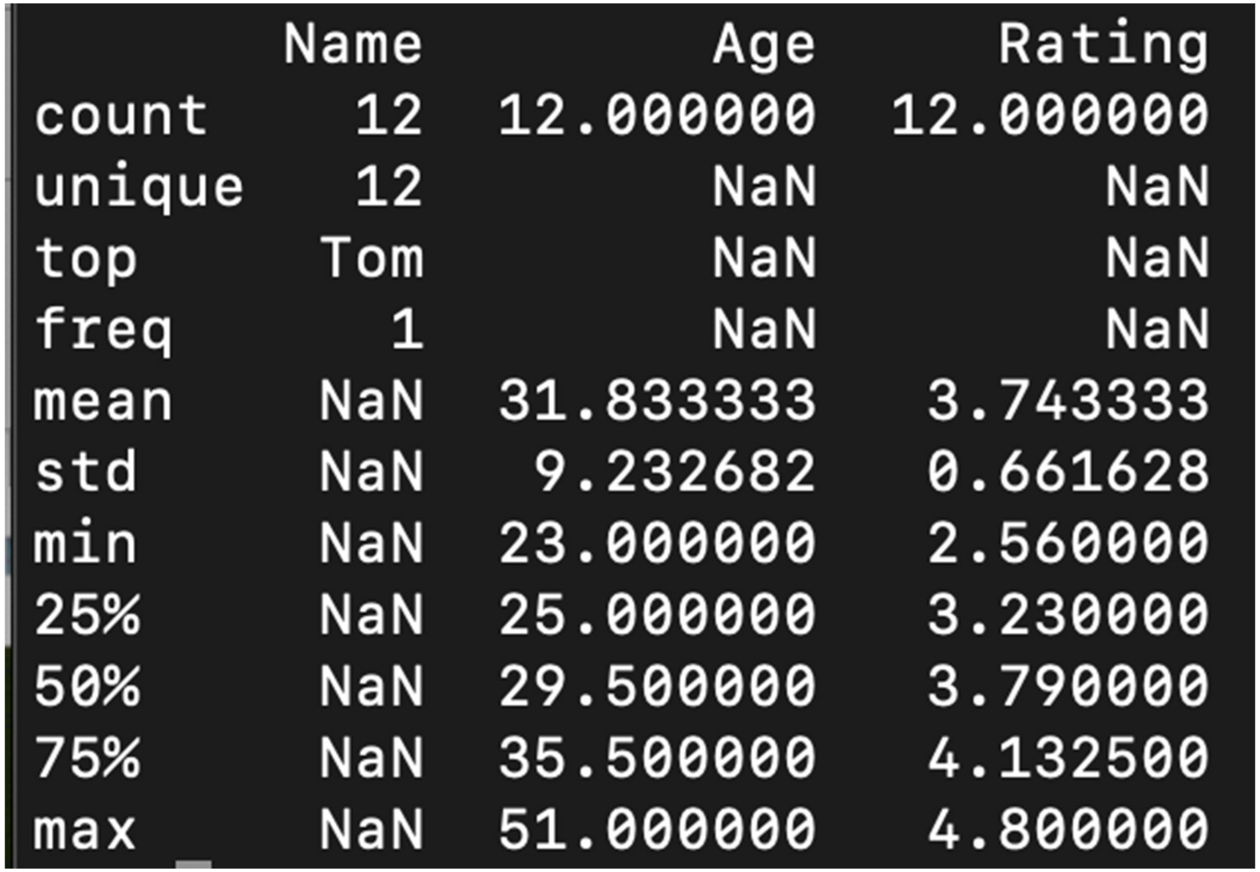
```
import pandas as pd
import matplotlib.pyplot as plt
 data = {'Tasks': [300,500,700]}
df =
pd.DataFrame(data,columns=['Tasks'],index
= ['Tasks Pending','Tasks Ongoing','Tasks
Completed’])
df.plot.pie(y='Tasks',figsize=(5,
5),autopct='%1.1f%%', startangle=90)
plt.show()
```

There are other libs in python for plotting graphs!

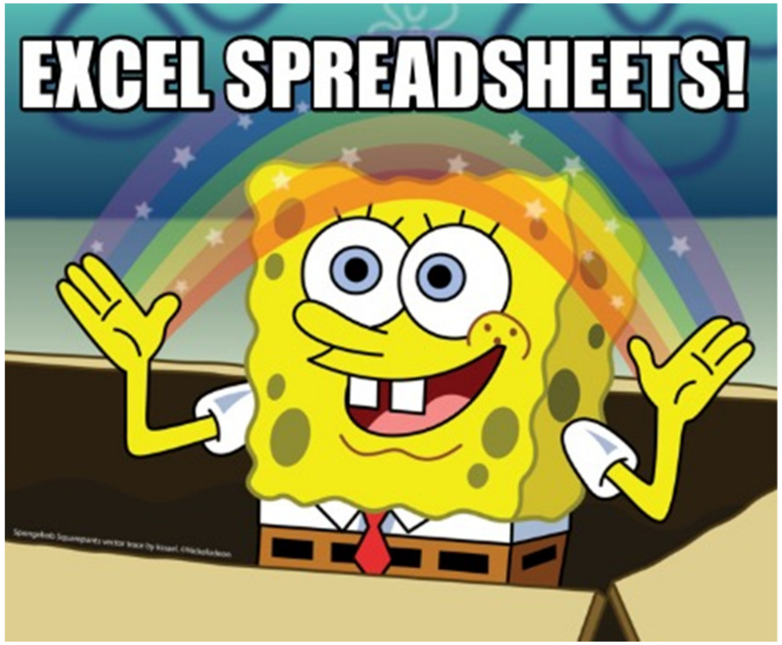
Descriptive statistics

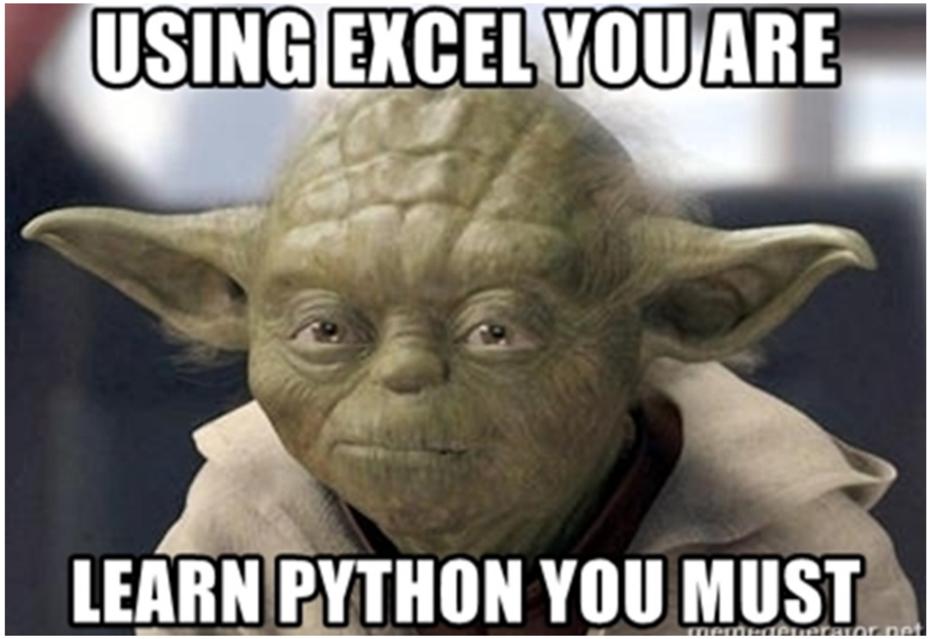
- Descriptive statistics summarizes or describes the characteristics of a data set.
- Descriptive statistics consists of two basic categories of measures: measures of central tendency and measures of variability (or spread).
- Measures of central tendency describe the center of a data set.
- Measures of variability or spread describe the dispersion of data within the set.

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])
}

#Create a DataFrame
df = pd.DataFrame(d)
print (df. describe(include='all'))
```



```
       Name        Age     Rating
count    12  12.000000  12.000000
unique   12        NaN        NaN
top     Tom        NaN        NaN
freq      1        NaN        NaN
mean    NaN  31.833333   3.743333
std     NaN   9.232682   0.661628
min     NaN  23.000000   2.560000
25%     NaN  25.000000   3.230000
50%     NaN  29.500000   3.790000
75%     NaN  35.500000   4.132500
max     NaN  51.000000   4.800000
```





You decide!

But Python is better, and it is free!