

```
In [ ]:
```

```
In [27]:
```

```
import pandas as pd
pd.options.display.max_rows = 10

pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

```
Out[27]:
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN
...
886	0	2	male	27.0	0	0	13.0000	S	NaN
887	1	1	female	19.0	0	0	30.0000	S	B
888	0	3	female	NaN	1	2	23.4500	S	NaN
889	1	1	male	26.0	0	0	30.0000	C	C
890	0	3	male	32.0	0	0	7.7500	Q	NaN

891 rows × 9 columns

```
pd.options.display.min_rows = 60
```

pd.options.display.max_rows = 10
pd.options.display.min_rows = 60

```
In [26]:
```

```
titanic = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

```
Out[26]:
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN
5	0	3	male	NaN	0	0	8.4583	Q	NaN
6	0	1	male	54.0	0	0	51.8625	S	E
7	0	3	male	2.0	3	1	21.0750	S	NaN
8	1	3	female	27.0	0	2	11.1333	S	NaN

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
9	1	2	female	14.0	1	0	30.0708	C	NaN
...
881	0	3	male	33.0	0	0	7.8958	S	NaN
882	0	3	female	22.0	0	0	10.5167	S	NaN
883	0	2	male	28.0	0	0	10.5000	S	NaN
884	0	3	male	25.0	0	0	7.0500	S	NaN
885	0	3	female	39.0	0	5	29.1250	Q	NaN
886	0	2	male	27.0	0	0	13.0000	S	NaN
887	1	1	female	19.0	0	0	30.0000	S	B
888	0	3	female	NaN	1	2	23.4500	S	NaN
889	1	1	male	26.0	0	0	30.0000	C	C
890	0	3	male	32.0	0	0	7.7500	Q	NaN

891 rows × 9 columns

In [29]: `titanic.head(2)`

Out[29]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C

In [30]: `titanic.tail(2)`

Out[30]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
889	1	1	male	26.0	0	0	30.00	C	C
890	0	3	male	32.0	0	0	7.75	Q	NaN

In [31]: `titanic.columns`

Out[31]:

```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'deck'],
      dtype='object')
```

In [33]: `titanic.index`

Out[33]:

```
RangeIndex(start=0, stop=891, step=1)
```

In [34]: `titanic.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column   Non-Null Count  Dtype

```

```
0    survived    891 non-null    int64
1    pclass      891 non-null    int64
2    sex         891 non-null    object
3    age         714 non-null    float64
4    sibsp       891 non-null    int64
5    parch       891 non-null    int64
6    fare         891 non-null    float64
7    embarked    889 non-null    object
8    deck        203 non-null    object
dtypes: float64(2), int64(4), object(3)
memory usage: 62.8+ KB
```

Using .info

Non null values means that the information is provided. This means that the information is not missing. Object is a string entry. Float is a decimal value. Integers of course do not have decimals.

The Shape attribute gives the amount of rows (indexes) and columns (characteristics) respectively.

```
In [36]: titanic.shape
```

```
Out[36]: (891, 9)
```

Describe is very important.

It shows us the non missing values. It gives the summary statistics of the data set.

```
In [39]: titanic.describe()
```

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

```
In [40]: type(titanic)
```

```
Out[40]: pandas.core.frame.DataFrame
```

```
In [41]: len(titanic)
```

```
Out[41]: 891
```

Round, rounds all of the floats down to the closest zero.

```
In [42]: round(titanic,0)
```

```
Out[42]:
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.0	S	NaN
1	1	1	female	38.0	1	0	71.0	C	C
2	1	3	female	26.0	0	0	8.0	S	NaN
3	1	1	female	35.0	1	0	53.0	S	C
4	0	3	male	35.0	0	0	8.0	S	NaN
...
886	0	2	male	27.0	0	0	13.0	S	NaN
887	1	1	female	19.0	0	0	30.0	S	B
888	0	3	female	NaN	1	2	23.0	S	NaN
889	1	1	male	26.0	0	0	30.0	C	C
890	0	3	male	32.0	0	0	8.0	Q	NaN

891 rows × 9 columns

Tapping into objects.

This is how you would get the minimum age for the passenger onboard the titanic. You would tap into the attribute of the characteristic (column), then you would use the method to find the minimum.

```
In [43]: titanic.age.min()
```

```
Out[43]: 0.42
```

```
In [50]: max_fare = titanic.fare.max()  
max_fare = round(max_fare, 2)  
max_fare
```

```
Out[50]: 512.33
```

Shape and sizes.

You can use .size to find out the number of elements. The shape gives the number of indexes and columns.

```
In [52]: titanic.shape
```

```
Out[52]: (891, 9)
```

```
In [53]: titanic.size
```

```
Out[53]: 8019
```

When you use .min you can find the minimum value for every characteristic i.e column. It will later depreciate so then it must be done for valid data types.

In [56]:

```
titanic.min()
```

```
C:\Users\alonz\AppData\Local\Temp\ipykernel_6932/3376850614.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.
```

Out[56]:

```
survived      0
pclass        1
sex          female
age         0.42
sibsp        0
parch        0
fare         0.0
dtype: object
```

Utilize the tab key for autocomplete suggestions.

Press the tab key while typing to get suggestions.

By typing titanic. I get autosuggestions such as min, max, etc. This will help you to find valid processes that can be applied to your code.

Hold shift and press tab for additional suggestions.

This will allow you to be able to show you different variables that can be defined within the parenthesis.

In []:

```
titanic.sort_values()
```

Signature:

```
titanic.sort_values(  
    by,  
    axis: 'Axis' = 0,  
    ascending=True,  
    inplace: 'bool' = False,  
    kind: 'str' = 'quicksort',  
    na_position: 'str' = 'last',  
    ignore_index: 'bool' = False,  
    key: 'ValueKeyFunc' = None,  
)
```

Sorting data.

You can sort by the value of a certain characterisitc. This will make the data be shorted in ascending order by default based on one column.

In [68]:

```
titanic.sort_values(by = "age")
```

Out[68]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
803	1	3	male	0.42	0	1	8.5167	C	NaN
755	1	2	male	0.67	1	1	14.5000	S	NaN
644	1	3	female	0.75	2	1	19.2583	C	NaN
469	1	3	female	0.75	2	1	19.2583	C	NaN
78	1	2	male	0.83	0	2	29.0000	S	NaN
...
859	0	3	male	NaN	0	0	7.2292	C	NaN
863	0	3	female	NaN	8	2	69.5500	S	NaN
868	0	3	male	NaN	0	0	9.5000	S	NaN
878	0	3	male	NaN	0	0	7.8958	S	NaN
888	0	3	female	NaN	1	2	23.4500	S	NaN

891 rows × 9 columns

You can press shift + tab 3 times in order to see what parameters you can change.

In [72]:

```
titanic.sort_values(by = "age", ascending = False)
```

Out[72]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
630	1	1	male	80.0	0	0	30.0000	S	A
851	0	3	male	74.0	0	0	7.7750	S	NaN
493	0	1	male	71.0	0	0	49.5042	C	NaN
96	0	1	male	71.0	0	0	34.6542	C	A
116	0	3	male	70.5	0	0	7.7500	Q	NaN
...
859	0	3	male	NaN	0	0	7.2292	C	NaN
863	0	3	female	NaN	8	2	69.5500	S	NaN
868	0	3	male	NaN	0	0	9.5000	S	NaN
878	0	3	male	NaN	0	0	7.8958	S	NaN
888	0	3	female	NaN	1	2	23.4500	S	NaN

891 rows × 9 columns

In []:

Position-based Indexing (iloc)

```
df.iloc[row index position(s), column index position(s)]
```

row index positions		column index positions					
		-5	-4	-3	-2	-1	
		0	1	2	3	4	
			Nationality	Club	World_Champion	Height	Goals_2018
		Player					
-5	0	Lionel Messi	Argentina	FC Barcelona	False	1.70	45
-4	1	Cristiano Ronaldo	Portugal	Juventus FC	False	1.87	44
-3	2	Neymar Junior	Brasil	Paris SG	False	1.75	28
-2	3	Kylian Mbappe	France	Paris SG	True	1.78	21
-1	4	Manuel Neuer	Germany	FC Bayern	True	1.93	0

Zero-based indexing applies!

Position-based Indexing (iloc) – Example 1

df.iloc[2, 1]

		-5	0	1	-3	-2	-1
		Nationality	Club	World_Champion	Height	Goals_2018	
		Player					
-5	0	Lionel Messi	Argentina	FC Barcelona	False	1.70	45
-4	1	Cristiano Ronaldo	Portugal	Juventus FC	False	1.87	44
-3	2	Neymar Junior	Brasil	Paris SG	False	1.75	28
-2	3	Kylian Mbappe	France	Paris SG	True	1.78	21
-1	4	Manuel Neuer	Germany	FC Bayern	True	1.93	0

Output is an element („Paris SG“).

Position-based Indexing (iloc) – Example 2

```
df.iloc[[2,3], 1:]
```

from position 1 till last (inclusive)

	1	2	3	4
	Club	World_Champion	Height	Goals_2018
Player				
2	Neymar Junior	Paris SG	False	1.75
3	Kylian Mbappe	Paris SG	True	21

Output is a DataFrame.

Position-based Indexing (`iloc`) – Example 3

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

inclusive exclusive all columns

	0	1	2	3	4
	Nationality	Club	World_Champion	Height	Goals_2018
	Player				
1	Cristiano Ronaldo	Portugal	Juventus FC	False	1.87
2	Neymar Junior	Brasil	Paris SG	False	1.75

Output is a DataFrame.

Position-based Indexing (iloc) – Example 4

```
df.iloc[-2:, 3]
```

last two rows

```
          3
Player
-2  3  Kylian Mbappe  1.78
-1  4  Manuel Neuer  1.93
Name: Height, dtype: float64
```

Output is a Pandas Series.

Label-based Indexing (loc)

```
df.loc[row label(s), column label(s)]
```

column labels

Player	Nationality	Club	World_Champion	Height	Goals_2018
Lionel Messi	Argentina	FC Barcelona	False	1.70	45
Cristiano Ronaldo	Portugal	Juventus FC	False	1.87	44
Neymar Junior	Brasil	Paris SG	False	1.75	28
Kylian Mbappe	France	Paris SG	True	1.78	21
Manuel Neuer	Germany	FC Bayern	True	1.93	0

row labels

Lionel Messi, Cristiano Ronaldo, Neymar Junior, Kylian Mbappe, Manuel Neuer

Label-based Indexing (loc) – Example 1

```
df.loc["Neymar Junior", "Club"]
```



The diagram illustrates the output of the `df.loc["Neymar Junior", "Club"]` command. A blue arrow points from the code to the table, and another blue arrow points from the table to the value "Paris SG" in the "Club" column for Neymar Junior.

Player	Nationality	Club	World_Champion	Height	Goals_2018
Lionel Messi	Argentina	FC Barcelona	False	1.70	45
Cristiano Ronaldo	Portugal	Juventus FC	False	1.87	44
Neymar Junior	Brasil	Paris SG	False	1.75	28
Kylian Mbappe	France	Paris SG	True	1.78	21
Manuel Neuer	Germany	FC Bayern	True	1.93	0

Output is an element ("Paris SG").

Label-based Indexing (loc) – Example 2

```
df.loc[[“Neymar Junior”, “Kylian Mbappe”], “Club”:]
```

from column “Club” till last (inclusive)

Player	Club	World_Champion	Height	Goals_2018
Neymar Junior	Paris SG	False	1.75	28
Kylian Mbappe	Paris SG	True	1.78	21

Output is a DataFrame.

Label-based Indexing (loc) – Example 3

```
df.loc["Lionel Messi" : "Neymar Junior", :]
```

inclusive

inclusive

all columns

Player	Nationality	Club	World_Champion	Height	Goals_2018
Lionel Messi	Argentina	FC Barcelona	False	1.70	45
Cristiano Ronaldo	Portugal	Juventus FC	False	1.87	44
Neymar Junior	Brasil	Paris SG	False	1.75	28

Output is a DataFrame.

Label-based Indexing (loc) – Example 4

```
df.loc[: "Kylian Mbappe", "Height"]
```



from very first till "Kylian Mbappe" (inclusive)

```
Player
Lionel Messi      1.70
Cristiano Ronaldo 1.87
Neymar Junior     1.75
Kylian Mbappe     1.78
Name: Height, dtype: float64
```

Output is a Pandas Series.

In [1]:	<pre>import pandas as pd</pre>																																																																																																												
In [2]:	<pre>summer = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\summer.csv")</pre>																																																																																																												
In [3]:	<pre>summer</pre>																																																																																																												
In [4]:	<pre>summer</pre>																																																																																																												
Out[4]:	<table border="1"><thead><tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Athlete</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr></thead><tbody><tr><td>0</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>HAJOS, Alfred</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr><tr><td>1</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>HERSCHMANN, Otto</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr><tr><td>2</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>DRIVAS, Dimitrios</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr><tr><td>3</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>MALOKINIS, Ioannis</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr><tr><td>4</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>CHASAPIS, Spiridon</td><td>GRC</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr><tr><td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>31160</td><td>2012</td><td>London</td><td>Wrestling</td><td>JANIKOWSKI, Damian</td><td>POL</td><td>Men</td><td>Wg 84 KG</td><td>Bronze</td></tr><tr><td>31161</td><td>2012</td><td>London</td><td>Wrestling</td><td>REZAEL, Ghassen Gholamreza</td><td>IRI</td><td>Men</td><td>Wg 96 KG</td><td>Gold</td></tr><tr><td>31162</td><td>2012</td><td>London</td><td>Wrestling</td><td>TOTROV, Rustam</td><td>RUS</td><td>Men</td><td>Wg 96 KG</td><td>Silver</td></tr><tr><td>31163</td><td>2012</td><td>London</td><td>Wrestling</td><td>ALEKSANYAN, Artur</td><td>ARM</td><td>Men</td><td>Wg 96 KG</td><td>Bronze</td></tr><tr><td>31164</td><td>2012</td><td>London</td><td>Wrestling</td><td>LIDBERG, Jimmy</td><td>SWE</td><td>Men</td><td>Wg 96 KG</td><td>Bronze</td></tr></tbody></table>	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal	0	1896	Athens	Aquatics	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold	1	1896	Athens	Aquatics	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver	2	1896	Athens	Aquatics	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze	3	1896	Athens	Aquatics	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold	4	1896	Athens	Aquatics	CHASAPIS, Spiridon	GRC	Men	100M Freestyle For Sailors	Silver	...									31160	2012	London	Wrestling	JANIKOWSKI, Damian	POL	Men	Wg 84 KG	Bronze	31161	2012	London	Wrestling	REZAEL, Ghassen Gholamreza	IRI	Men	Wg 96 KG	Gold	31162	2012	London	Wrestling	TOTROV, Rustam	RUS	Men	Wg 96 KG	Silver	31163	2012	London	Wrestling	ALEKSANYAN, Artur	ARM	Men	Wg 96 KG	Bronze	31164	2012	London	Wrestling	LIDBERG, Jimmy	SWE	Men	Wg 96 KG	Bronze
Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal																																																																																																					
0	1896	Athens	Aquatics	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold																																																																																																					
1	1896	Athens	Aquatics	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver																																																																																																					
2	1896	Athens	Aquatics	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze																																																																																																					
3	1896	Athens	Aquatics	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold																																																																																																					
4	1896	Athens	Aquatics	CHASAPIS, Spiridon	GRC	Men	100M Freestyle For Sailors	Silver																																																																																																					
...																																																																																																													
31160	2012	London	Wrestling	JANIKOWSKI, Damian	POL	Men	Wg 84 KG	Bronze																																																																																																					
31161	2012	London	Wrestling	REZAEL, Ghassen Gholamreza	IRI	Men	Wg 96 KG	Gold																																																																																																					
31162	2012	London	Wrestling	TOTROV, Rustam	RUS	Men	Wg 96 KG	Silver																																																																																																					
31163	2012	London	Wrestling	ALEKSANYAN, Artur	ARM	Men	Wg 96 KG	Bronze																																																																																																					
31164	2012	London	Wrestling	LIDBERG, Jimmy	SWE	Men	Wg 96 KG	Bronze																																																																																																					
	<p>31165 rows × 9 columns</p>																																																																																																												
	<h2>Selecting indexes.</h2>																																																																																																												
	<p>You can select indexes by using .iloc and typing the index number of the row that you want to select.</p>																																																																																																												
In [8]:	<pre>summer.iloc[0]</pre>																																																																																																												
Out[8]:	<pre>Year: 1896 City: Athens Sport: Aquatics Discipline: Swimming Athlete: HAJOS, Alfred Country: HUN Gender: Men Event: 100M Freestyle Medal: Gold Name: 0, dtype: object</pre>																																																																																																												
In [22]:	<pre>summer.iloc[[0]]</pre>																																																																																																												
Out[22]:	<table border="1"><thead><tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Athlete</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr></thead><tbody><tr><td>0</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>HAJOS, Alfred</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr></tbody></table>	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal	0	1896	Athens	Aquatics	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold																																																																																										
Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal																																																																																																					
0	1896	Athens	Aquatics	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold																																																																																																					
	<h2>You can enter into a specific cell.</h2>																																																																																																												
	<p>All you do is select the index number first into the brackets. You must use .iloc. For the second number you put the column number that you want to select. Remember the count for python starts at zero.</p>																																																																																																												
In [25]:	<pre>summer.head(3)</pre>																																																																																																												
Out[25]:	<table border="1"><thead><tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Athlete</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr></thead><tbody><tr><td>0</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>HAJOS, Alfred</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr><tr><td>1</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>HERSCHMANN, Otto</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr><tr><td>2</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>DRIVAS, Dimitrios</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr></tbody></table>	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal	0	1896	Athens	Aquatics	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold	1	1896	Athens	Aquatics	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver	2	1896	Athens	Aquatics	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze																																																																								
Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal																																																																																																					
0	1896	Athens	Aquatics	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold																																																																																																					
1	1896	Athens	Aquatics	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver																																																																																																					
2	1896	Athens	Aquatics	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze																																																																																																					
In [31]:	<pre># You can select the city of the first year zero using [index_number, column_number]</pre>																																																																																																												
Out[31]:	<pre>'Athens'</pre>																																																																																																												
	<p>You can enter the index that you want to observe. You can also select a range of characteristics that you want to observe as well.</p>																																																																																																												
In [34]:	<pre>summer.iloc[0, 0:4]</pre>																																																																																																												
Out[34]:	<pre>Year: 1896 City: Athens Sport: Aquatics Discipline: Swimming Name: 0, dtype: object</pre>																																																																																																												
In [36]:	<pre>summer.iloc[1:4, 3:1]</pre>																																																																																																												
Out[36]:	<table border="1"><thead><tr><th>Discipline</th><th>Athlete</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr></thead><tbody><tr><td>1</td><td>Swimming</td><td>HERSCHMANN, Otto</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr><tr><td>2</td><td>Swimming</td><td>DRIVAS, Dimitrios</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr><tr><td>3</td><td>Swimming</td><td>MALOKINIS, Ioannis</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr></tbody></table>	Discipline	Athlete	Country	Gender	Event	Medal	1	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver	2	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze	3	Swimming	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold																																																																																	
Discipline	Athlete	Country	Gender	Event	Medal																																																																																																								
1	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver																																																																																																							
2	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze																																																																																																							
3	Swimming	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold																																																																																																							
	<h2>You can use a list in the column section of iloc in order to select non sequential characteristics.</h2>																																																																																																												
In [37]:	<pre>summer.head(3)</pre>																																																																																																												
Out[37]:	<table border="1"><thead><tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Athlete</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr></thead><tbody><tr><td>0</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>HAJOS, Alfred</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr><tr><td>1</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>HERSCHMANN, Otto</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr><tr><td>2</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>DRIVAS, Dimitrios</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr></tbody></table>	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal	0	1896	Athens	Aquatics	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold	1	1896	Athens	Aquatics	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver	2	1896	Athens	Aquatics	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze																																																																								
Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal																																																																																																					
0	1896	Athens	Aquatics	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold																																																																																																					
1	1896	Athens	Aquatics	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver																																																																																																					
2	1896	Athens	Aquatics	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze																																																																																																					
In [44]:	<pre>events = [0, 1, 7, 8] summer.iloc[0:2, events]</pre>																																																																																																												
Out[44]:	<pre>Year City Event Medal 0 1896 Athens 100M Freestyle Gold 1 1896 Athens 100M Freestyle Silver</pre>																																																																																																												
In [52]:	<pre># Here is another way to write the same code. summer.iloc[0:3, [0, 1, 7, 8]]</pre>																																																																																																												
Out[52]:	<table border="1"><thead><tr><th>Year</th><th>City</th><th>Event</th><th>Medal</th></tr></thead><tbody><tr><td>0</td><td>1896</td><td>Athens</td><td>100M Freestyle</td><td>Gold</td></tr><tr><td>1</td><td>1896</td><td>Athens</td><td>100M Freestyle</td><td>Silver</td></tr><tr><td>2</td><td>1896</td><td>Athens</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr></tbody></table>	Year	City	Event	Medal	0	1896	Athens	100M Freestyle	Gold	1	1896	Athens	100M Freestyle	Silver	2	1896	Athens	100M Freestyle For Sailors	Bronze																																																																																									
Year	City	Event	Medal																																																																																																										
0	1896	Athens	100M Freestyle	Gold																																																																																																									
1	1896	Athens	100M Freestyle	Silver																																																																																																									
2	1896	Athens	100M Freestyle For Sailors	Bronze																																																																																																									
	<p>Using iloc can be used instead of using the column format by selecting all indexes using ":" and typing the particular field of interest.</p>																																																																																																												
In [51]:	<pre>summer.iloc[:, 4].equals(summer["Athlete"])</pre>																																																																																																												
Out[51]:	<pre>True</pre>																																																																																																												
In [62]:	<pre>summer.iloc[:, 4]</pre>																																																																																																												
Out[62]:	<pre>0 HAJOS, Alfred 1 HERSCHEMANN, Otto 2 DRIVAS, Dimitrios 3 MALOKINIS, Ioannis 4 CHASAPIS, Spiridon ... 31160 JANIKOWSKI, Damian 31161 REZAEL, Ghassen Gholamreza 31162 TOTROV, Rustam 31163 ALEXSANIAN, Artur 31164 LIDBERG, Jimmy Name: Athlete, Length: 31165, dtype: object</pre>																																																																																																												
In [63]:	<pre>type(summer.iloc[:, 4])</pre>																																																																																																												
Out[63]:	<pre>pandas.core.series.Series</pre>																																																																																																												
	<p>To make iloc a data frame you must put the column into a list. Even if the characteristic has one value.</p>																																																																																																												
In [65]:	<pre>summer.iloc[:, [4]]</pre>																																																																																																												
Out[65]:	<table border="1"><thead><tr><th>Athlete</th></tr></thead><tbody><tr><td>0 HAJOS, Alfred</td></tr><tr><td>1 HERSCHEMANN, Otto</td></tr><tr><td>2 DRIVAS, Dimitrios</td></tr><tr><td>3 MALOKINIS, Ioannis</td></tr><tr><td>4 CHASAPIS, Spiridon</td></tr><tr><td>...</td></tr><tr><td>31160 JANIKOWSKI, Damian</td></tr><tr><td>31161 REZAEL, Ghassen Gholamreza</td></tr><tr><td>31162 TOTROV, Rustam</td></tr><tr><td>31163 ALEXSANIAN, Artur</td></tr><tr><td>31164 LIDBERG, Jimmy</td></tr></tbody></table>	Athlete	0 HAJOS, Alfred	1 HERSCHEMANN, Otto	2 DRIVAS, Dimitrios	3 MALOKINIS, Ioannis	4 CHASAPIS, Spiridon	...	31160 JANIKOWSKI, Damian	31161 REZAEL, Ghassen Gholamreza	31162 TOTROV, Rustam	31163 ALEXSANIAN, Artur	31164 LIDBERG, Jimmy																																																																																																
Athlete																																																																																																													
0 HAJOS, Alfred																																																																																																													
1 HERSCHEMANN, Otto																																																																																																													
2 DRIVAS, Dimitrios																																																																																																													
3 MALOKINIS, Ioannis																																																																																																													
4 CHASAPIS, Spiridon																																																																																																													
...																																																																																																													
31160 JANIKOWSKI, Damian																																																																																																													
31161 REZAEL, Ghassen Gholamreza																																																																																																													
31162 TOTROV, Rustam																																																																																																													
31163 ALEXSANIAN, Artur																																																																																																													
31164 LIDBERG, Jimmy																																																																																																													
	<p>31165 rows × 1 columns</p>																																																																																																												
In [66]:	<pre>type(summer.iloc[:, [4]])</pre>																																																																																																												
Out[66]:	<pre>pandas.core.frame.DataFrame</pre>																																																																																																												
In [68]:	<pre>summer[["Athlete"]]</pre>																																																																																																												
Out[68]:	<table border="1"><thead><tr><th>Athlete</th></tr></thead><tbody><tr><td>0 HAJOS, Alfred</td></tr><tr><td>1 HERSCHEMANN, Otto</td></tr><tr><td>2 DRIVAS, Dimitrios</td></tr><tr><td>3 MALOKINIS, Ioannis</td></tr><tr><td>4 CHASAPIS, Spiridon</td></tr><tr><td>...</td></tr><tr><td>31160 JANIKOWSKI, Damian</td></tr><tr><td>31161 REZAEL, Ghassen Gholamreza</td></tr><tr><td>31162 TOTROV, Rustam</td></tr><tr><td>31163 ALEXSANIAN, Artur</td></tr><tr><td>31164 LIDBERG, Jimmy</td></tr></tbody></table>	Athlete	0 HAJOS, Alfred	1 HERSCHEMANN, Otto	2 DRIVAS, Dimitrios	3 MALOKINIS, Ioannis	4 CHASAPIS, Spiridon	...	31160 JANIKOWSKI, Damian	31161 REZAEL, Ghassen Gholamreza	31162 TOTROV, Rustam	31163 ALEXSANIAN, Artur	31164 LIDBERG, Jimmy																																																																																																
Athlete																																																																																																													
0 HAJOS, Alfred																																																																																																													
1 HERSCHEMANN, Otto																																																																																																													
2 DRIVAS, Dimitrios																																																																																																													
3 MALOKINIS, Ioannis																																																																																																													
4 CHASAPIS, Spiridon																																																																																																													
...																																																																																																													
31160 JANIKOWSKI, Damian																																																																																																													
31161 REZAEL, Ghassen Gholamreza																																																																																																													
31162 TOTROV, Rustam																																																																																																													
31163 ALEXSANIAN, Artur																																																																																																													
31164 LIDBERG, Jimmy																																																																																																													
In [69]:	<pre>summer[["Athlete"]]</pre>																																																																																																												
Out[69]:	<pre>0 HAJOS, Alfred 1 HERSCHEMANN, Otto 2 DRIVAS, Dimitrios 3 MALOKINIS, Ioannis 4 CHASAPIS, Spiridon ... 31160 JANIKOWSKI, Damian 31161 REZAEL, Ghassen Gholamreza 31162 TOTROV, Rustam 31163 ALEXSANIAN, Artur 31164 LIDBERG, Jimmy Name: Athlete, Length: 31165, dtype: object</pre>																																																																																																												
	<p>You can find the data on multiple indexes by putting the indexes in a list.</p>																																																																																																												
In [70]:	<pre>summer.loc[[0, 1, 7, 8]]</pre>																																																																																																												
Out[70]:	<table border="1"><thead><tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Athlete</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr></thead><tbody><tr><td>0</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>HAJOS, Alfred</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr><tr><td>1</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>HERSCHEMANN, Otto</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr><tr><td>7</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>DRIVAS, Dimitrios</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr><tr><td>8</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>MALOKINIS, Ioannis</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr></tbody></table>	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal	0	1896	Athens	Aquatics	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold	1	1896	Athens	Aquatics	HERSCHEMANN, Otto	AUT	Men	100M Freestyle	Silver	7	1896	Athens	Aquatics	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze	8	1896	Athens	Aquatics	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold																																																															
Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal																																																																																																					
0	1896	Athens	Aquatics	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold																																																																																																					
1	1896	Athens	Aquatics	HERSCHEMANN, Otto	AUT	Men	100M Freestyle	Silver																																																																																																					
7	1896	Athens	Aquatics	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze																																																																																																					
8	1896	Athens	Aquatics	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold																																																																																																					
	<p>You can use .loc after .iloc.</p>																																																																																																												
In [71]:	<pre># For example we can use the table to find Michael Phelps' first medal.</pre>																																																																																																												
	<p>Note that the "i" can be used to be understood as integer of the index.</p>																																																																																																												
In [72]:	<pre>summer[["Athlete"]]</pre>																																																																																																												
Out[72]:	<pre>0 HAJOS, Alfred 1 HERSCHEMANN, Otto 2 DRIVAS, Dimitrios 3 MALOKINIS, Ioannis 4 CHASAPIS, Spiridon ... 31160 JANIKOWSKI, Damian 31161 REZAEL, Ghassen Gholamreza 31162 TOTROV, Rustam 31163 ALEXSANIAN, Artur 31164 LIDBERG, Jimmy Name: Athlete, Length: 31165, dtype: object</pre>																																																																																																												
In [73]:	<pre>Note that summer.City would give the same result.</pre>																																																																																																												
In [74]:	<pre>type(summer["City"])</pre>																																																																																																												
Out[74]:	<pre>pandas.core.series.Series</pre>																																																																																																												
	<p>To have multiple columns you must use double brackets.</p>																																																																																																												
	<p>Using dataframes never data values via a DataFrame for a datafram. In terms of use double brackets and for this is a good idea. to get into the habit of</p>																																																																																																												
In [75]:	<pre>summer[["Year", "City", "Gender", "Medal"]]</pre>																																																																																																												
Out[75]:	<table border="1"><thead><tr><th>Year</th><th>City</th><th>Gender</th><th>Medal</th></tr></thead><tbody><tr><td>0</td><td>1896</td><td>Athens</td><td>Gold</td></tr><tr><td>1</td><td>1896</td><td>Athens</td><td>Silver</td></tr><tr><td>2</td><td>1896</td><td>Athens</td><td>Bronze</td></tr><tr><td>3</td><td>1896</td><td>Athens</td><td>Silver</td></tr><tr><td>4</td><td>1896</td><td>Athens</td><td>Gold</td></tr><tr><td>...</td><td></td><td></td><td></td></tr><tr><td>31160</td><td>2012</td><td>London</td><td>Bronze</td></tr><tr><td>31161</td><td>2012</td><td>London</td><td>Gold</td></tr><tr><td>31162</td><td>2012</td><td>London</td><td>Silver</td></tr><tr><td>31163</td><td>2012</td><td>London</td><td>Bronze</td></tr><tr><td>31164</td><td>2012</td><td>London</td><td>Bronze</td></tr></tbody></table>	Year	City	Gender	Medal	0	1896	Athens	Gold	1	1896	Athens	Silver	2	1896	Athens	Bronze	3	1896	Athens	Silver	4	1896	Athens	Gold	...				31160	2012	London	Bronze	31161	2012	London	Gold	31162	2012	London	Silver	31163	2012	London	Bronze	31164	2012	London	Bronze																																																												
Year	City	Gender	Medal																																																																																																										
0	1896	Athens	Gold																																																																																																										
1	1896	Athens	Silver																																																																																																										
2	1896	Athens	Bronze																																																																																																										
3	1896	Athens	Silver																																																																																																										
4	1896	Athens	Gold																																																																																																										
...																																																																																																													
31160	2012	London	Bronze																																																																																																										
31161	2012	London	Gold																																																																																																										
31162	2012	London	Silver																																																																																																										
31163	2012	London	Bronze																																																																																																										
31164	2012	London	Bronze																																																																																																										
	<p>31165 rows × 4 columns</p>																																																																																																												
	<p>Very important: you can change the index for a row from the default numeric value: you can change the index for a row from the default numeric value.</p>																																																																																																												
	<p>This is done by specifying index_col = "column_name" after the import.</p>																																																																																																												
	<p>The index has changed to "Athlete" in this workbook.</p>																																																																																																												
In [76]:	<pre>summer = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\summer.csv", index_col = "Athlete")</pre>																																																																																																												
In [77]:	<pre>summer</pre>																																																																																																												
Out[77]:	<table border="1"><thead><tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Athlete</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr></thead><tbody><tr><td>0</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>HAJOS, Alfred</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr><tr><td>1</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>HERSCHEMANN, Otto</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr><tr><td>2</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>DRIVAS, Dimitrios</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr><tr><td>3</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>MALOKINIS, Ioannis</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr><tr><td>4</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>CHASAPIS, Spiridon</td><td>GRC</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr><tr><td>...</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>31160</td><td>2012</td><td>London</td><td>Wrestling</td><td>JANIKOWSKI, Damian</td><td>POL</td><td>Men</td><td>Wg 84 KG</td><td>Bronze</td></tr><tr><td>31161</td><td>2012</td><td>London</td><td>Wrestling</td><td>REZAEL, Ghassen Gholamreza</td><td>IRI</td><td>Men</td><td>Wg 96 KG</td><td>Gold</td></tr><tr><td>31162</td><td>2012</td><td>London</td><td>Wrestling</td><td>TOTROV, Rustam</td><td>RUS</td><td>Men</td><td>Wg 96 KG</td><td>Silver</td></tr><tr><td>31163</td><td>2012</td><td>London</td><td>Wrestling</td><td>ALEXSANIAN, Artur</td><td>ARM</td><td>Men</td><td>Wg 96 KG</td><td>Bronze</td></tr><tr><td>31164</td><td>2012</td><td>London</td><td>Wrestling</td><td>LIDBERG, Jimmy</td><td>SWE</td><td>Men</td><td>Wg 96 KG</td><td>Bronze</td></tr></tbody></table>	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal	0	1896	Athens	Aquatics	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold	1	1896	Athens	Aquatics	HERSCHEMANN, Otto	AUT	Men	100M Freestyle	Silver	2	1896	Athens	Aquatics	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze	3	1896	Athens	Aquatics	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold	4	1896	Athens	Aquatics	CHASAPIS, Spiridon	GRC	Men	100M Freestyle For Sailors	Silver	...									31160	2012	London	Wrestling	JANIKOWSKI, Damian	POL	Men	Wg 84 KG	Bronze	31161	2012	London	Wrestling	REZAEL, Ghassen Gholamreza	IRI	Men	Wg 96 KG	Gold	31162	2012	London	Wrestling	TOTROV, Rustam	RUS	Men	Wg 96 KG	Silver	31163	2012	London	Wrestling	ALEXSANIAN, Artur	ARM	Men	Wg 96 KG	Bronze	31164	2012	London	Wrestling	LIDBERG, Jimmy	SWE	Men	Wg 96 KG	Bronze
Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal																																																																																																					
0	1896	Athens	Aquatics	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold																																																																																																					
1	1896	Athens	Aquatics	HERSCHEMANN, Otto	AUT	Men	100M Freestyle	Silver																																																																																																					
2	1896	Athens	Aquatics	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze																																																																																																					
3	1896	Athens	Aquatics	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold																																																																																																					
4	1896	Athens	Aquatics	CHASAPIS, Spiridon	GRC	Men	100M Freestyle For Sailors	Silver																																																																																																					
...																																																																																																													
31160	2012	London	Wrestling	JANIKOWSKI, Damian	POL	Men	Wg 84 KG	Bronze																																																																																																					
31161	2012	London	Wrestling	REZAEL, Ghassen Gholamreza	IRI	Men	Wg 96 KG	Gold																																																																																																					
31162	2012	London	Wrestling	TOTROV, Rustam	RUS	Men	Wg 96 KG	Silver																																																																																																					
31163	2012	London	Wrestling	ALEXSANIAN, Artur	ARM	Men	Wg 96 KG	Bronze																																																																																																					
31164	2012	London	Wrestling	LIDBERG, Jimmy	SWE	Men	Wg 96 KG	Bronze																																																																																																					
	<p>31165 rows × 9 columns</p>																																																																																																												
	<p>Now we can see the athlete serves as the root of the chart. We can see which athlete won which medal in every year.</p>																																																																																																												
	<p>Now we can see the athlete serves as the root of the chart. We can see which athlete won which medal in every year.</p>																																																																																																												
	<p>Important.</p>																																																																																																												
	<p>When finding an index based off of text and not a number you will do so by using .loc.</p>																																																																																																												
In [78]:	<pre># Use .loc for a string based index only.</pre>																																																																																																												
Out[78]:	<pre># use .loc["HELIOS, Michael"]</pre>																																																																																																												
	<p>You can still find the index based off of the number of its order. The index is however no longer displayed.</p>																																																																																																												
In [79]:	<pre>summer[["Athlete"]]</pre>																																																																																																												
Out[79]:	<pre>0 HAJOS, Alfred 1 HERSCHEMANN, Otto 2 DRIVAS, Dimitrios 3 MALOKINIS, Ioannis 4 CHASAPIS, Spiridon ... 31160 JANIKOWSKI, Damian 31161 REZAEL, Ghassen Gholamreza 31162 TOTROV, Rustam 31163 ALEXSANIAN, Artur 31164 LIDBERG, Jimmy Name: Athlete, Length: 31165, dtype: object</pre>																																																																																																												
	<p>Note that the "index_col" must be changed to the list of names as shown before.</p>																																																																																																												
In [80]:	<pre>type(summer[["Athlete"]])</pre>																																																																																																												
Out[80]:	<pre>pandas.core.frame.DataFrame</pre>																																																																																																												
In [81]:	<pre>summer[["Athlete"]]</pre>																																																																																																												
Out[81]:	<pre>0 HAJOS, Alfred 1 HERSCHEMANN, Otto 2 DRIVAS, Dimitrios 3 MALOKINIS, Ioannis 4 CHASAPIS, Spiridon ... 31160 JANIKOWSKI, Damian 31161 REZAEL, Ghassen Gholamreza 31162 TOTROV, Rustam 31163 ALEXSANIAN, Artur 31164 LIDBERG, Jimmy Name: Athlete, Length: 31165, dtype: object</pre>																																																																																																												
	<p>You can use .loc to find the first entries in a certain list, in conjunction with loc.</p>																																																																																																												
In [82]:	<pre>summer.loc[[0, 1, 7, 8]]</pre>																																																																																																												
Out[82]:	<table border="1"><thead><tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Athlete</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr></thead><tbody><tr><td>0</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>HAJOS, Alfred</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr><tr><td>1</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>HERSCHEMANN, Otto</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr><tr><td>7</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>DRIVAS, Dimitrios</td></tr></tbody></table>	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal	0	1896	Athens	Aquatics	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold	1	1896	Athens	Aquatics	HERSCHEMANN, Otto	AUT	Men	100M Freestyle	Silver	7	1896	Athens	Aquatics	DRIVAS, Dimitrios																																																																												
Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal																																																																																																					
0	1896	Athens	Aquatics	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold																																																																																																					
1	1896	Athens	Aquatics	HERSCHEMANN, Otto	AUT	Men	100M Freestyle	Silver																																																																																																					
7	1896	Athens	Aquatics	DRIVAS, Dimitrios																																																																																																									

```
In [1]: import pandas as pd

In [2]: titanic = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")

In [3]: titanic.head()

Out[3]: survived pclass sex age sibsp parch fare embarked deck
0 0 3 male 22.0 1 0 7.2500 S NaN
1 1 1 female 38.0 1 0 71.2833 C C
2 0 1 3 female 26.0 0 0 7.9250 S NaN
3 1 1 female 35.0 1 0 53.1000 S C
4 0 3 male 35.0 0 0 8.0500 S NaN

In [5]: titanic.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries: 0 to 890
Data columns (total 9 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   survived    891 non-null   int64  
 1   pclass      891 non-null   int64  
 2   sex         891 non-null   object  
 3   age         891 non-null   float64
 4   sibsp       891 non-null   int64  
 5   parch       891 non-null   int64  
 6   fare        889 non-null   float64
 7   embarked    889 non-null   object  
 8   deck        203 non-null   object  
dtypes: float64(2), int64(4), object(3)
memory usage: 62.8+ KB

In [9]: titanic.describe()

Out[9]: survived pclass age sibsp parch fare
count 891.000000 891.000000 714.000000 891.000000 891.000000
mean 0.383838 2.308642 29.699118 0.523008 0.381594 32.204208
std 0.466592 0.836071 14.526497 1.102743 0.806057 49.693429
min 0.000000 1.000000 0.420000 0.000000 0.000000 0.000000
25% 0.000000 2.000000 20.125000 0.000000 0.000000 7.910400
50% 0.000000 3.000000 28.000000 0.000000 0.000000 14.454200
75% 1.000000 3.000000 38.000000 1.000000 0.000000 31.000000
max 1.000000 3.000000 80.000000 8.000000 6.000000 512.329200

Series data.

In [10]: type(titanic["age"])

Out[10]: pandas.core.series.Series

In [11]: titanic["age"].equals(titanic.age)

Out[11]: True

In [12]: age = titanic["age"]

In [16]: age.head(n=3)

Out[16]: 0 22.0
1 38.0
2 26.0
Name: age, dtype: float64

In [14]: age.tail()

Out[14]: 886 27.0
887 19.0
888 NaN
889 26.0
890 32.0
Name: age, dtype: float64

In [15]: age.dtype

Out[15]: dtype('float64')

Info does not work for series data. It must be a dataframe. Here is an example of the error message.

In [17]: age.info()

AttributeError: 'Series' object has no attribute 'info'

In [18]: age.shape

Out[18]: (891, 9)

In [19]: len(age)

Out[19]: 891

In [20]: age.index

Out[20]: RangeIndex(start=0, stop=891, step=1)

You can describe data within a series and get statistics.

In [21]: age.describe()

Out[21]: count 714.000000
mean 29.699118
std 14.526497
min 0.420000
25% 20.125000
50% 28.000000
75% 38.000000
max 80.000000
Name: age, dtype: float64

Series can get all data types except.info()

In [22]: age.sum()

Out[22]: 21205.17

In [23]: age.mean()

Out[23]: 29.69911764705882

In [24]: age.std()

Out[24]: 14.526497332334044

In [25]: age.min()

Out[25]: 0.42

In [26]: age.max()

Out[26]: 80.0

In [27]: age.median()

Out[27]: 28.0

In [29]: age.mode()

Out[29]: 0 24.0
dtype: float64
For the sum method you need to have the default skip empty value enabled. If it is not enabled then the calculation will be wrong.

In [30]: age.sum(skipna = False)

Out[30]: nan

In [31]: age.sum(skipna = True)

Out[31]: 21205.17
Remember press the shift key within parenthesis to see the explanation of what can go within brackets.

In [34]: # Unique shows how many non repeated values there are.
age.unique()

Out[34]: array([ 22.,  38.,  26.,  35.,  nan, 54.,  2.,  27.,  14.,  18.,  19.,  40.,  29.,  53.,  28.5,  5.,  21.,  42.,  15.,  3.,  7.,  16.,  25.,  0.83,  30.,  33.,  23.,  45.,  17.,  32.,  51.,  55.5,  40.5,  44.,  1.,  61.,  52.,  32.5,  9.,  36.5,  45.5,  20.5,  62.,  41.,  50.,  63.,  23.5,  53.,  57.,  80.,  70.,  24.5,  6.,  0.67,  30.5,  0.42,  34.5,  74. ])
Using the length in conjunction with the unique values we can find out how many unique values we have.

In [35]: len(age.unique())

Out[35]: 89

Get a real count of the number of unique values.

See how many unique values are here by using the .nunique() This will not include the null values by default.

In [38]: age.nunique()

Out[38]: 88
Use shift + tab to figure out the functionality of functions/methods inside of parenthesis. Hold the shift key.

You can get a count of how often a value is recorded.

You can do this by using .value_counts() Using dropna = False will show all of the missing values.

In [42]: age.value_counts(dropna = False)

Out[42]: NaN 177
24.00 30
22.00 27
18.00 26
28.00 25
...
36.50 1
55.50 1
0.92 1
23.50 1
74.00 1
Name: age, Length: 89, dtype: int64
You can find out the percent frequency. The percent is not calculating as these values are not divided by 100. You just have to make normalize = True

In [45]: age.value_counts(dropna = True, sort = True, ascending = False, normalize = True)

Out[45]: 24.00 0.042017
22.00 0.037915
18.00 0.036415
19.00 0.035014
28.00 0.035014
...
36.50 0.001401
55.50 0.001401
0.92 0.001401
23.50 0.001401
74.00 0.001401
Name: age, Length: 88, dtype: float64
Bins can be used to put data into categories. Meaning the subset of numbers are divided into intervals.

In [44]: age.value_counts(bins = 5)

Out[44]: (32.252, 40.168] 186
(40.168, 48.084] 100
(48.084, 64.098] 69
(64.098, 80.012] 51
Name: age, dtype: float64
The unique method is not possible to find the range using ptp for pandas. You would have to use numpy. However this process is so rudimentary it is not even possible to get it to work.

We are only using pandas with the original titanic, including the regular age.

To find the range just subtract the minimum from the maximum.

In [62]: age.describe()

Out[62]: count 724.699018
mean 29.526497
std 14.526497
min 0.420000
25% 20.125000
50% 28.000000
75% 38.000000
max 80.000000
Name: age, dtype: float64
The minimum value is the first one to show up in the alphabet.

In [66]: age.letter = age.letter.str.lower()

Out[66]: letter
0 22.0
1 38.0
2 26.0
Name: letter, dtype: float64
The letter method is not possible to find the range using ptp for pandas. You would have to use numpy. However this process is so rudimentary it is not even possible to get it to work.

The copy() method

You can use the copy() method to edit a derivative of the data and not the original data. If you do not use the copy method you will contaminate the data you imported.

In [12]: age = titanic.age.copy()

Out[12]: age
0 22.0
1 38.0
2 26.0
Name: age, dtype: float64
age.letter = 29
age.letter
0 22.0
1 38.0
2 26.0
Name: age, dtype: float64
If you do not use .copy() the main data at the start as well, you would contaminate the data i.e. age = titanic.age() Then using age[0] = 29 would

Sorting and introduction to the inplace-parameters.

In [14]: import pandas as pd

Now we are going to make a dictionary. There will be seven entries for the seven days of the week. We will notice that python will automatically set the values in order numerically.

In [19]: dic = {1:10, 2:25, 3:30, 4:35, 5:2, 6:1, 7:None}

Out[19]: {1: 10, 2: 25, 3: 30, 4: 35, 5: 2, 6: 1, 7: None}
For the time being it is not possible to find the range using ptp for pandas. You would have to use numpy. However this process is so rudimentary it is not even possible to get it to work.

Convert a dictionary into a pandas series.

In [19]: sales = pd.Series(dic)

Out[19]: sales
0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
dic is made if the original dictionary. The sort parameter is applied to a copy of the dictionary that is imported.

In [14]: sales.unique()

Out[14]: 25.0
30.0
36.0
30.0
20.0
25.0
20.0
Name: sales, dtype: float64
The default value makes the number sort itself via ascending order.

In [10]: sales.sort_index(ascending = True, inplace = False)

Out[10]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
sales.sort_index(ascending = False, inplace = False)
sales.sort_index(ascending = False, inplace = True, na_position = "last")
sales.sort_index(ascending = False, inplace = True, na_position = "first")
This is what happens when we turn ascending order off.

In [12]: sales.sort_value(ascending = False, inplace = True, na_position = "first")

Out[12]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
This is what happens when we turn ascending order off.

In [16]: sales.sort_value(ascending = False, inplace = True, na_position = "first")

Out[16]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0
2 36.0
3 30.0
4 20.0
5 25.0
6 20.0
Name: sales, dtype: float64
The sort_index is different than sort_value index is the key value are the entries of the dictionary.

You can sort by values and not by the index. In this way you will be able to determine where the non null values are placed. By pressing shift + tab you can determine that the two options are first and last.

In [15]: sales.sort_value(ascending = False, inplace = True, na_position = "last")

Out[15]: 0 25.0
1 30.0

```

In [1]:	import pandas as pd																																																																																																												
In [3]:	summer = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NB\Video_Lecture_NB\summer.csv")																																																																																																												
In [4]:	summer.head()																																																																																																												
Out[4]:	<table border="1"> <thead> <tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Athlete</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>0</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HAJOS, Alfred</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>1</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HERSCHMANN, Otto</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>2</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>DRIVAS, Dimitrios</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>3</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>MALOKNIS, Ioannis</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>4</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>CHASAPIS, Spiridon</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> </tbody> </table>	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal	0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold	1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver	2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze	3	1896	Athens	Aquatics	Swimming	MALOKNIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold	4	1896	Athens	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver																																																	
Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal																																																																																																					
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold																																																																																																				
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver																																																																																																				
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze																																																																																																				
3	1896	Athens	Aquatics	Swimming	MALOKNIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold																																																																																																				
4	1896	Athens	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver																																																																																																				
We can change the index to the name of the athlete instead of the default numerical value.																																																																																																													
In [13]:	summer = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NB\Video_Lecture_NB\summer.csv", index_col = "Athlete")																																																																																																												
In [14]:	summer.tail()																																																																																																												
Out[14]:	<table border="1"> <thead> <tr><th>Athlete</th><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>JANIKOWSKI, Damian</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>POL</td><td>Men</td><td>Wg 84 KG</td><td>Bronze</td></tr> <tr><td>REZAEL, Ghader</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>IRI</td><td>Men</td><td>Wg 96 KG</td><td>Gold</td></tr> <tr><td>TOTROV, Rustam</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>RUS</td><td>Men</td><td>Wg 96 KG</td><td>Silver</td></tr> <tr><td>ALEKSANYAN, Artur</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>ARM</td><td>Men</td><td>Wg 96 KG</td><td>Bronze</td></tr> <tr><td>LIDBERG, Jimmy</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>SWE</td><td>Men</td><td>Wg 96 KG</td><td>Bronze</td></tr> </tbody> </table>	Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal	JANIKOWSKI, Damian	2012	London	Wrestling	Wrestling Freestyle	POL	Men	Wg 84 KG	Bronze	REZAEL, Ghader	2012	London	Wrestling	Wrestling Freestyle	IRI	Men	Wg 96 KG	Gold	TOTROV, Rustam	2012	London	Wrestling	Wrestling Freestyle	RUS	Men	Wg 96 KG	Silver	ALEKSANYAN, Artur	2012	London	Wrestling	Wrestling Freestyle	ARM	Men	Wg 96 KG	Bronze	LIDBERG, Jimmy	2012	London	Wrestling	Wrestling Freestyle	SWE	Men	Wg 96 KG	Bronze																																																						
Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																																																																					
JANIKOWSKI, Damian	2012	London	Wrestling	Wrestling Freestyle	POL	Men	Wg 84 KG	Bronze																																																																																																					
REZAEL, Ghader	2012	London	Wrestling	Wrestling Freestyle	IRI	Men	Wg 96 KG	Gold																																																																																																					
TOTROV, Rustam	2012	London	Wrestling	Wrestling Freestyle	RUS	Men	Wg 96 KG	Silver																																																																																																					
ALEKSANYAN, Artur	2012	London	Wrestling	Wrestling Freestyle	ARM	Men	Wg 96 KG	Bronze																																																																																																					
LIDBERG, Jimmy	2012	London	Wrestling	Wrestling Freestyle	SWE	Men	Wg 96 KG	Bronze																																																																																																					
In [15]:	summer.info()																																																																																																												
Out[15]:	<pre><class 'pandas.core.frame.DataFrame'> Index: 31165 entries, HAJOS, Alfred to LIDBERG, Jimmy Data columns (total 9 columns): # Column Non-Null Count Dtype --- ~~~~~ 0 Year 31165 non-null int64 1 City 31165 non-null object 2 Sport 31165 non-null object 3 Discipline 31165 non-null object 4 Country 31161 non-null object 5 Gender 31165 non-null object 6 Event 31165 non-null object 7 Medal 31165 non-null object memory usage: 2.1+ MB</pre>																																																																																																												
In [16]:	summer.index																																																																																																												
Out[16]:	<pre>Index(['HAJOS', 'ALFRED', 'HERSCHMANN', 'OTTO', 'DRIVAS', 'DIMITRIOS', 'MALOKNIS', 'IOANNIS', 'CHASAPIS', 'SPIRIDON', 'CHOROPHAS', 'EFSTATHIOS', 'NEUMANN', 'PAUL', 'REZAEL', 'GHADER', 'TOTROV', 'RUSTAM', 'ALEKSANYAN', 'ARTUR', 'LIDBERG', 'JIMMY'], dtype='object', name='Athlete', length=31165)</pre>																																																																																																												
summer.columns																																																																																																													
In [17]:	type(summer.columns)																																																																																																												
Out[17]:	pandas.core.indexes.base.Index																																																																																																												
Note that "axes" is used to refer to the index instead of axis.																																																																																																													
In [21]:	summer.axes																																																																																																												
Out[21]:	<pre>[Index('Athlete', 'Year', 'City', 'Sport'), dtype='object']</pre>																																																																																																												
Remember you can slice characteristics or columns using slices.																																																																																																													
In [22]:	summer.columns[1:3]																																																																																																												
Out[22]:	Index(['Year', 'City', 'Sport'], dtype='object')																																																																																																												
You can slice indexes as well using slices.																																																																																																													
In [24]:	#You can select indexes using their number. summer.index[0]																																																																																																												
Out[24]:	'HAJOS, Alfred'																																																																																																												
In [25]:	# You can pick the last item using -1. summer.index[-1]																																																																																																												
Out[25]:	'LIDBERG, Jimmy'																																																																																																												
.unique can be used to show if the data or index contains values that have only been used once.																																																																																																													
The unique can be used to find out if their is repeated values. If the answer is true then the entry is only found once inside of the data.																																																																																																													
In [26]:	summer.index.is_unique																																																																																																												
Out[26]:	False																																																																																																												
.get_loc can find the index number of a string that you input.																																																																																																													
In [28]:	#This is one of the last names in the data. summer.index.get_loc("ALEKSANYAN, Artur")																																																																																																												
Out[28]:	31163																																																																																																												
In [30]:	#This is one of the first names in the data. summer.index.get_loc("DRIVAS, Dimitrios")																																																																																																												
Out[30]:	2																																																																																																												
Changing Row Index Labels																																																																																																													
In [31]:	import pandas as pd																																																																																																												
In [33]:	summer = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NB\Video_Lecture_NB\summer.csv", index_col = "Athlete")																																																																																																												
In [34]:	summer.head()																																																																																																												
Out[34]:	<table border="1"> <thead> <tr><th>Athlete</th><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>HAJOS, Alfred</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>HERSCHMANN, Otto</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>DRIVAS, Dimitrios</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>MALOKNIS, Ioannis</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>CHASAPIS, Spiridon</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> </tbody> </table>	Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal	HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze	MALOKNIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold	CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver																																																						
Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																																																																					
HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																																																																					
HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																																																																					
DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																																																																					
MALOKNIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold																																																																																																					
CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver																																																																																																					
In [35]:	summer.index																																																																																																												
Out[35]:	<pre>Index(['HAJOS', 'ALFRED', 'HERSCHMANN', 'OTTO', 'DRIVAS', 'DIMITRIOS', 'MALOKNIS', 'IOANNIS', 'CHASAPIS', 'SPIRIDON', 'CHOROPHAS', 'EFSTATHIOS', 'NEUMANN', 'PAUL', 'REZAEL', 'GHADER', 'TOTROV', 'RUSTAM', 'ALEKSANYAN', 'ARTUR', 'LIDBERG', 'JIMMY'], dtype='object', name='Athlete', length=31165)</pre>																																																																																																												
.reset_index() can reset the index																																																																																																													
This can bring the index back to the default system. That system is numeric.																																																																																																													
If you use .reset_index(drop = False) the previous index (in this case athlete) will remain, just after the conventional numeric index.																																																																																																													
The default is set to false.																																																																																																													
In [37]:	summer.reset_index(drop = False)																																																																																																												
Out[37]:	<table border="1"> <thead> <tr><th>Athlete</th><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>0 HAJOS, Alfred</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>1 HERSCHMANN, Otto</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>2 DRIVAS, Dimitrios</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>3 MALOKNIS, Ioannis</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>4 CHASAPIS, Spiridon</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> <tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr> <tr><td>31160 JANIKOWSKI, Damian</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>POL</td><td>Men</td><td>Wg 84 KG</td><td>Bronze</td></tr> <tr><td>31161 REZAEL, Ghader</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>IRI</td><td>Men</td><td>Wg 96 KG</td><td>Gold</td></tr> <tr><td>31162 TOTROV, Rustam</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>RUS</td><td>Men</td><td>Wg 96 KG</td><td>Silver</td></tr> <tr><td>31163 ALEXSANYAN, Artur</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>ARM</td><td>Men</td><td>Wg 96 KG</td><td>Bronze</td></tr> <tr><td>31164 LIDBERG, Jimmy</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>SWE</td><td>Men</td><td>Wg 96 KG</td><td>Bronze</td></tr> </tbody> </table>	Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal	0 HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	1 HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	2 DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze	3 MALOKNIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold	4 CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver	31160 JANIKOWSKI, Damian	2012	London	Wrestling	Wrestling Freestyle	POL	Men	Wg 84 KG	Bronze	31161 REZAEL, Ghader	2012	London	Wrestling	Wrestling Freestyle	IRI	Men	Wg 96 KG	Gold	31162 TOTROV, Rustam	2012	London	Wrestling	Wrestling Freestyle	RUS	Men	Wg 96 KG	Silver	31163 ALEXSANYAN, Artur	2012	London	Wrestling	Wrestling Freestyle	ARM	Men	Wg 96 KG	Bronze	31164 LIDBERG, Jimmy	2012	London	Wrestling	Wrestling Freestyle	SWE	Men	Wg 96 KG	Bronze
Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																																																																					
0 HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																																																																					
1 HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																																																																					
2 DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																																																																					
3 MALOKNIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold																																																																																																					
4 CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver																																																																																																					
...																																																																																																					
31160 JANIKOWSKI, Damian	2012	London	Wrestling	Wrestling Freestyle	POL	Men	Wg 84 KG	Bronze																																																																																																					
31161 REZAEL, Ghader	2012	London	Wrestling	Wrestling Freestyle	IRI	Men	Wg 96 KG	Gold																																																																																																					
31162 TOTROV, Rustam	2012	London	Wrestling	Wrestling Freestyle	RUS	Men	Wg 96 KG	Silver																																																																																																					
31163 ALEXSANYAN, Artur	2012	London	Wrestling	Wrestling Freestyle	ARM	Men	Wg 96 KG	Bronze																																																																																																					
31164 LIDBERG, Jimmy	2012	London	Wrestling	Wrestling Freestyle	SWE	Men	Wg 96 KG	Bronze																																																																																																					
31165 rows × 9 columns																																																																																																													
If you choose to set .reset_index(drop = True) the previous index will disappear.																																																																																																													
It will not reappear in the original spot before it was set to index. Hence the name "drop".																																																																																																													
In [38]:	summer.reset_index(drop = True)																																																																																																												
Out[38]:	<table border="1"> <thead> <tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>0</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>1</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>2</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>3</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>4</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> <tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr> <tr><td>31165</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>POL</td><td>Men</td><td>Wg 84 KG</td><td>Bronze</td></tr> <tr><td>31166</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>IRI</td><td>Men</td><td>Wg 96 KG</td><td>Gold</td></tr> <tr><td>31167</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>RUS</td><td>Men</td><td>Wg 96 KG</td><td>Silver</td></tr> <tr><td>31168</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>ARM</td><td>Men</td><td>Wg 96 KG</td><td>Bronze</td></tr> <tr><td>31169</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>SWE</td><td>Men</td><td>Wg 96 KG</td><td>Bronze</td></tr> </tbody> </table>	Year	City	Sport	Discipline	Country	Gender	Event	Medal	0	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	1	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	2	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze	3	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold	4	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver	31165	2012	London	Wrestling	Wrestling Freestyle	POL	Men	Wg 84 KG	Bronze	31166	2012	London	Wrestling	Wrestling Freestyle	IRI	Men	Wg 96 KG	Gold	31167	2012	London	Wrestling	Wrestling Freestyle	RUS	Men	Wg 96 KG	Silver	31168	2012	London	Wrestling	Wrestling Freestyle	ARM	Men	Wg 96 KG	Bronze	31169	2012	London	Wrestling	Wrestling Freestyle	SWE	Men	Wg 96 KG	Bronze	
Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																																																																						
0	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																																																																					
1	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																																																																					
2	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																																																																					
3	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold																																																																																																					
4	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver																																																																																																					
...																																																																																																					
31165	2012	London	Wrestling	Wrestling Freestyle	POL	Men	Wg 84 KG	Bronze																																																																																																					
31166	2012	London	Wrestling	Wrestling Freestyle	IRI	Men	Wg 96 KG	Gold																																																																																																					
31167	2012	London	Wrestling	Wrestling Freestyle	RUS	Men	Wg 96 KG	Silver																																																																																																					
31168	2012	London	Wrestling	Wrestling Freestyle	ARM	Men	Wg 96 KG	Bronze																																																																																																					
31169	2012	London	Wrestling	Wrestling Freestyle	SWE	Men	Wg 96 KG	Bronze																																																																																																					
31165 rows × 8 columns																																																																																																													
We set the index back to athlete here.																																																																																																													
In [39]:	summer.reset_index(drop = False)																																																																																																												
Out[39]:	<table border="1"> <thead> <tr><th>Athlete</th><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>0 HAJOS, Alfred</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>1 HERSCHMANN, Otto</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>2 DRIVAS, Dimitrios</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>3 MALOKNIS, Ioannis</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>4 CHASAPIS, Spiridon</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> <tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr> <tr><td>31160 JANIKOWSKI, Damian</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>POL</td><td>Men</td><td>Wg 84 KG</td><td>Bronze</td></tr> <tr><td>31161 REZAEL, Ghader</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>IRI</td><td>Men</td><td>Wg 96 KG</td><td>Gold</td></tr> <tr><td>31162 TOTROV, Rustam</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>RUS</td><td>Men</td><td>Wg 96 KG</td><td>Silver</td></tr> <tr><td>31163 ALEXSANYAN, Artur</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>ARM</td><td>Men</td><td>Wg 96 KG</td><td>Bronze</td></tr> <tr><td>31164 LIDBERG, Jimmy</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>SWE</td><td>Men</td><td>Wg 96 KG</td><td>Bronze</td></tr> </tbody> </table>	Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal	0 HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	1 HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	2 DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze	3 MALOKNIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold	4 CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver	31160 JANIKOWSKI, Damian	2012	London	Wrestling	Wrestling Freestyle	POL	Men	Wg 84 KG	Bronze	31161 REZAEL, Ghader	2012	London	Wrestling	Wrestling Freestyle	IRI	Men	Wg 96 KG	Gold	31162 TOTROV, Rustam	2012	London	Wrestling	Wrestling Freestyle	RUS	Men	Wg 96 KG	Silver	31163 ALEXSANYAN, Artur	2012	London	Wrestling	Wrestling Freestyle	ARM	Men	Wg 96 KG	Bronze	31164 LIDBERG, Jimmy	2012	London	Wrestling	Wrestling Freestyle	SWE	Men	Wg 96 KG	Bronze
Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																																																																					
0 HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																																																																					
1 HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																																																																					
2 DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																																																																					
3 MALOKNIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold																																																																																																					
4 CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver																																																																																																					
...																																																																																																					
31160 JANIKOWSKI, Damian	2012	London	Wrestling	Wrestling Freestyle	POL	Men	Wg 84 KG	Bronze																																																																																																					
31161 REZAEL, Ghader	2012	London	Wrestling	Wrestling Freestyle	IRI	Men	Wg 96 KG	Gold																																																																																																					
31162 TOTROV, Rustam	2012	London	Wrestling	Wrestling Freestyle	RUS	Men	Wg 96 KG	Silver																																																																																																					
31163 ALEXSANYAN, Artur	2012	London	Wrestling	Wrestling Freestyle	ARM	Men	Wg 96 KG	Bronze																																																																																																					
31164 LIDBERG, Jimmy	2012	London	Wrestling	Wrestling Freestyle	SWE	Men	Wg 96 KG	Bronze																																																																																																					
31165 rows × 9 columns																																																																																																													
You can use .set_index to change the index after importing the file.																																																																																																													
Before this the index would be set while the file was being imported.																																																																																																													
In [41]:	#By default the drop value is set to False. summer.set_index("Year", drop = False)																																																																																																												
Out[41]:	<table border="1"> <thead> <tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>0</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>1</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>2</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>3</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>4</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> <tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr> <tr><td>31165</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>POL</td><td>Men</td><td>Wg 84 KG</td><td>Bronze</td></tr> <tr><td>31166</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>IRI</td><td>Men</td><td>Wg 96 KG</td><td>Gold</td></tr> <tr><td>31167</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>RUS</td><td>Men</td><td>Wg 96 KG</td><td>Silver</td></tr> <tr><td>31168</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>ARM</td><td>Men</td><td>Wg 96 KG</td><td>Bronze</td></tr> <tr><td>31169</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>SWE</td><td>Men</td><td>Wg 96 KG</td><td>Bronze</td></tr> </tbody> </table>	Year	City	Sport	Discipline	Country	Gender	Event	Medal	0	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	1	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	2	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze	3	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold	4	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver	31165	2012	London	Wrestling	Wrestling Freestyle	POL	Men	Wg 84 KG	Bronze	31166	2012	London	Wrestling	Wrestling Freestyle	IRI	Men	Wg 96 KG	Gold	31167	2012	London	Wrestling	Wrestling Freestyle	RUS	Men	Wg 96 KG	Silver	31168	2012	London	Wrestling	Wrestling Freestyle	ARM	Men	Wg 96 KG	Bronze	31169	2012	London	Wrestling	Wrestling Freestyle	SWE	Men	Wg 96 KG	Bronze	
Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																																																																						
0	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																																																																					
1	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																																																																					
2	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																																																																					
3	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold																																																																																																					
4	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver																																																																																																					
...																																																																																																					
31165	2012	London	Wrestling	Wrestling Freestyle	POL	Men	Wg 84 KG	Bronze																																																																																																					
31166	2012	London	Wrestling	Wrestling Freestyle	IRI	Men	Wg 96 KG	Gold																																																																																																					
31167	2012	London	Wrestling	Wrestling Freestyle	RUS	Men	Wg 96 KG	Silver																																																																																																					
31168	2012	London	Wrestling	Wrestling Freestyle	ARM	Men	Wg 96 KG	Bronze																																																																																																					
31169	2012	London	Wrestling	Wrestling Freestyle	SWE	Men	Wg 96 KG	Bronze																																																																																																					
31165 rows × 8 columns																																																																																																													
.reset_index() can reset the index																																																																																																													
This can bring the index back to the default system. That system is numeric.																																																																																																													
If you use .reset_index(drop = False) the previous index (in this case athlete) will remain, just after the conventional numeric index.																																																																																																													
The default is set to false.																																																																																																													
In [37]:	summer.reset_index(drop = False)																																																																																																												
Out[37]:	<table border="1"> <thead> <tr><th>Athlete</th><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>0 HAJOS, Alfred</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>1 HERSCHMANN, Otto</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>2 DRIVAS, Dimitrios</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>3 MALOKNIS, Ioannis</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For</td></tr></tbody></table>	Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal	0 HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	1 HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	2 DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze	3 MALOKNIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For																																																																
Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																																																																					
0 HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																																																																					
1 HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																																																																					
2 DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																																																																					
3 MALOKNIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For																																																																																																						

Medal_No	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal
0	Medal_No1	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold
1	Medal_No2	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver
2	Medal_No3	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze
3	Medal_No4	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold
4	Medal_No5	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver
...									
31160	Medal_No31161	2012	London	Wrestling	Wrestling Freestyle	POL	Men	Wg 84 KG	Bronze
31161	Medal_No31162	2012	London	Wrestling	Wrestling Freestyle	IRI	Men	Wg 96 KG	Gold
31162	Medal_No31163	2012	London	Wrestling	Wrestling Freestyle	RUS	Men	Wg 96 KG	Silver
31163	Medal_No31164	2012	London	Wrestling	Wrestling Freestyle	ARM	Men	Wg 96 KG	Bronze
31164	Medal_No31165	2012	London	Wrestling	Wrestling Freestyle	SWE	Men	Wg 96 KG	Bronze
31165	rows	9	columns						

In [1]:	import pandas as pd																																																											
In [2]:	summer = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\summer.csv")																																																											
In [3]:	summer.head()																																																											
Out[4]:	<table border="1"> <thead> <tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Athlete</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>0</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HAJOS, Alfred</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>1</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HERSCHMANN, Otto</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>2</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>DRIVAS, Dimitrios</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>3</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>MALOKNIS, Ioannis</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>4</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>CHASAPIS, Spiridon</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> </tbody> </table>	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal	0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold	1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver	2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze	3	1896	Athens	Aquatics	Swimming	MALOKNIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold	4	1896	Athens	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver
Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal																																																				
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold																																																			
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver																																																			
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze																																																			
3	1896	Athens	Aquatics	Swimming	MALOKNIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold																																																			
4	1896	Athens	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver																																																			
We can change the index to the name of the athlete instead of the default numerical value.																																																												
In [5]:	summer = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\summer.csv", index_col = "Athlete")																																																											
In [6]:	summer.tail(3)																																																											
Out[7]:	<table border="1"> <thead> <tr><th>Athlete</th><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>JANIKOWSKI, Damian</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>POL</td><td>Men</td><td>Wg 84 KG</td><td>Bronze</td></tr> <tr><td>REZAEI, Ghased Gholamreza</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>IRI</td><td>Men</td><td>Wg 96 KG</td><td>Gold</td></tr> <tr><td>TOTROV, Rustam</td><td>2012</td><td>London</td><td>Wrestling</td><td>Wrestling Freestyle</td><td>RUS</td><td>Men</td><td>Wg 96 KG</td><td>Silver</td></tr> </tbody> </table>	Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal	JANIKOWSKI, Damian	2012	London	Wrestling	Wrestling Freestyle	POL	Men	Wg 84 KG	Bronze	REZAEI, Ghased Gholamreza	2012	London	Wrestling	Wrestling Freestyle	IRI	Men	Wg 96 KG	Gold	TOTROV, Rustam	2012	London	Wrestling	Wrestling Freestyle	RUS	Men	Wg 96 KG	Silver																							
Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																				
JANIKOWSKI, Damian	2012	London	Wrestling	Wrestling Freestyle	POL	Men	Wg 84 KG	Bronze																																																				
REZAEI, Ghased Gholamreza	2012	London	Wrestling	Wrestling Freestyle	IRI	Men	Wg 96 KG	Gold																																																				
TOTROV, Rustam	2012	London	Wrestling	Wrestling Freestyle	RUS	Men	Wg 96 KG	Silver																																																				
In [8]:	summer.info()																																																											
Out[9]:	<class 'pandas.core.frame.DataFrame'> Index: 31165 entries, HAJOS, Alfred to LIDBERG, Jimmy Data columns (total 9 columns): # Column Non-Null Count Dtype --- 0 Year 31165 non-null int64 1 City 31165 non-null object 2 Sport 31165 non-null object 3 Discipline 31165 non-null object 4 Country 31161 non-null object 5 Gender 31165 non-null object 6 Event 31165 non-null object 7 Medal 31165 non-null object 8 Athlete 31165 non-null object dtypes: int64(1), object(7) memory usage: 2.1+ MB																																																											
In [10]:	summer.index																																																											
Out[11]:	Index(['HAJOS, Alfred', 'HERSCHMANN, Otto', 'DRIVAS, Dimitrios', 'MALOKNIS, Ioannis', 'CHASAPIS, Spiridon', 'CHOROPHAS, Efstrathios', 'REZAEI, Ghased', 'TOTROV, Rustam', 'LIDBERG, Jimmy'], dtype='object', name='Athlete', length=31165)																																																											
summer.columns																																																												
In [12]:	type(summer.columns)																																																											
Out[12]:	pandas.core.indexes.base.Index																																																											
Note that "axes" is used to refer to the index instead of axis.																																																												
In [13]:	summer.axes																																																											
Out[14]:	[Index(['Athlete', 'Year', 'City', 'Sport'], dtype='object')]																																																											
Remember you can slice characteristics or columns using slices.																																																												
In [15]:	summer[['Athlete', 'Year', 'City', 'Sport']]																																																											
Out[16]:	Index(['HAJOS, Alfred', 'HERSCHMANN, Otto', 'DRIVAS, Dimitrios', 'MALOKNIS, Ioannis', 'CHASAPIS, Spiridon', 'CHOROPHAS, Efstrathios', 'REZAEI, Ghased', 'TOTROV, Rustam', 'LIDBERG, Jimmy'], dtype='object', name='Athlete', length=31165)																																																											
summer.columns																																																												
In [17]:	type(summer.columns)																																																											
Out[18]:	pandas.core.indexes.base.Index																																																											
You can slice indexes as well using slices.																																																												
In [19]:	# You can select indexes using their number.																																																											
Out[20]:	summer[0]																																																											
Out[21]:	'HAJOS, Alfred'																																																											
In [22]:	summer[0:1]																																																											
Out[22]:	Index(['Year', 'City', 'Sport'], dtype='object')																																																											
In [23]:	You can slice indexes as well using slices.																																																											
In [24]:	# You can pick the last item using -1.																																																											
Out[25]:	summer[-1]																																																											
Out[26]:	'LIDBERG, Jimmy'																																																											
In [27]:	# Is unique can be used to show if the data or index contains values that have only been used once.																																																											
Out[28]:	The is unique can be used to find out if their are repeated values. If the answer is true then the entry is only found once inside of the data.																																																											
In [29]:	summer.index.is_unique																																																											
Out[29]:	False																																																											
.get_loc can find the index number of a string that you input.																																																												
In [30]:	# This is one of the last names in the data.																																																											
Out[31]:	summer.index.get_loc("ALEKSANYAN, Artur")																																																											
Out[32]:	31163																																																											
In [33]:	# This is one of the first names in the data.																																																											
Out[34]:	summer.index.get_loc("DRIVAS, Dimitrios")																																																											
Out[35]:	2																																																											
In [36]:	# You can pick the last item using -1.																																																											
Out[37]:	summer.index[-1]																																																											
Out[38]:	'LIDBERG, Jimmy'																																																											
In [39]:	.is_unique can be used to prevent duplicates from happening in this case.																																																											
Out[40]:	summer.set_index('Year', drop = True)																																																											
In [41]:	# By default the drop value is set to False.																																																											
Out[42]:	summer.set_index('Year', drop = False)																																																											
In [43]:	summer.set_index('Year', drop = True)																																																											
Out[44]:	<table border="1"> <thead> <tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>0</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>1</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>2</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>3</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>4</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> </tbody> </table>	Year	City	Sport	Discipline	Country	Gender	Event	Medal	0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze	3	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold	4	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver											
Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																					
0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																					
1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																					
2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																					
3	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold																																																					
4	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver																																																					
In [45]:	summer.set_index('Year', drop = True)																																																											
Out[46]:	<table border="1"> <thead> <tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>0</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>1</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>2</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>3</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>4</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> </tbody> </table>	Year	City	Sport	Discipline	Country	Gender	Event	Medal	0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze	3	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold	4	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver											
Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																					
0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																					
1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																					
2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																					
3	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold																																																					
4	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver																																																					
In [47]:	summer.index.is_unique																																																											
Out[48]:	False																																																											
In [49]:	# No index change is unique because they are immutable.																																																											
Out[50]:	summer.index[0] == 1894																																																											
In [51]:	summer.set_index('Year', drop = True)																																																											
Out[52]:	<table border="1"> <thead> <tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>0</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>1</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>2</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>3</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>4</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> </tbody> </table>	Year	City	Sport	Discipline	Country	Gender	Event	Medal	0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze	3	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold	4	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver											
Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																					
0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																					
1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																					
2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																					
3	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold																																																					
4	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver																																																					
In [53]:	summer.set_index('Year', drop = True)																																																											
Out[54]:	summer.set_index('Year', drop = True)																																																											
In [55]:	summer.set_index('Year', drop = True)																																																											
Out[56]:	summer.set_index('Year', drop = True)																																																											
In [57]:	.reset_index() can reset the index																																																											
Out[58]:	This can bring the index back to the default system. That system is numeric.																																																											
In [59]:	If you use .reset_index(drop = False) the previous index (in this case athlete) will remain, just after the conventional numeric index.																																																											
Out[60]:	The default is set to false.																																																											
In [61]:	summer.reset_index(drop = False)																																																											
Out[62]:	<table border="1"> <thead> <tr><th>Athlete</th><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>HAJOS, Alfred</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>HERSCHMANN, Otto</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>DRIVAS, Dimitrios</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>MALOKNIS, Ioannis</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>CHASAPIS, Spiridon</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> </tbody> </table>	Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal	HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze	MALOKNIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold	CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver					
Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																				
HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																				
HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																				
DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																				
MALOKNIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold																																																				
CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver																																																				
In [63]:	summer.set_index('Year', drop = True)																																																											
Out[64]:	<table border="1"> <thead> <tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>0</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>1</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>2</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>3</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>4</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> </tbody> </table>	Year	City	Sport	Discipline	Country	Gender	Event	Medal	0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze	3	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold	4	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver											
Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																					
0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																					
1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																					
2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																					
3	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold																																																					
4	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver																																																					
In [65]:	summer.set_index('Year', drop = True)																																																											
Out[66]:	<table border="1"> <thead> <tr><th>Athlete</th><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>HAJOS, Alfred</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>HERSCHMANN, Otto</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>DRIVAS, Dimitrios</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>MALOKNIS, Ioannis</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>CHASAPIS, Spiridon</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> </tbody> </table>	Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal	HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze	MALOKNIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold	CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver					
Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																				
HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																				
HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																				
DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																				
MALOKNIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold																																																				
CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver																																																				
In [67]:	summer.set_index('Year', drop = True)																																																											
Out[68]:	<table border="1"> <thead> <tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>0</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>1</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>2</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>3</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>4</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> </tbody> </table>	Year	City	Sport	Discipline	Country	Gender	Event	Medal	0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze	3	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold	4	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver											
Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																					
0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																					
1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																					
2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																					
3	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold																																																					
4	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver																																																					
In [69]:	summer.set_index('Year', drop = True)																																																											
Out[70]:	<table border="1"> <thead> <tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>0</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>1</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>2</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>3</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>4</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> </tbody> </table>	Year	City	Sport	Discipline	Country	Gender	Event	Medal	0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze	3	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold	4	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver											
Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																					
0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																					
1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																					
2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																					
3	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold																																																					
4	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver																																																					
In [71]:	summer.set_index('Year', drop = True)																																																											
Out[72]:	<table border="1"> <thead> <tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>0</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>1</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>2</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>3</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>4</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> </tbody> </table>	Year	City	Sport	Discipline	Country	Gender	Event	Medal	0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze	3	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold	4	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver											
Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																					
0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																					
1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																					
2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																					
3	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold																																																					
4	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver																																																					
In [73]:	summer.set_index('Year', drop = True)																																																											
Out[74]:	<table border="1"> <thead> <tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>0</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>1</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>2</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr> <tr><td>3</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr> <tr><td>4</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr> </tbody> </table>	Year	City	Sport	Discipline	Country	Gender	Event	Medal	0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze	3	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold	4	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver											
Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																					
0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																					
1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																					
2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																					
3	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold																																																					
4	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver																																																					
In [75]:	summer.set_index('Year', drop = True)																																																											
Out[76]:	<table border="1"> <thead> <tr><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr> </thead> <tbody> <tr><td>0</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr> <tr><td>1</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr> <tr><td>2</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr></tbody></table>	Year	City	Sport	Discipline	Country	Gender	Event	Medal	0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold	1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver	2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																											
Year	City	Sport	Discipline	Country	Gender	Event	Medal																																																					
0	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold																																																					
1	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver																																																					
2	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze																																																					

Out[187]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck	
0	1	1	male	80.0	0	0	30.0000	S	A	
1	0	3	male	74.0	0	0	7.7500	S	NaN	
2	0	1	male	71.0	0	0	49.5042	C	NaN	
3	0	1	male	71.0	0	0	34.6542	C	A	
4	1	0	3	male	70.5	0	0	7.7500	Q	NaN
...	
851	0	3	male	NaN	0	0	7.2292	C	NaN	
852	1	0	3	female	NaN	8	2	69.5500	S	NaN
853	0	3	male	NaN	0	0	9.5000	S	NaN	
854	0	3	male	NaN	0	0	7.8958	S	NaN	
855	0	3	female	NaN	1	2	23.4500	S	NaN	

891 rows x 9 columns

You can pass on multiple values by using a list.

Use `sort_values([list of columns])`

This will prioritize the data based on the columns inputed. The subsequent columns after the first can be thought of as tie breakers.

Thus the data goes from highest pclass to lowest class, oldest to youngest, male to female. This is due to the descending order highest to lowest. For the alphabet it will go from the latest letter to the earlier letters in reverse alphabetical order.

In [189]:

```
titanic.sort_values(["pclass", "sex", "age"], ascending = False)
```

Out[189]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck	
0	1	1	male	80.0	0	0	30.0000	S	A	
1	0	3	male	74.0	0	0	7.7500	S	NaN	
2	0	1	male	71.0	0	0	49.5042	C	NaN	
3	0	1	male	71.0	0	0	34.6542	C	A	
4	1	0	3	male	70.5	0	0	7.7500	Q	NaN
...	
851	0	3	male	NaN	0	0	7.2292	C	NaN	
852	1	0	3	female	NaN	8	2	69.5500	S	NaN
853	0	3	male	NaN	0	0	9.5000	S	NaN	
854	0	3	male	NaN	0	0	7.8958	S	NaN	
855	0	3	female	NaN	1	2	23.4500	S	NaN	

891 rows x 9 columns

You can make a list to determine if the order is ascending or descending for each individual column using a list.

In [191]:

```
titanic.sort_values(["pclass", "sex", "age"], ascending = [True, False, True], inplace = True)
```

In [192]:

```
titanic.head(1)
```

#This chart goes from pclass smallest to largest.
#Sex male to female.
#Youngest to oldest.

Out[192]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	1	3	male	74.0	0	0	7.7500	S	NaN
1	0	3	male	70.5	0	0	7.7500	Q	NaN
2	0	3	male	65.0	0	0	7.7500	Q	NaN
3	0	3	male	61.0	0	0	6.2375	S	NaN
4	0	3	male	59.0	0	0	7.2500	S	NaN
...
334	1	1	female	NaN	1	0	133.6500	S	NaN
375	1	1	female	NaN	1	0	82.1708	C	NaN
457	1	1	female	NaN	1	0	51.8625	S	D
849	1	1	female	NaN	1	0	89.1042	C	C

891 rows x 9 columns

You can make a list to determine if the order is ascending or descending for each individual column using a list.

In [193]:

```
titanic.sort_values(["pclass", "sex", "age"], ascending = [True, False, True], inplace = True)
```

In [194]:

```
titanic.head(1)
```

#This chart goes from pclass smallest to largest.
#Sex male to female.
#Youngest to oldest.

Out[194]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	1	3	male	0.92	1	2	151.5500	S	C
1	1	1	male	4.00	0	2	81.8583	S	A
2	0	1	male	11.00	1	2	120.0000	S	B
3	0	1	male	17.00	0	2	110.8833	C	C
4	0	1	male	18.00	1	0	108.9000	C	C

891 rows x 9 columns

Revert back to the original dataframe.

In [195]:

```
titanic.sort_index(ascending = True, inplace = True)
```

In [196]:

```
titanic
```

Out[196]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	1	1	male	0.92	1	2	151.5500	S	C
1	1	1	male	4.00	0	2	81.8583	S	A
2	0	1	male	11.00	1	2	120.0000	S	B
3	0	1	male	17.00	0	2	110.8833	C	C
4	0	1	male	18.00	1	0	108.9000	C	C

891 rows x 9 columns

nunique(), nlargest(), and nsmallest() with DataFrames

In [197]:

```
import pandas as pd
```

In [198]:

```
titanic = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

In [199]:

```
titanic.head()
```

Out[199]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	1	female	26.0	0	0	7.9250	S	NaN
3	0	3	male	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN

In [200]:

```
titanic.tail(1)
```

Out[200]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	2	male	27.0	0	0	13.00	S	NaN
1	0	1	female	19.0	0	0	30.00	S	B
2	0	3	female	NaN	1	2	23.4500	S	NaN
3	0	1	male	26.0	0	0	30.0000	C	C
4	0	3	male	32.0	0	0	7.7500	Q	NaN

891 rows x 9 columns

nunique(), nlargest(), and nsmallest() with DataFrames

In [201]:

```
import pandas as pd
```

In [202]:

```
titanic = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

In [203]:

```
titanic.head()
```

Out[203]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	1	female	26.0	0	0	7.9250	S	NaN
3	0	3	male	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN

In [204]:

```
titanic.tail(1)
```

Out[204]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	2	male	27.0	0	0	13.00	S	NaN
1	0	1	female	19.0	0	0	30.00	S	B
2	0	3	female	NaN	1	2	23.4500	S	NaN
3	0	1	male	26.0	0	0	30.00	C	C
4	0	3	male	32.0	0	0	7.7500	Q	NaN

891 rows x 9 columns

#You cannot use unique for a whole dataframe. You have to select a characteristic.

In [205]:

```
titanic.unique()
```

Out[205]:

```
AttributeError: 'DataFrame' object has no attribute 'unique'
```

In [206]:

```
titanic.head(1)
```

Out[206]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	1	3	male	0.92	1	2	151.5500	S	C

In [207]:

```
#You cannot use unique for a whole dataframe. You have to select a characteristic.
```

In [208]:

```
titanic.unique()
```

Out[208]:

```
AttributeError: 'DataFrame' object has no attribute 'unique'
```

In [209]:

```
titanic.unique(1)
```

Out[209]:

```
AttributeError: 'DataFrame' object has no attribute 'unique'
```

In [210]:

```
titanic.unique(1, axis = 0)
```

Out[210]:

```
array([22.0, 38.0, 26.0, 35.0, 30.0, 19.0, 28.0, 26.0, 24.0, 23.0, 20.0, 18.0, 16.0, 14.0, 12.0, 10.0, 8.0, 6.0, 4.0, 2.0, 0.0], dtype = float64)
```

n.unique values can be used for a DataFrame.

In [211]:

```
titanic.unique(axis = 0, dropna = True)
```

Out[211]:

```
array([22.0, 38.0, 26.0, 35.0, 30.0, 19.0, 28.0, 26.0, 24.0, 23.0, 20.0, 18.0, 16.0, 14.0, 12.0, 10.0, 8.0, 6.0, 4.0, 2.0, 0.0], dtype = float64)
```

In [212]:

```
titanic.unique(1)
```

Out[212]:

```
array([22.0, 38.0, 26.0, 35.0, 30.0, 19.0, 28.0, 26.0, 24.0, 23.0, 20.0, 18.0, 16.0, 14.0, 12.0, 10.0, 8.0, 6.0, 4.0, 2.0, 0.0], dtype = float64)
```

In [213]:

```
#You cannot use unique for a whole dataframe. You have to select a characteristic.
```

In [214]:

```
titanic.unique(1, axis = 0)
```

Out[214]:

```
array([22.0, 38.0, 26.0, 35.0, 30.0, 19.0, 28.0, 26.0, 24.0, 23.0, 20.0, 18.0, 16.0, 14.0, 12.0, 10.0, 8.0, 6.0, 4.0, 2.0, 0.0], dtype = float64)
```

In [215]:

```
#You cannot use unique for a whole dataframe. You have to select a characteristic.
```

In [216]:

```
titanic.unique(1, axis = 0, dropna = True)
```

Out[216]:

```
array([22.0, 38.0, 26.0, 35.0, 30.0, 19.0, 28.0, 26.0, 24.0, 23.0, 20.0, 18.0, 16.0, 14.0, 12.0, 10.0, 8.0, 6.0, 4.0, 2.0, 0.0], dtype = float64)
```

In [217]:

```
#You cannot use unique for a whole dataframe. You have to select a characteristic.
```

In [218]:

```
titanic.unique(1, axis = 0, dropna = True)
```

Out[218]:

```
array([22.0, 38.0, 26.0, 35.0, 30.0, 19.0, 28.0, 26.0, 24.0, 23.0, 20.0, 18.0, 16.0, 14.0, 12.0, 10.0, 8.0, 6.0, 4.0, 2.0, 0.0], dtype = float64)
```

In [219]:

```
#You cannot use unique for a whole dataframe. You have to select a characteristic.
```

In [220]:

```
titanic.unique(1, axis = 0, dropna = True)
```

Out[220]:

```
array([22.0, 38.0, 26.0, 35.0, 30.0, 19.0, 28.0, 26.0, 24.0, 23.0, 20.0, 18.0, 16.0, 14.0, 12.0, 10.0, 8.0, 6.0, 4.0, 2.0, 0.0], dtype = float64)
```

In [221]:

```
#You cannot use unique for a whole dataframe. You have to select a characteristic.
```

In [222]:

```
titanic.unique(1, axis = 0, dropna = True)
```

Out[222]:

```
array([22.0, 38.0, 26.0, 35.0, 30.0, 19.0, 28.0, 26.0, 24.0, 23.0, 20.0, 18.0, 16.0, 14.
```


Intro to NA Values

```
In [16]: import pandas as pd
import numpy as np
sales = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\sales.csv", index_col = 0)

NaN stands for not a number.
```

```
In [9]: sales
```

```
Out[9]:
```

	Mon	Tue	Wed	Thu	Fri
Steven	34	27	15	NaN	33
Mike	45	9	74	87.0	12
Andi	17	33	54	8.0	29
Paul	87	67	27	45.0	7

```
In [10]: sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, Steven to Paul
Data columns (total 5 columns):
 # Column  Non-Null Count  Dtype  
---  --- 
 0   Mon      4 non-null    int64  
 1   Tue      4 non-null    int64  
 2   Wed      4 non-null    int64  
 3   Thu      3 non-null    float64 
 4   Fri      4 non-null    int64  
dtypes: float64(1), int64(4)
memory usage: 192.0+ bytes
```

```
In [13]: sales.loc["Steven", "Thu"]
```

```
Out[13]: nan
```

```
In [14]: sales
```

```
Out[14]:
```

	Mon	Tue	Wed	Thu	Fri
Steven	34	27	15.0	NaN	33
Mike	45	9	74	87.0	12
Andi	17	33	54	8.0	29
Paul	87	67	27.0	45.0	7

Add missing values or NaN to data.

We can use numpy to add a NaN by using .iloc and typing in the coordinates of the missing value.

```
In [17]: sales.iloc[2,2] = np.nan
```

```
In [18]: sales
```

```
Out[18]:
```

	Mon	Tue	Wed	Thu	Fri
Steven	34	27	15.0	NaN	33
Mike	45	9	74	87.0	12
Andi	17	33	54	8.0	29
Paul	87	67	27.0	45.0	7

```
In [19]: sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, Steven to Paul
Data columns (total 5 columns):
 # Column  Non-Null Count  Dtype  
---  --- 
 0   Mon      4 non-null    int64  
 1   Tue      4 non-null    int64  
 2   Wed      4 non-null    int64  
 3   Thu      3 non-null    float64 
 4   Fri      4 non-null    int64  
dtypes: float64(2), int64(3)
memory usage: 192.0+ bytes
```

Empty string values "" do not show up as NaN. That is something that has to be kept in mind.

This means that you cannot rely on the .info() to show that data is missing.

Handling missing values.

Option 1: Ignore missing values.

Pandas is equipped to handle missing values by ignoring them in statistical processes.

Option 2: Delete missing values.

The observation (cell) and/or the column can be deleted.

Option 3: Replace missing values.

This can be done with the actual value or the approximate value.

```
In [24]: titanic = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

```
In [26]: titanic.head()
```

```
Out[26]:
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	0	1	female	35.0	0	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN

```
In [28]: # You can tell the missing values based off of the RangeIndex that will show the total amount of cells.
# Any element that has fewer than the range index is a missing cell. There could be more due to formating errors.
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 # Column  Non-Null Count  Dtype  
---  --- 
 0   survived   891 non-null   int64  
 1   pclass     891 non-null   int64  
 2   sex        891 non-null   object 
 3   age        714 non-null   float64 
 4   sibsp      891 non-null   int64  
 5   parch      891 non-null   int64  
 6   fare       891 non-null   float64 
 7   embarked   891 non-null   object 
 8   deck       203 non-null   object 
dtypes: float64(2), int64(4), object(3)
memory usage: 62.8+ KB
```

The .isna() shows us whether or not the value is missing. If the value is true it is missing.

```
In [39]: titanic.isna()
```

```
Out[29]:
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	True
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	True
...
886	False	False	False	False	False	False	False	False	True
887	False	False	False	False	False	False	False	False	False
888	False	False	False	False	True	False	False	False	True
889	False	False	False	False	False	False	False	False	False
890	False	False	False	False	False	False	False	False	True

891 rows x 9 columns

.isna().sum() sums up the missing value for each characteristic.

```
In [31]: titanic.isna().sum()
```

```
Out[31]:
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
survived	0	0	0	177	0	0	0	0	0
pclass	0	0	0	0	0	0	0	0	0
sex	0	0	0	0	0	0	0	0	0
age	177	0	0	0	0	0	0	0	0
sibsp	0	0	0	0	0	0	0	0	0
parch	0	0	0	0	0	0	0	0	0
fare	0	0	0	0	0	0	0	0	0
embarked	0	0	0	0	0	0	0	0	0
deck	0	0	0	0	0	0	0	0	0

For the embarked column it is best to do nothing because there are only two missing values and they are not that important to the analysis.

```
In [33]: titanic.loc[titanic.embarked.isna()]
```

```
Out[33]:
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
61	1	1	female	38.0	0	0	80.00	NaN	B
829	1	1	female	62.0	0	0	80.00	NaN	B

```
In [36]: ## We can drop data that has an NaN.
```

```
In [37]: # Before value.
titanic.shape
```

```
Out[37]: (891, 9)
```

```
In [38]: titanic.dropna()
```

```
Out[38]:
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
1	1	1	female	38.0	1	0	71.2833	C	C
3	1	1	female	35.0	1	0	53.1000	S	C
6	0	1	male	54.0	0	0	51.8625	S	E
10	1	3	female	47.0	1	1	16.7000	S	G
11	1	1	female	58.0	0	0	26.5500	S	C
...
871	1	1	female	47.0	1	1	52.5542	S	D
872	1	1	male	33.0	0	0	5.0000	S	B
879	1	1	female	56.0	0	1	83.1583	C	C
887	1	1	female	19.0	0	0	30.0000	S	B
889	1	1	male	26.0	0	0	30.0000	C	C

182 rows x 9 columns

```
In [43]: ## Dropping any index with an empty cell made 600 pieces of data become eliminated.
```

```
In [42]: titanic.dropna().shape
```

```
Out[42]: (182, 9)
```

```
In [47]: ## The default value is any for the .dropna() is how = "any".
# That means that the index is dropped if there is any of the columns missing in and index.
titanic.dropna(how = "all").shape
```

```
Out[47]: (891, 9)
```

```
In [48]: # To drop columns the axis needs to equal 1. axis = 1
```

Three columns were dropped due to the missing values.

See below.

```
In [51]: titanic.dropna(axis = 1, how = "any").shape
```

```
Out[51]: (891, 6)
```

	survived	pclass	sex	age	sibsp	parch
survived	0	0	0	177	0	0
pclass	0	0	0	0	0	0
sex	0	0	0	0	0	0
age	177	0	0	0	0	0
sibsp	0	0	0	0	0	0
parch	0	0	0	0	0	0

Thresh allows you to only drop a column or a row if there is a certain number of missing values.

```
In [59]: titanic.dropna(axis = 1, thresh = 500, inplace = True)
```

```
In [61]: # Notice that one column was deleted.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 # Column  Non-Null Count  Dtype  
---  --- 
 0   survived   891 non-null   int64  
 1   pclass     891 non-null   int64  
 2   sex        891 non-null   object 
 3   age        714 non-null   float64 
 4   sibsp      891 non-null   int64  
 5   parch      891 non-null   int64  
 6   fare       891 non-null   float64 
 7   embarked   891 non-null   object 
 8   deck       203 non-null   object 
dtypes: float64(2), int64(4), object(3)
memory usage: 62.8+ KB
```

```
In [63]: # This usage shows every instance of a NaN for age.
```

```
titanic.loc[titanic.age.isna()]
```

```
Out[63]:
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
5	0	3	male	Nan	0	0	8.483	Q	
17	1	2	male	Nan	0	0	72.250	C	
19	1	3	female	Nan	0	0	72.250	C	
26	0	3	female	Nan	0	0	7.250	Q	
...
859	0	3	female	Nan	0	0	7.2292	C	
863	0	3	female	Nan	0	0	69.5500	S	
878	0	3	male	Nan	0	0	7.9500	S	
879	0	3	male	Nan	0	0	7.9500	S	
888	0	3	male	Nan	0	0	7.9500	S	
889	0	3	female	Nan	1	2	34.5000	S	

177 rows x 8 columns

```
In [66]: mean_age = titanic.age.mean()
mean_age
```

```
Out[66]: 29.69176470582
```

```
In [68]: titanic.age.fillna(value = mean_age, inplace = True)
```

```
Out[68]:
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
5	0	3							

```
In [2]:  
import pandas as pd  
titanic= pd.read_csv(r"C:\Users\alonz\Downloads\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

```
In [3]:  
titanic.head()
```

```
Out[3]:
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN

```
In [4]:  
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 891 entries, 0 to 890  
Data columns (total 9 columns):  
 #   Column      Non-Null Count  Dtype     
---  --    
 0   survived    891 non-null    int64    
 1   pclass      891 non-null    int64    
 2   sex         891 non-null    object    
 3   age         714 non-null    float64   
 4   sibsp       891 non-null    int64    
 5   parch       891 non-null    int64    
 6   fare         891 non-null    float64   
 7   embarked    889 non-null    object    
 8   deck         203 non-null    object    
dtypes: float64(2), int64(4), object(3)  
memory usage: 62.8+ KB
```

Import matplotlib into python.

matplotlib is abbreviated as plt.

```
In [5]:  
import matplotlib.pyplot as plt  
# Note that you have to use .plot() as a method and plt. as root for other things.
```

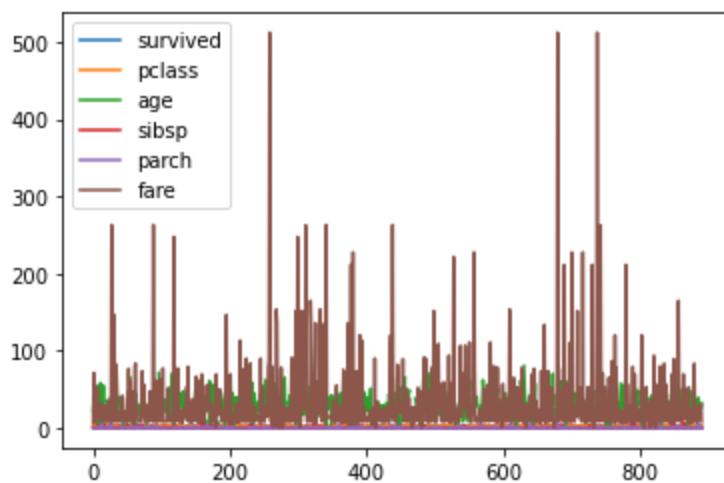
To show a plot you must use a .plot() method. If you use that you can get a chart.

You can use plt.show() afterwards to show the graph. This seems unnecessary at this point.

```
In [8]:  
titanic.plot()  
plt.show()
```

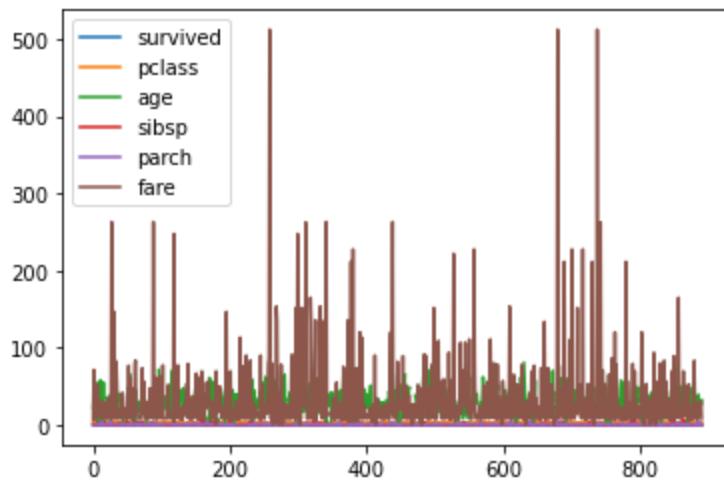
```
Out[8]:
```





```
In [15]: # Only need .plot() to show the chart.  
titanic.plot()
```

```
Out[15]: <AxesSubplot:>
```

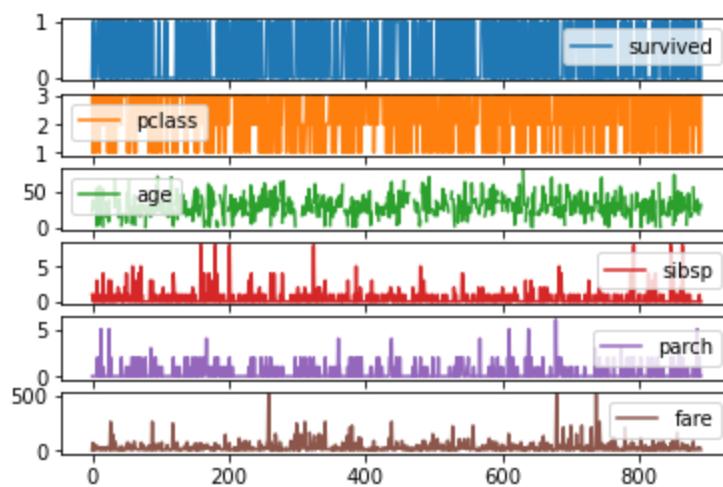


Remember to use shift + tab to see the other options that you have within the method .plot() to customize the plotting.

```
In [17]: ## Subplots enable us to see the plots seperated on different graphs.
```

```
In [18]: titanic.plot(subplots = True)
```

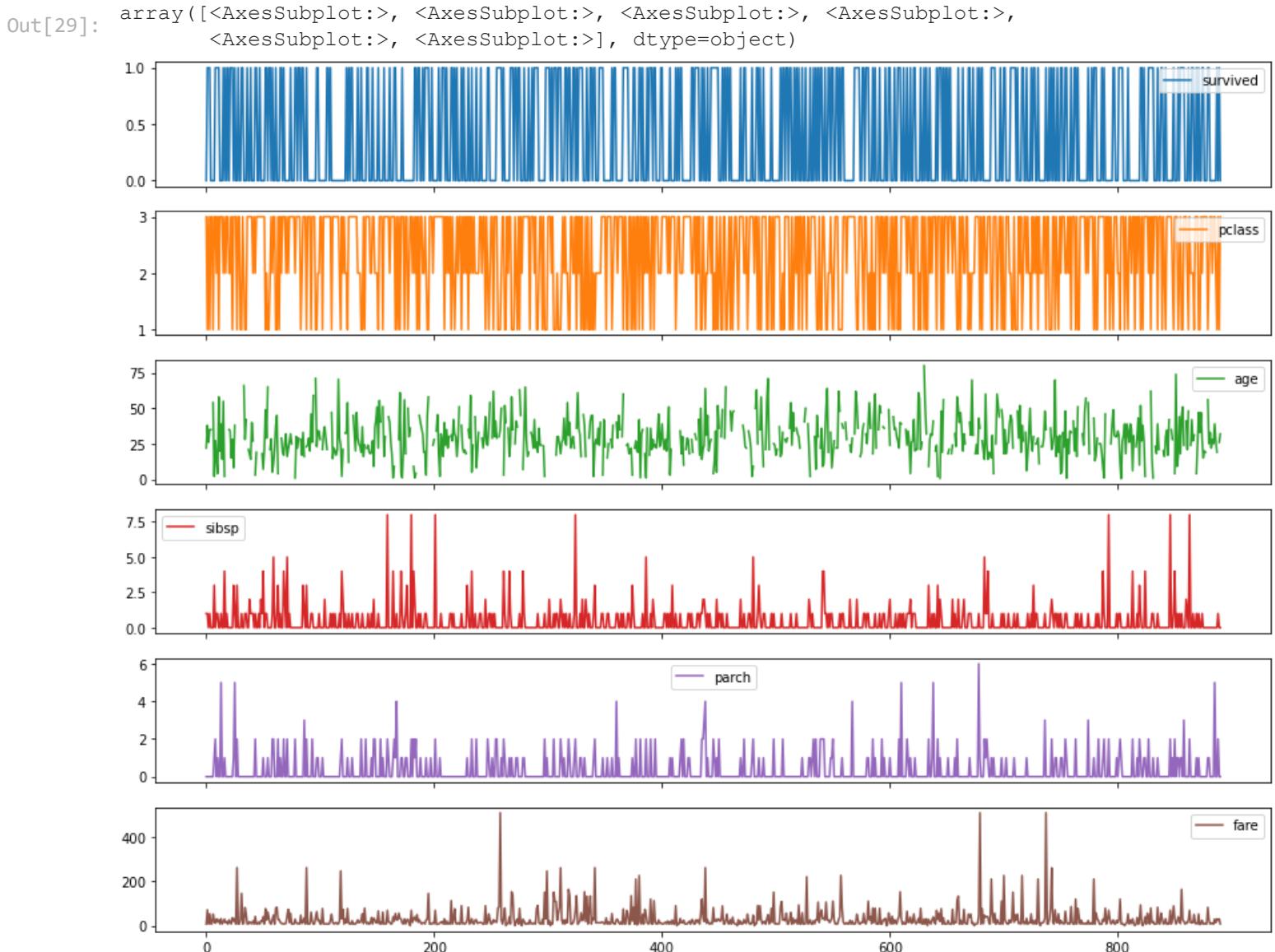
```
Out[18]: array([<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,  
   <AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



The size of the plot can be changed by using figsize(width, height)

The more you scale you height the more spread apart the plotting will be. The higher your height is the easier it is to differentiate different measurements. The unit is in inches.

```
In [29]: #By default the sharex is set to True. The value is not even displayed.
titanic.plot(subplots = True, figsize = (15, 12), sharex = True)
```



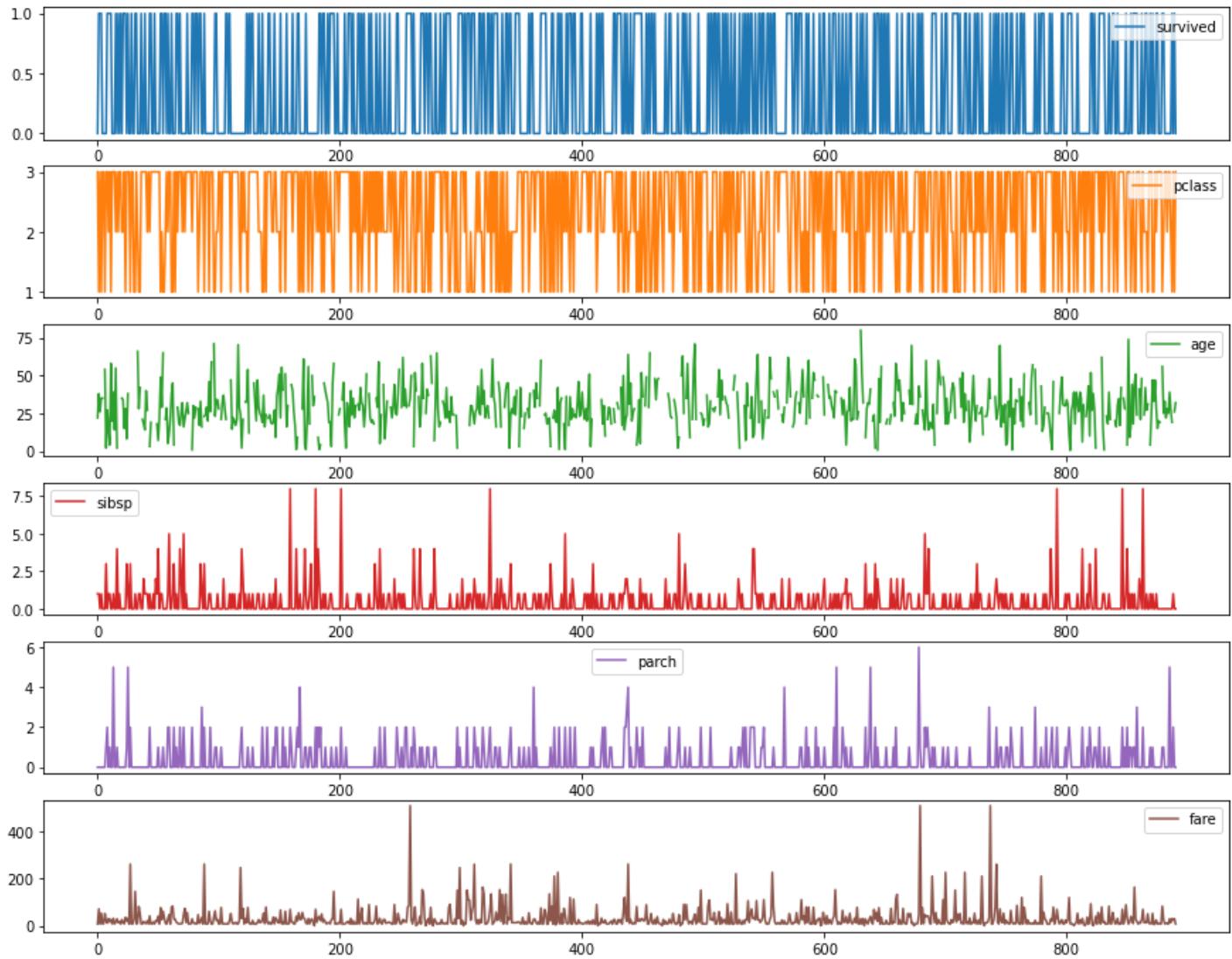
Set the `sharex` to `False` in order to have each subplot have their own x-axis instead of sharing one.

In [32]:

```
titanic.plot(subplots = True, figsize = (15, 12), sharex = False)  
# sibsp measured the number of sibblings and spouses onboard.
```

Out[32]:

```
array([<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,  
<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



Dataplots can share y values. This is turned off by default.

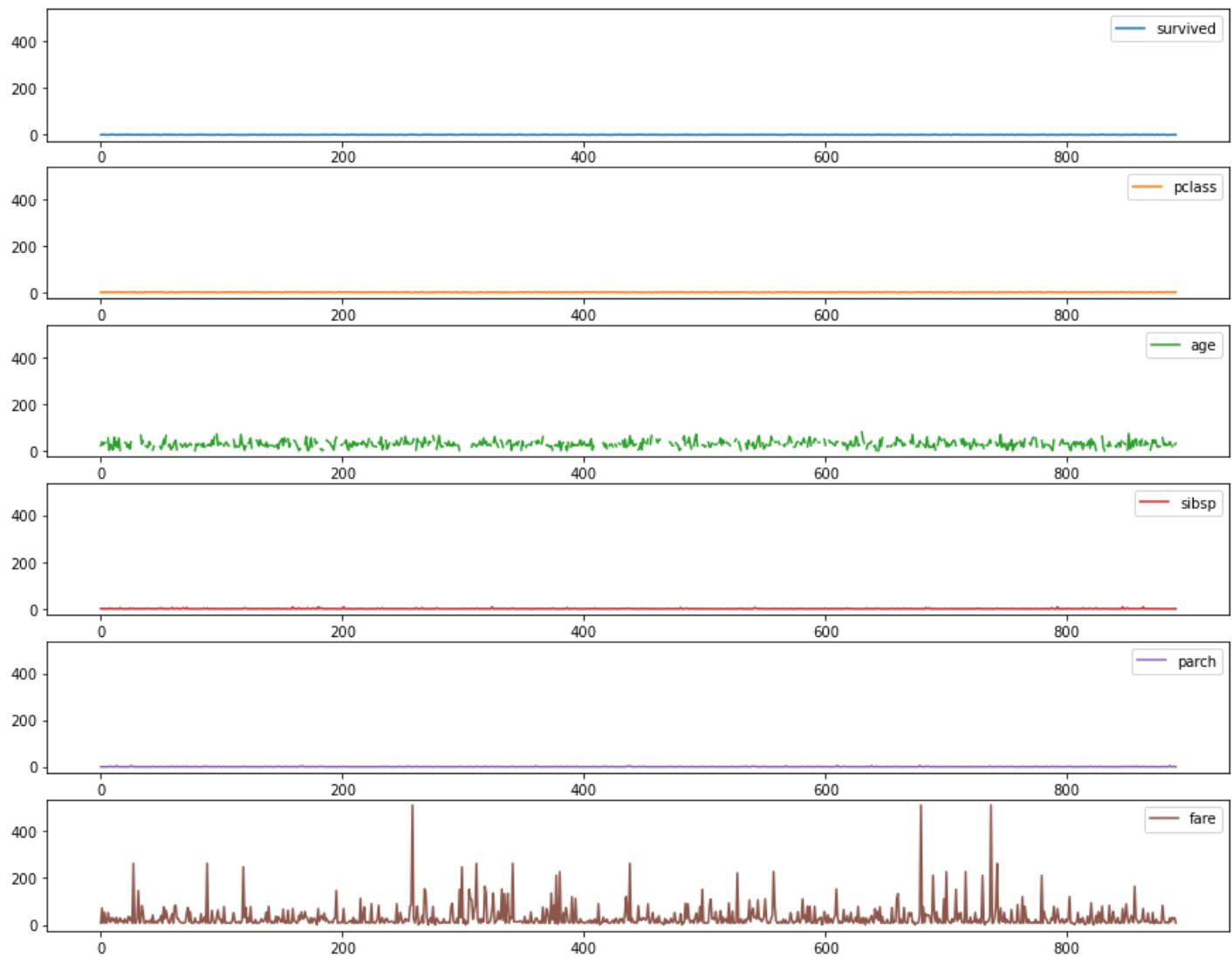
Share Y is not ideal because it homogenizes the scaling of the charts. They have their own y-axis but the range is the same. This can make the scaling look bad if the ranges vary drastically.

In [42]:

```
titanic.plot(figsize = (15, 12), subplots = True, sharex = False, sharey = True)
```

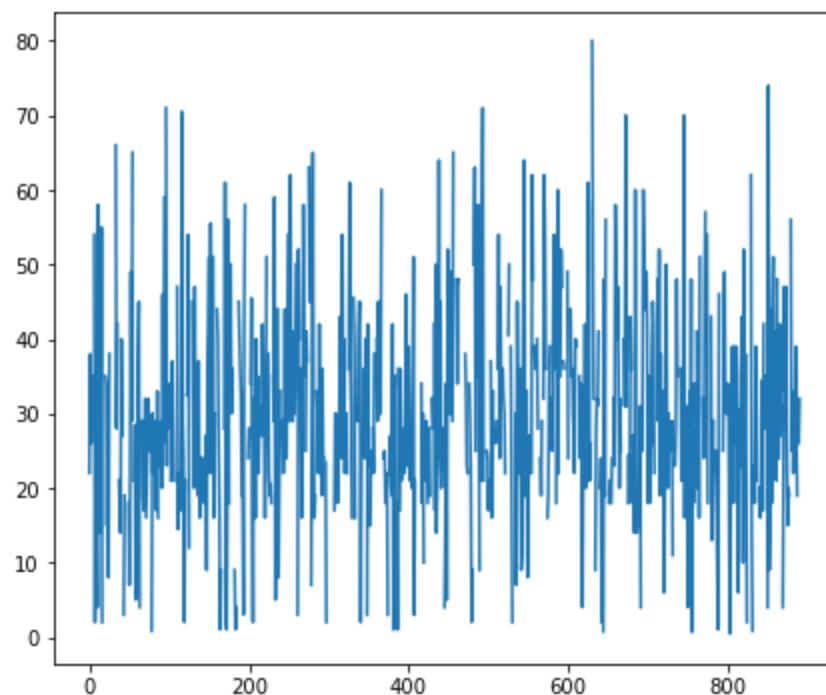
Out[42]:

```
array([<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>,  
<AxesSubplot:>, <AxesSubplot:>], dtype=object)
```



You can plot characteristics individually.

```
In [50]: titanic.age.plot(figsize = (7, 6))  
plt.show()
```



Important and dense Section.

.

You can change the fontsize using fontsize = number Color can change by using c = "color_name"

You can change the linestyle. linestyle = "-" by default. It can be changed to a dashed line by setting

Default line linestyle = "-"

Dashed line linestyle = "--"

Dotted line linestyle = ":"

There are many other things You can do.

You can create titles, legends, and labels.

To do this you must you Matplot lib plt. directly. Type plt. to see your options.

Common selections are plt.title or plt.legend etc.

Labels and titles should be followed by a "fontsize"

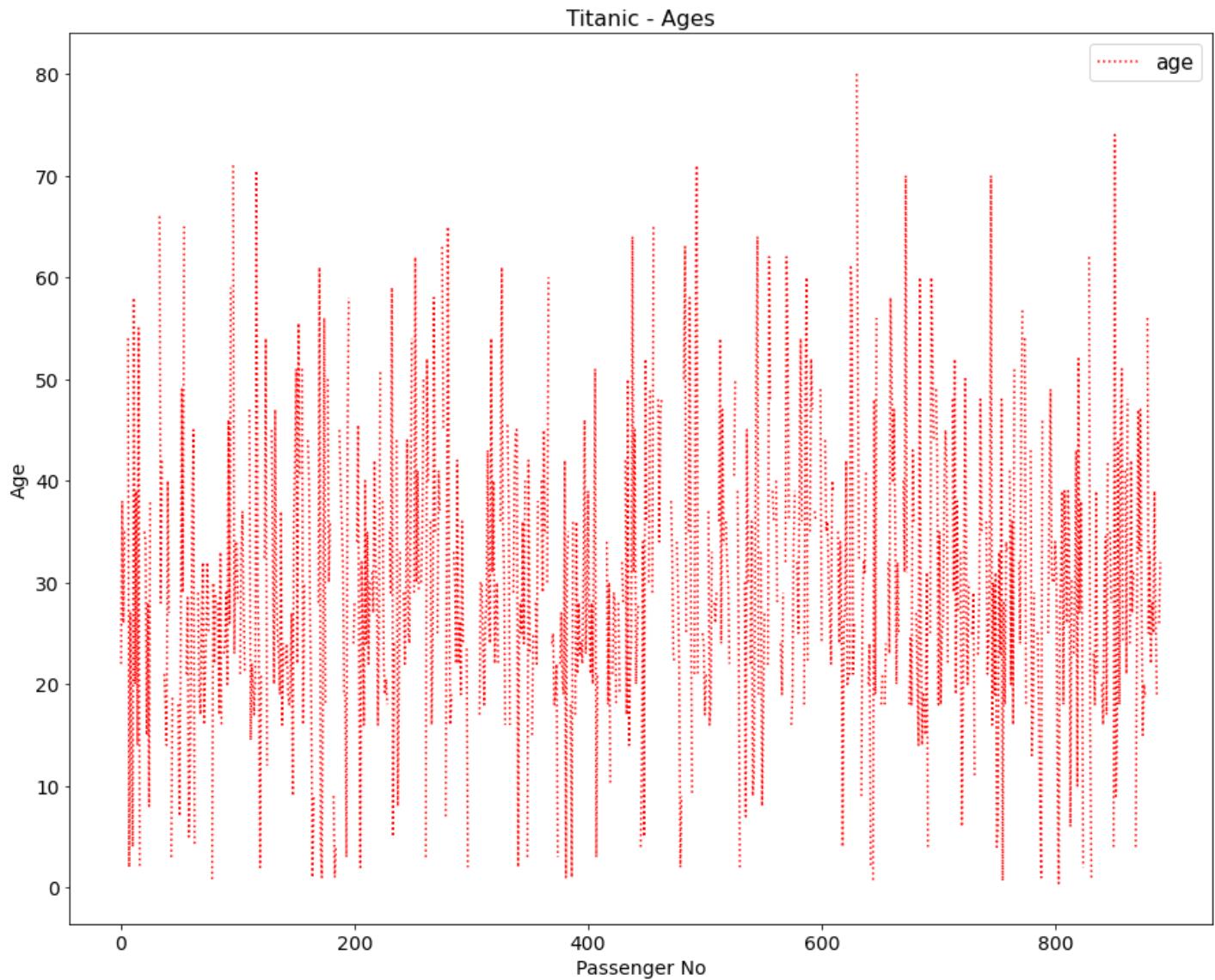
You can add a grid as well use plt.grid

In [78]:

```
titanic.age.plot(figsize = (15, 12), fontsize = 14, c = "red", linestyle = ":")
plt.title("Titanic - Ages", fontsize = 16)
plt.legend(loc = "best", fontsize = 15) # Set legend to left corner by setting location to
# plt.legend(loc = 3, fontsize = 15)
plt.xlabel("Passenger No", fontsize = 14)
plt.ylabel("Age", fontsize = 14)
# plt.grid()
```

Out[78]:

```
Text(0, 0.5, 'Age')
```



Limiting the scope of the graph on the x-axis.

Use `xlim` to limit the scope of the graph `xlim = (min, max)` to see a limited graph.

Limiting the X and Y using `xlim =()` and `ylim = ()` is important because it can prevent the high values of the graph from being cutoff.

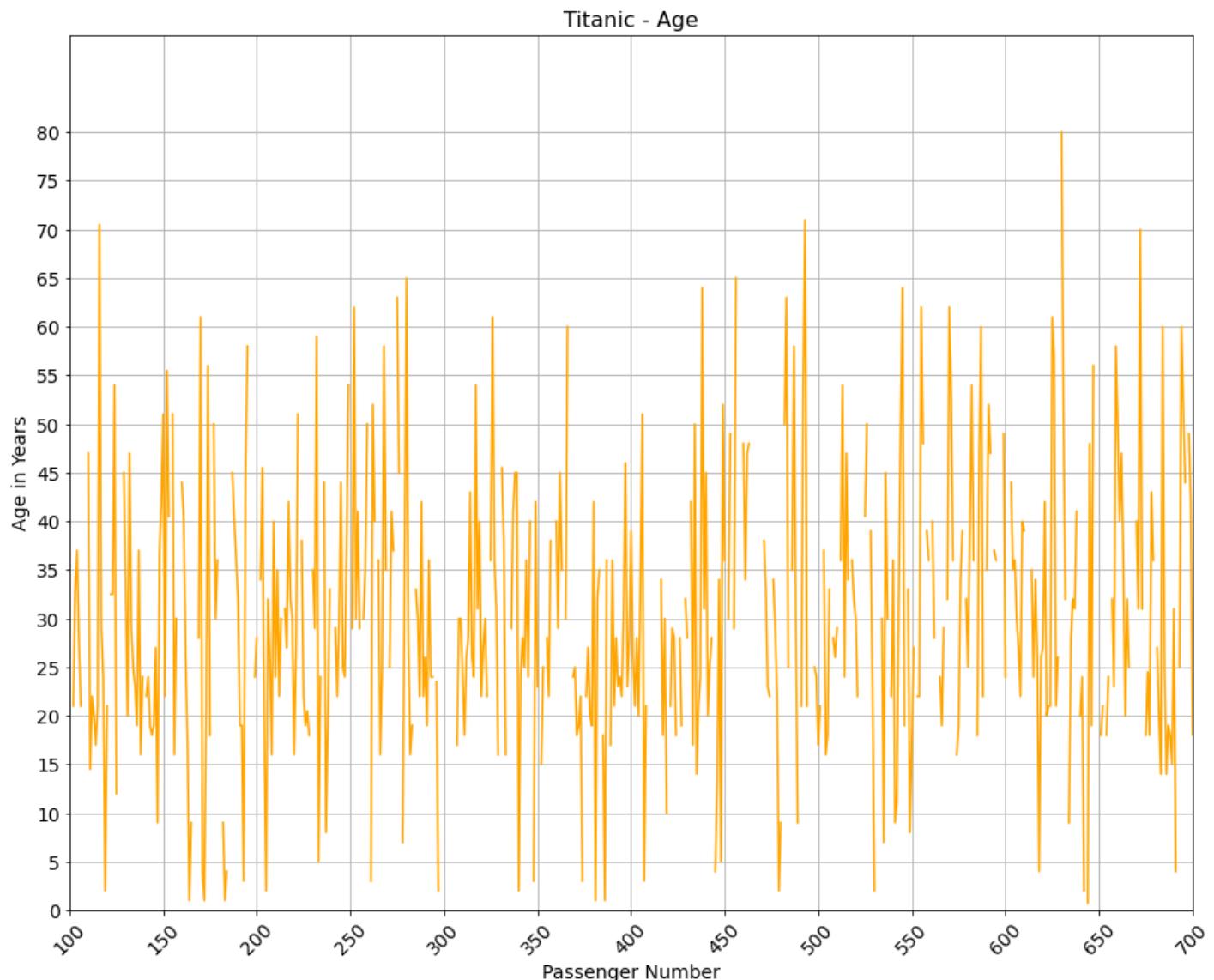
`xticks` and `yticks` takes a list of values that act as the spacing between each value of the axis. They should be equidistant from each other.

Use `rot` to rotate ticks by a certain amount of degrees.

In [103...]

```
titanic.age.plot(figsize = (15, 12), fontsize = 14, c = "orange", linestyle = "--", xlim = plt.grid()
plt.title("Titanic - Age", fontsize = 16)
plt.ylabel("Age in Years", fontsize = 14)
plt.xlabel("Passenger Number", fontsize = 14)
#
```

```
Out[103... Text(0.5, 0, 'Passenger Number')
```



Changing the style of the graph.

The data type makes it susceptible to be used with certain graph styles. You can see a list of those styles here:

```
In [106... plt.style.available
```

```
Out[106... ['Solarize_Light2',
 '_classic_test_patch',
 '_mpl-gallery',
 '_mpl-gallery-nogrid',
 'bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn',
 'seaborn-bright',
 'seaborn-colorblind',
 'seaborn-dark',
 'seaborn-dark-palette',
 'seaborn-darkgrid',
 'seaborn-deep',
 'seaborn-muted',
```

```
'seaborn-notebook',
'seaborn-paper',
'seaborn-pastel',
'seaborn-poster',
'seaborn-talk',
'seaborn-ticks',
'seaborn-white',
'seaborn-whitegrid',
'tableau-colorblind10']
```

In order to demonstrate the change a fresh graph will be made to show off the new style.

plot.style is very important press tab to show what you can do with it.

To show a new graph style:

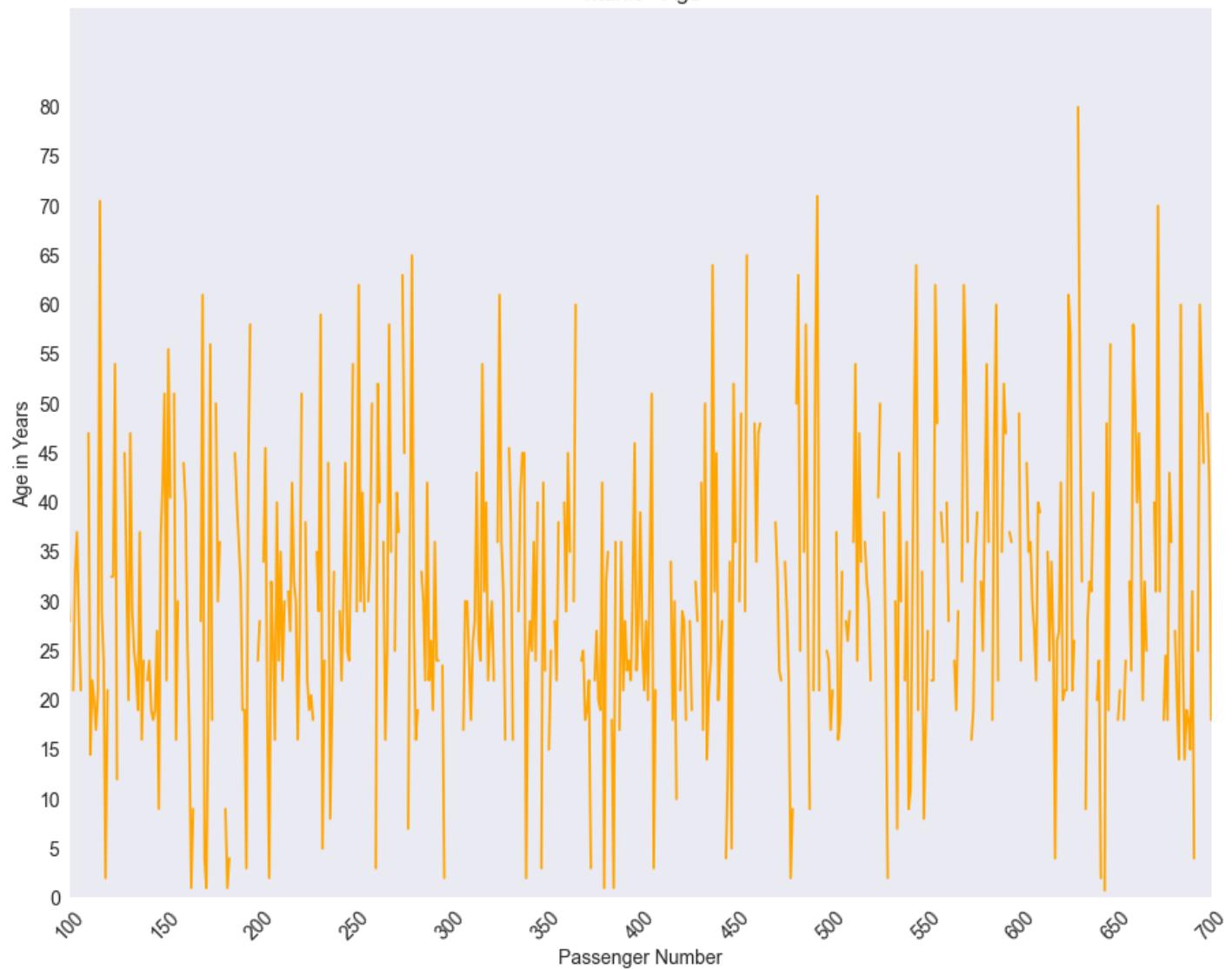
```
plt.style.use("style_name_from_avail_list")
```

Note that the plot.style.use after it is run will change the graph after it is ran regardless of order of appearance like other things in jupyter.

```
In [117...]: plt.style.use("seaborn")
```

```
In [118...]: titanic.age.plot(figsize = (15, 12), fontsize = 14, c = "orange", linestyle = "--", xlim = plt.grid()
plt.title("Titanic - Age", fontsize = 16)
plt.ylabel("Age in Years", fontsize = 14)
plt.xlabel("Passenger Number", fontsize = 14)
```

```
Out[118...]: Text(0.5, 0, 'Passenger Number')
```



Another example of a graph that is styled.

To Apply a New Graph you must properly reset.

In order to apply a new style the old style has to be properly reset.

Note that the plt.grid() changes the default value of grids for styles. Grids should be added as a last step.

In [152...]

```
# Reset Grid Style
plt.rcParams.update(plt.rcParamsDefault)
%matplotlib inline
```

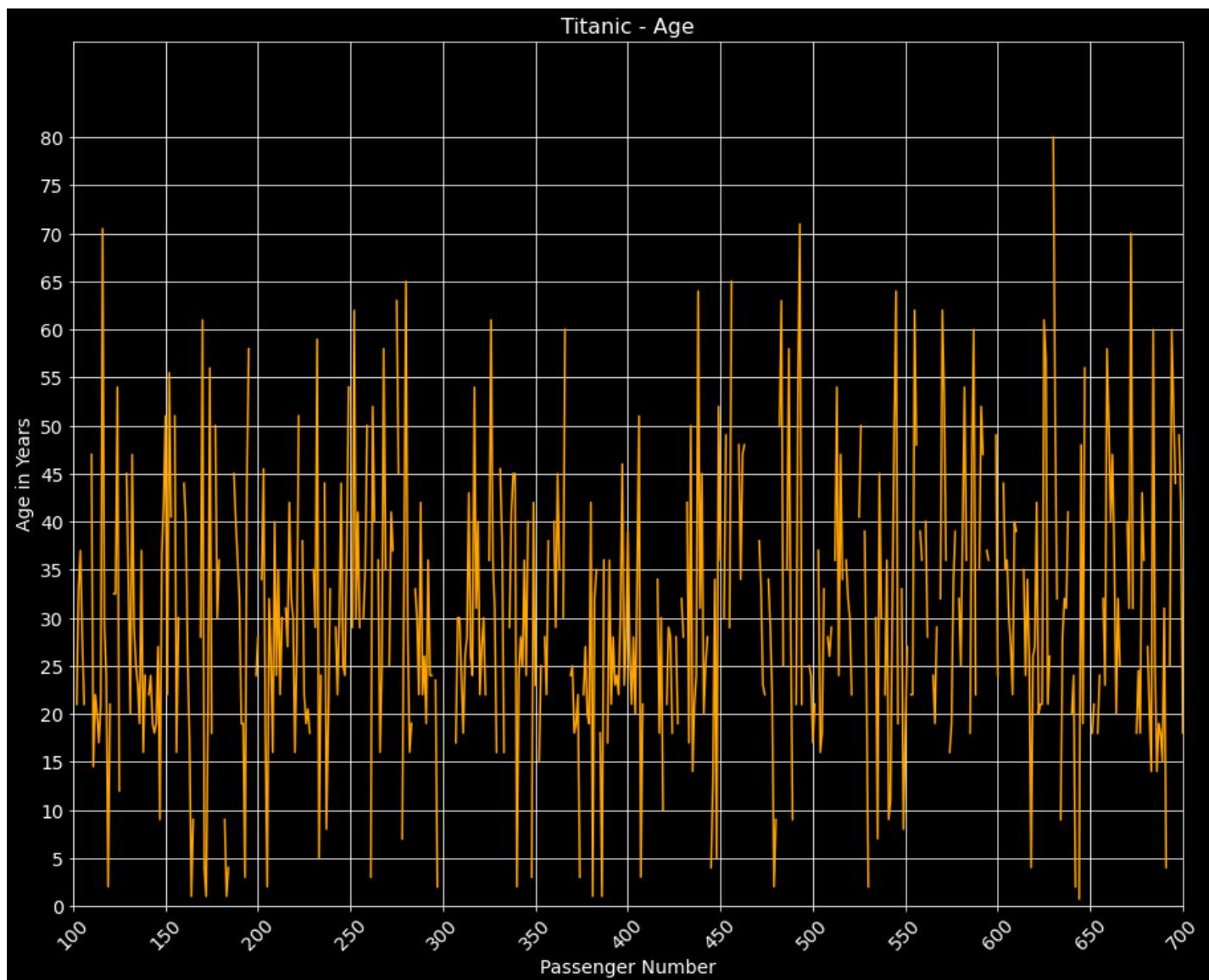
In [153...]

```
plt.style.use("dark_background")
```

In [154...]

```
titanic.age.plot(figsize = (15, 12), fontsize = 14, c = "orange", linestyle = "-",
plt.grid()
plt.title("Titanic - Age", fontsize = 16)
plt.ylabel("Age in Years", fontsize = 14)
```

```
plt.xlabel("Passenger Number", fontsize = 14)  
plt.show()
```



In [104]:

```
## Changig the intervals for the graph. These are used above for the xticks = xticks1 etc.
```

In [94]:

```
xticks1 = [x for x in range(0, 900, 50)]  
xticks1
```

Out[94]:

```
[0,  
 50,  
 100,  
 150,  
 200,  
 250,  
 300,  
 350,  
 400,  
 450,  
 500,  
 550,  
 600,  
 650,  
 700,  
 750,  
 800,  
 850]
```

```
In [95]:  
    yticks1 = [y for y in range(0, 81, 5)]  
    yticks1
```

```
Out[95]: [0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80]
```

Histograms

Histograms are charts that count how many values fit into a certain interval.

```
In [155...]  
    # Reset Grid Style  
    plt.rcParams.update(plt.rcParamsDefault)  
    %matplotlib inline
```

```
In [ ]:  
    import pandas as pd  
    import matplotlib.pyplot as plt
```

```
In [159...]  
    titanic = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

```
In [160...]  
    titanic.head()
```

```
Out[160]:  
    survived  pclass  sex  age  sibsp  parch  fare  embarked  deck  
    0          0      3  male  22.0      1      0    7.2500      S    NaN  
    1          1      1  female  38.0      1      0   71.2833      C      C  
    2          1      3  female  26.0      0      0    7.9250      S    NaN  
    3          1      1  female  35.0      1      0   53.1000      S      C  
    4          0      3  male  35.0      0      0    8.0500      S    NaN
```

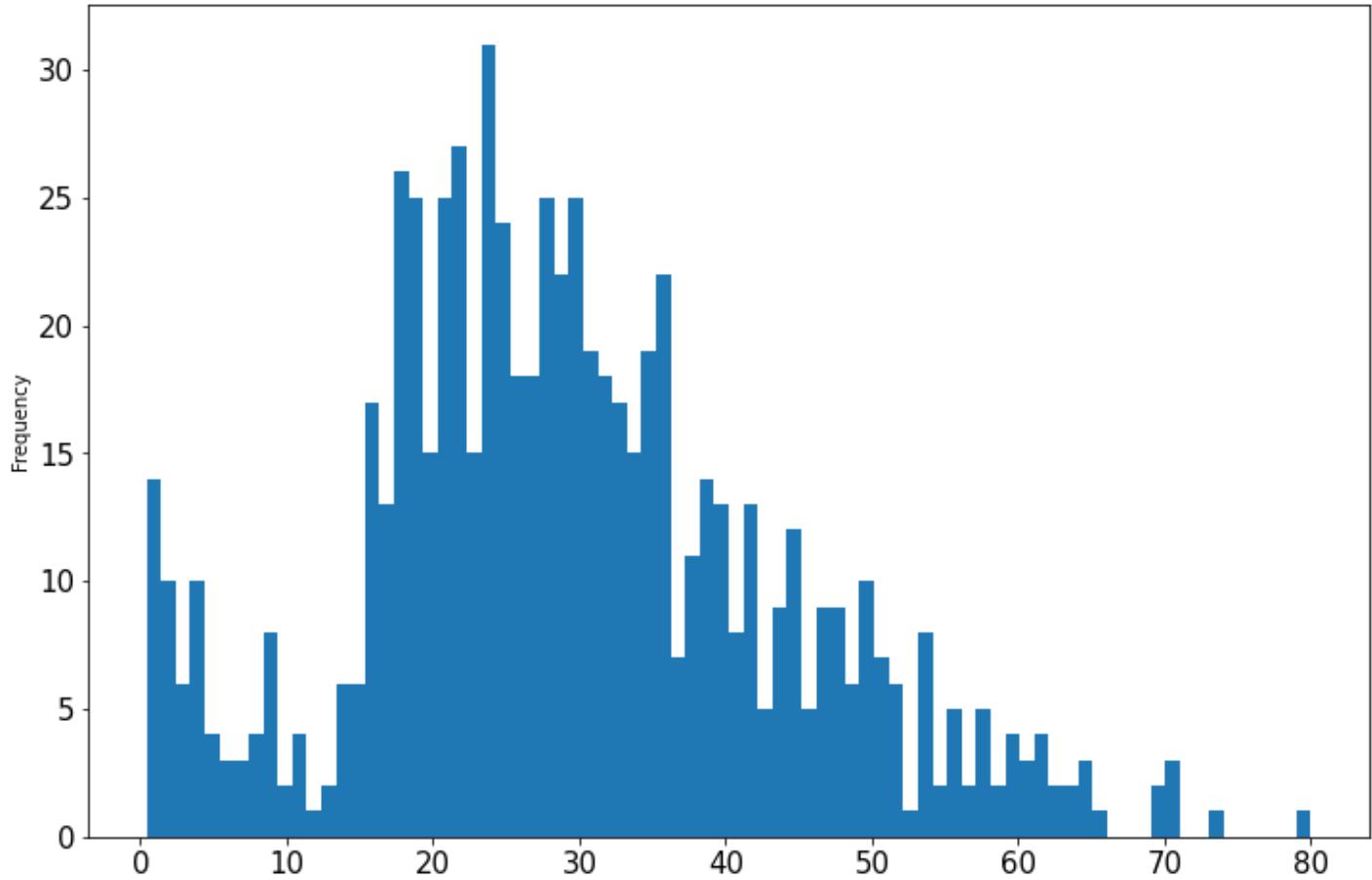
```
In [162...]  
    titanic.age.value_counts(bins = 10)
```

```
Out[162]:  
    (16.336, 24.294]      177  
    (24.294, 32.252]      169  
    (32.252, 40.21]       118  
    (40.21, 48.168]        70  
    (0.339, 8.378]         54  
    (8.378, 16.336]        46  
    (48.168, 56.126]        45  
    (56.126, 64.084]        24  
    (64.084, 72.042]         9  
    (72.042, 80.0]           2  
    Name: age, dtype: int64
```

We can make the kind of chart into a histogram by using kind = "hist"

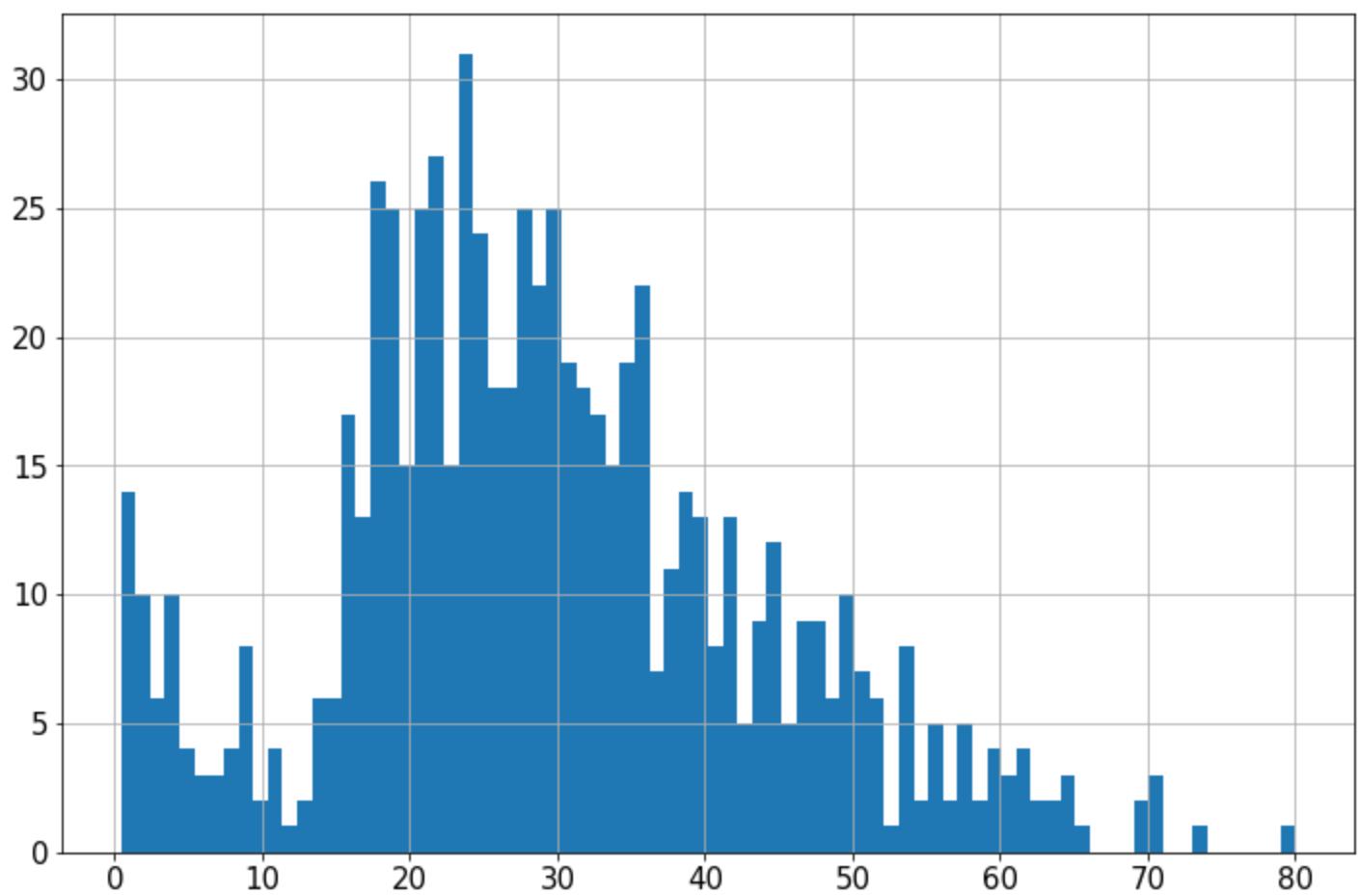
Bins shows how the data can be subdivided into different intervals. The higher the bins the smaller the intervals.

```
In [195...]  
titanic.age.plot(kind = "hist", figsize = (12,8), fontsize = 15, bins = 80)  
plt.show()  
#You can use cumulative = True here as well.  
# density = True works here as well
```



Alternate way to make a histogram. Use .hist() method to create one.

```
In [179...]  
titanic.age.hist(figsize = (12, 8), bins = 80, xlabelsize = 15, ylabelsize = 15)  
plt.show()  
# By changing the kind = "hist" on the plot method you get the yaxis labeled. Here it is 1
```

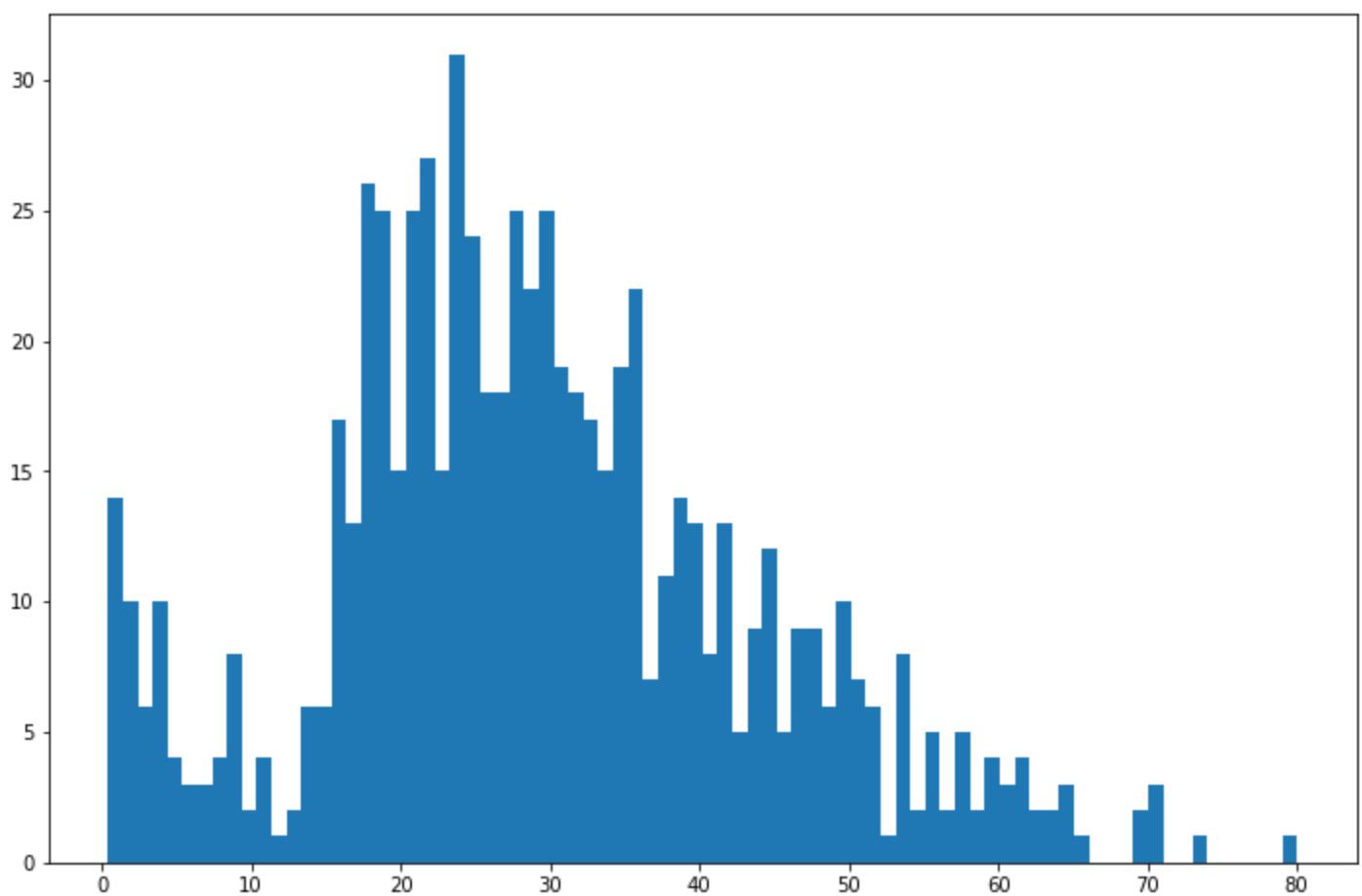


You can use the plt. way to make a histogram chart as well.

You must use .dropna() at the end to get rid of missing values or you will get an error message.

In [185...]

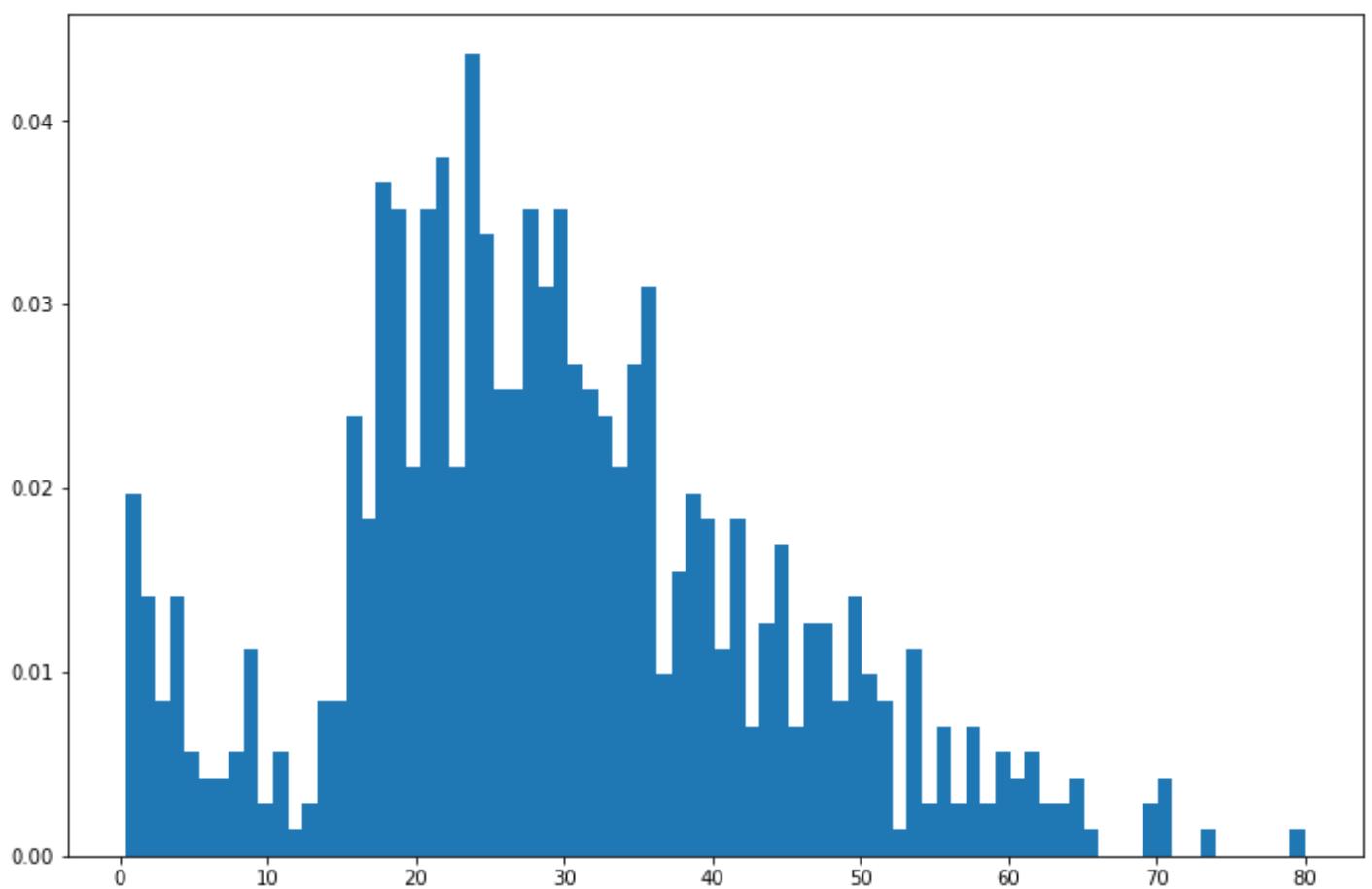
```
plt.figure(figsize = (12, 8))
plt.hist(titanic.age.dropna(), bins = 80, density = False, cumulative = False)
plt.show()
```



Density will show a percentage of the values each datapoint will get.

density = True

```
In [190...]:  
plt.figure(figsize = (12, 8))  
plt.hist(titanic.age.dropna(), bins = 80, density = True, cumulative = False)  
plt.show()
```



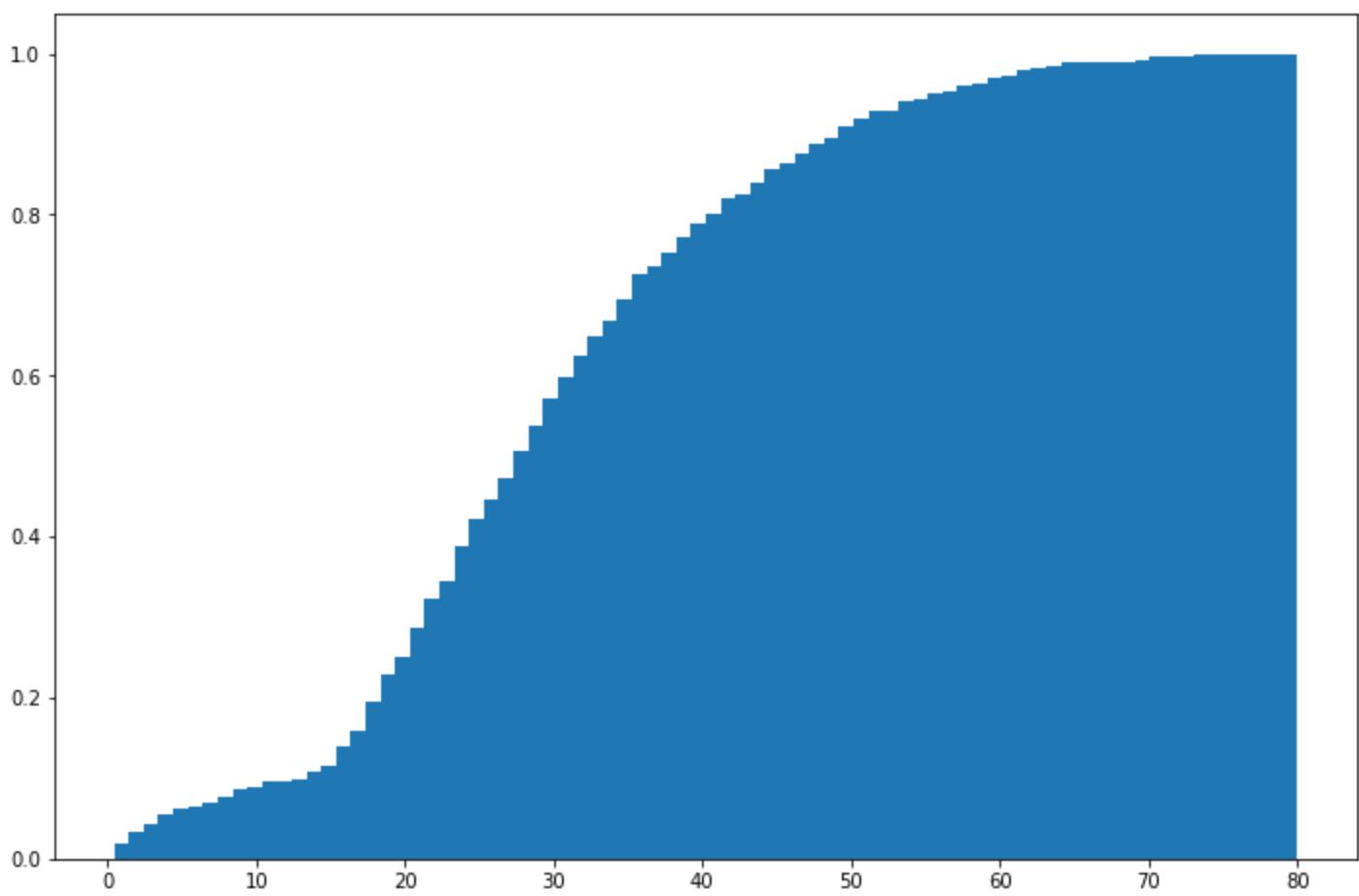
Culmalative will show the percent of values that are at or below a certain number.

Cululative = True

This can be used on the titanic.plot()

In [197...]

```
plt.figure(figsize = (12, 8))
plt.hist(titanic.age.dropna(), bins = 80, density = True, cumulative = True)
plt.show()
```



Scatterplots

In [200...]

```
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use("seaborn")
```

In [201...]

```
titanic = pd.read_csv(r"C:\Users\alonz\Downloads\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

In [202...]

```
titanic.head()
```

Out[202...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN

To make a scatterplot you need to change kind = "scatter" and you need to define the x and y column.

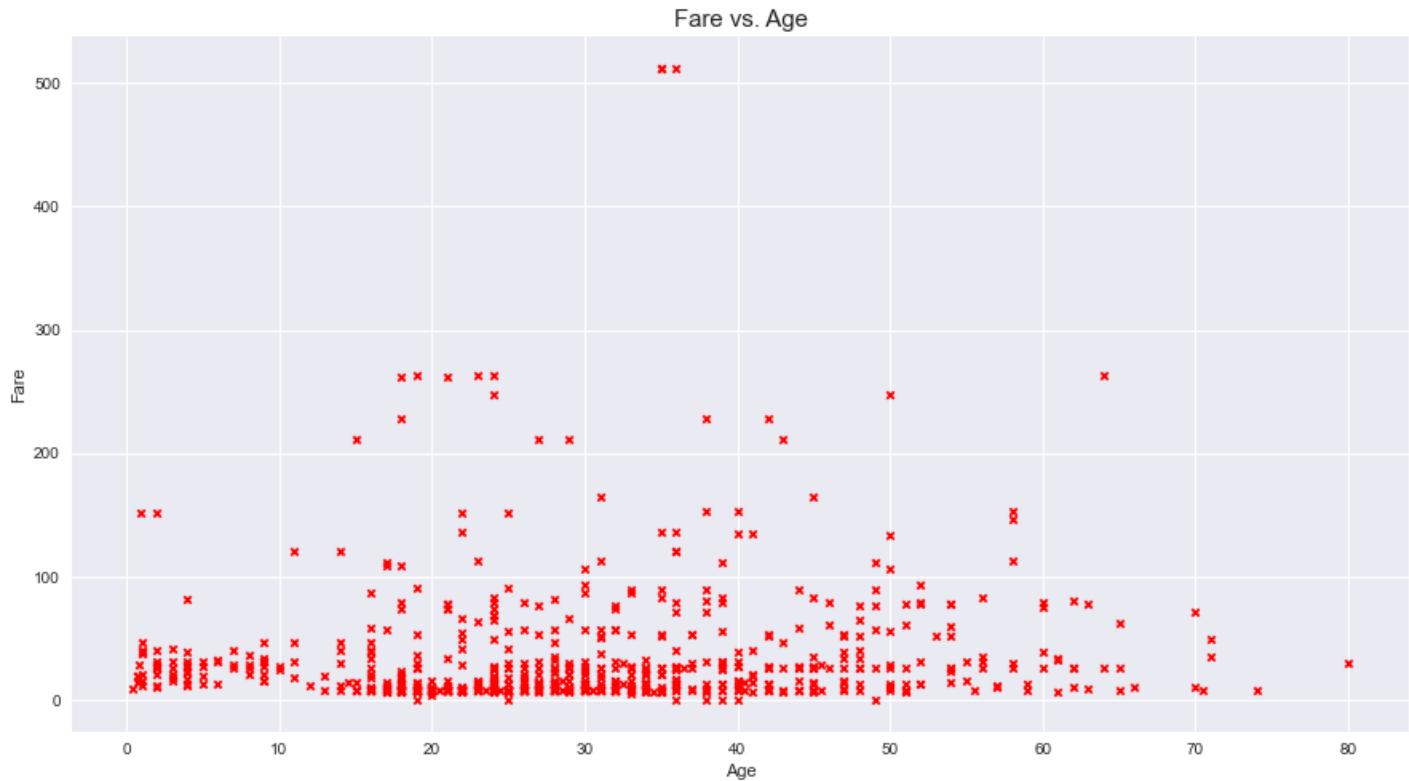
You can change the dot shape. You need to set marker = "x" or "X" or "o" or "D". Note that "o" is the default.

s = interger changes the size of the dots.

In [217...]

```
titanic.plot(kind = "scatter", figsize = (15, 8), x = "age", y = "fare", c = "red", marker="x", s=100)
```

Out[217...]



No correlation is detected between price and age.

3D Scatterplot In order to make it 3D, just add c = "other_characteristic"

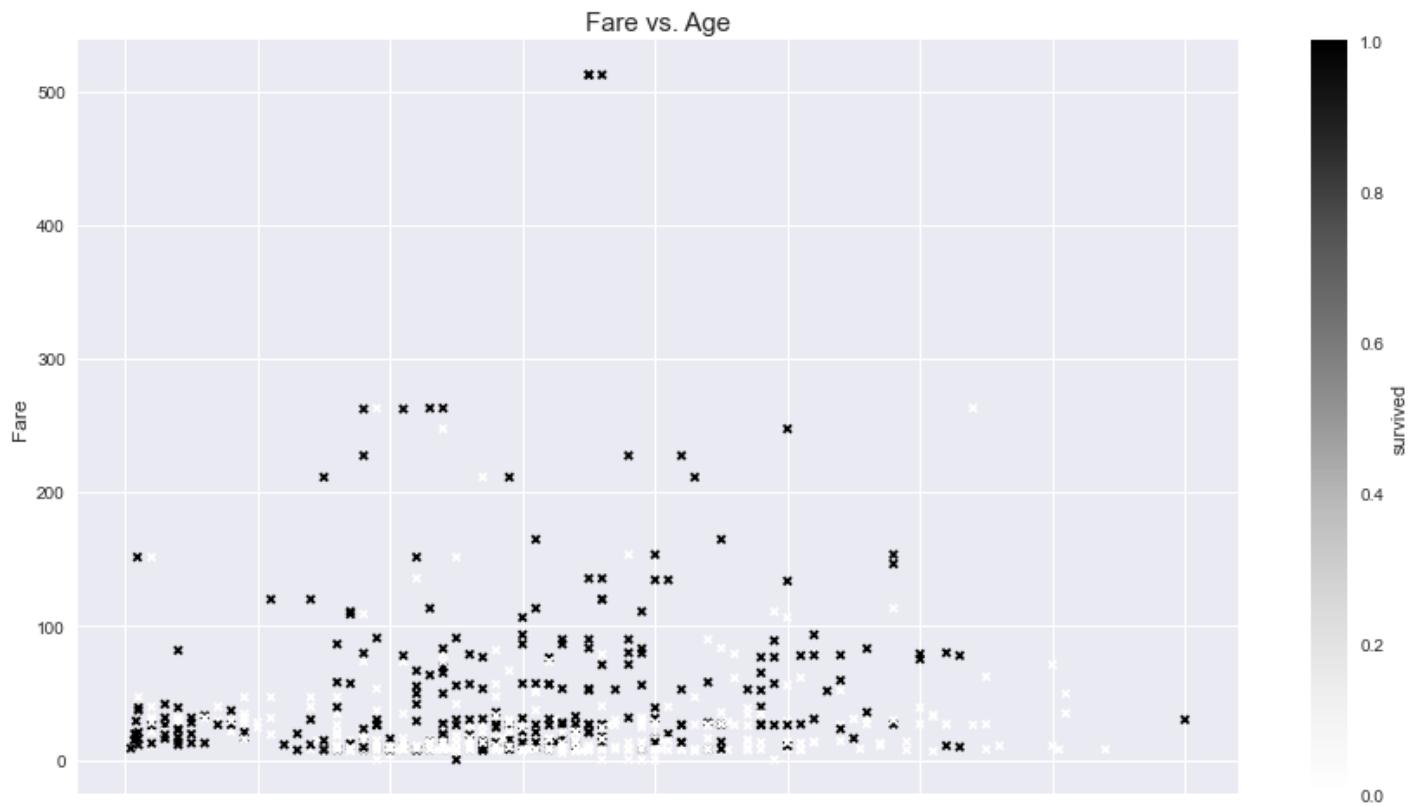
Yes usually c = "color" however it can be used to add depth to the data that is being examined. c becomes like a z variable essentially.

In [221...]

```
titanic.plot(kind = "scatter", figsize = (15, 8), x = "age", y = "fare", c = "survived", marker="x", s=100)
```

Out[221...]

[]



Add Color to a 3D Scatterplot using colormap = "color_map_name"

You can add color to a 3D scatterplot by using one of the default values that are provided in the documentation.

Perceptually Uniform Sequential colormaps



In [224...]

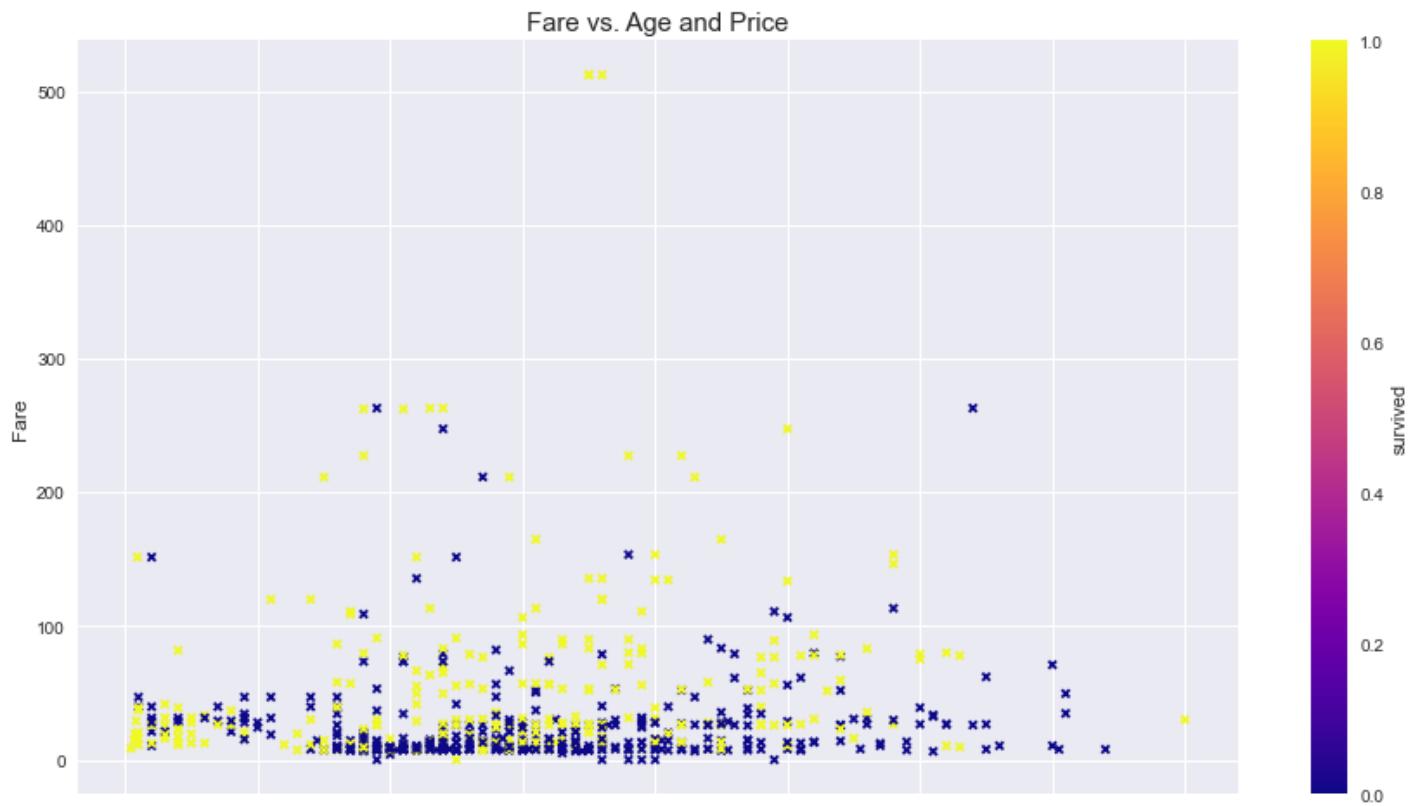
```
## Documentation can be found here: https://matplotlib.org/stable/tutorials/colors/colormaps.html
```

In [232...]

```
titanic.plot(kind = "scatter", figsize = (15, 8), x = "age", y = "fare", c = "survived", r)
plt.title("Fare vs. Age and Price", fontsize = 15)
plt.xlabel("Age", fontsize = 12)
plt.ylabel("Fare", fontsize = 12)
plt.plot()
```

Out[232...]

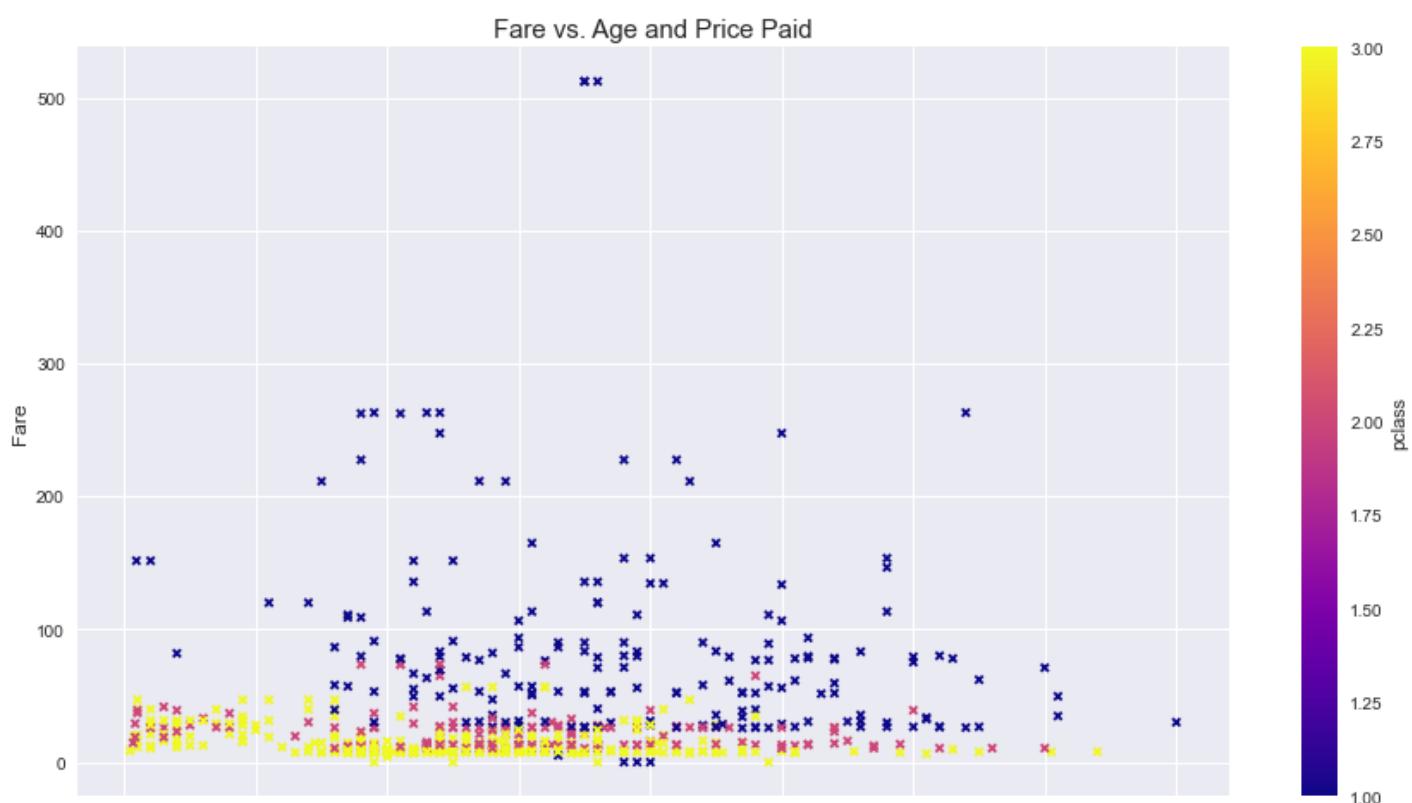
[]



Age vs. Fare Vs Class

```
In [229...]: titanic.plot(kind = "scatter", figsize = (15, 8), x = "age", y = "fare", c = "pclass", ma...  
plt.title("Fare vs. Age and Price Paid", fontsize = 15)  
plt.xlabel("Age", fontsize = 12)  
plt.ylabel("Fare", fontsize = 12)  
plt.plot()
```

```
Out[229...]: []
```



Working with Seaborn

In [234...]

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [235...]

```
titanic= pd.read_csv(r"C:\Users\alonz\Downloads\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

In [236...]

```
titanic.head()
```

Out[236...]

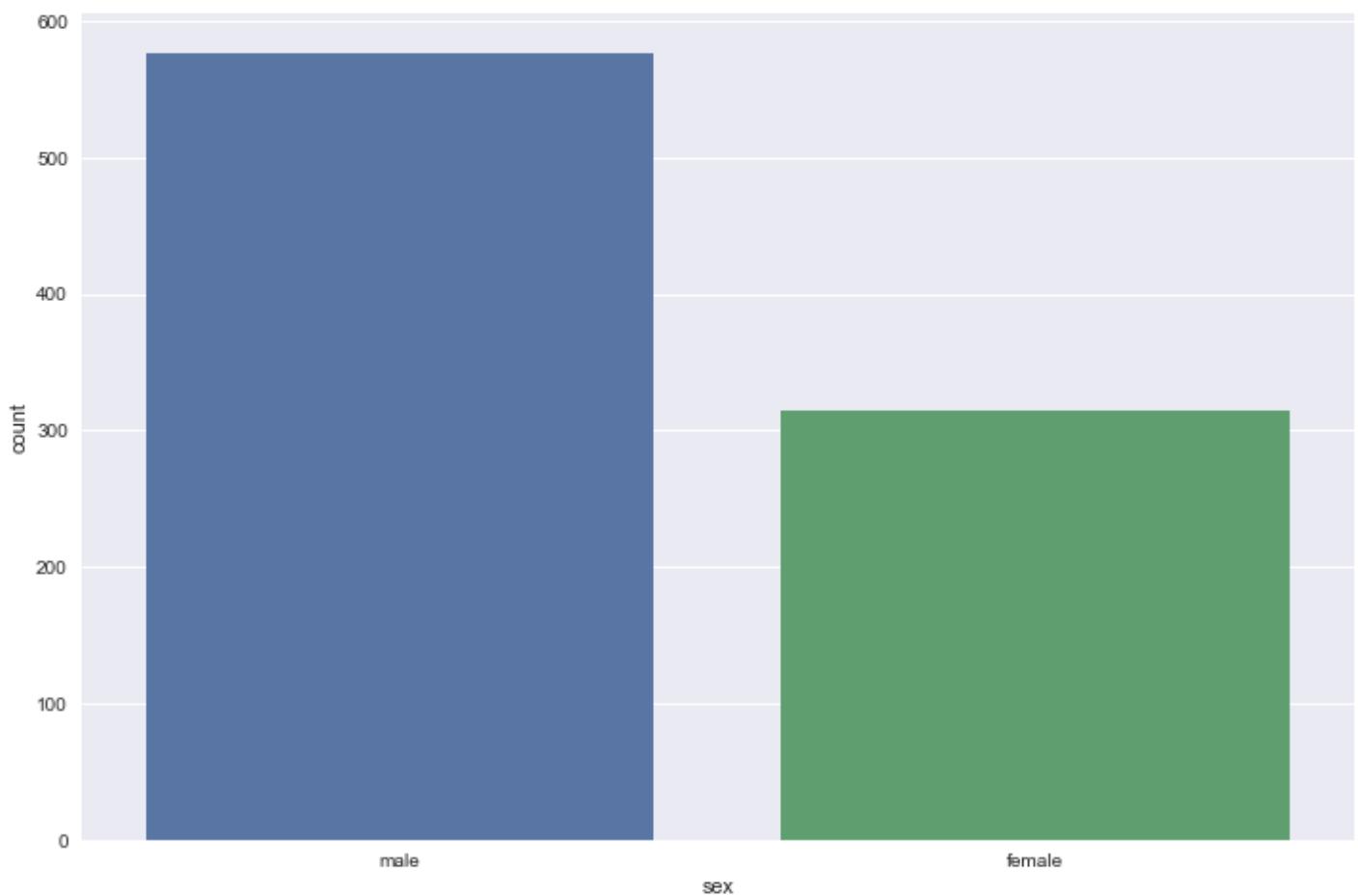
	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN

Countplot allows you to be able to count the amount of elements in each group.

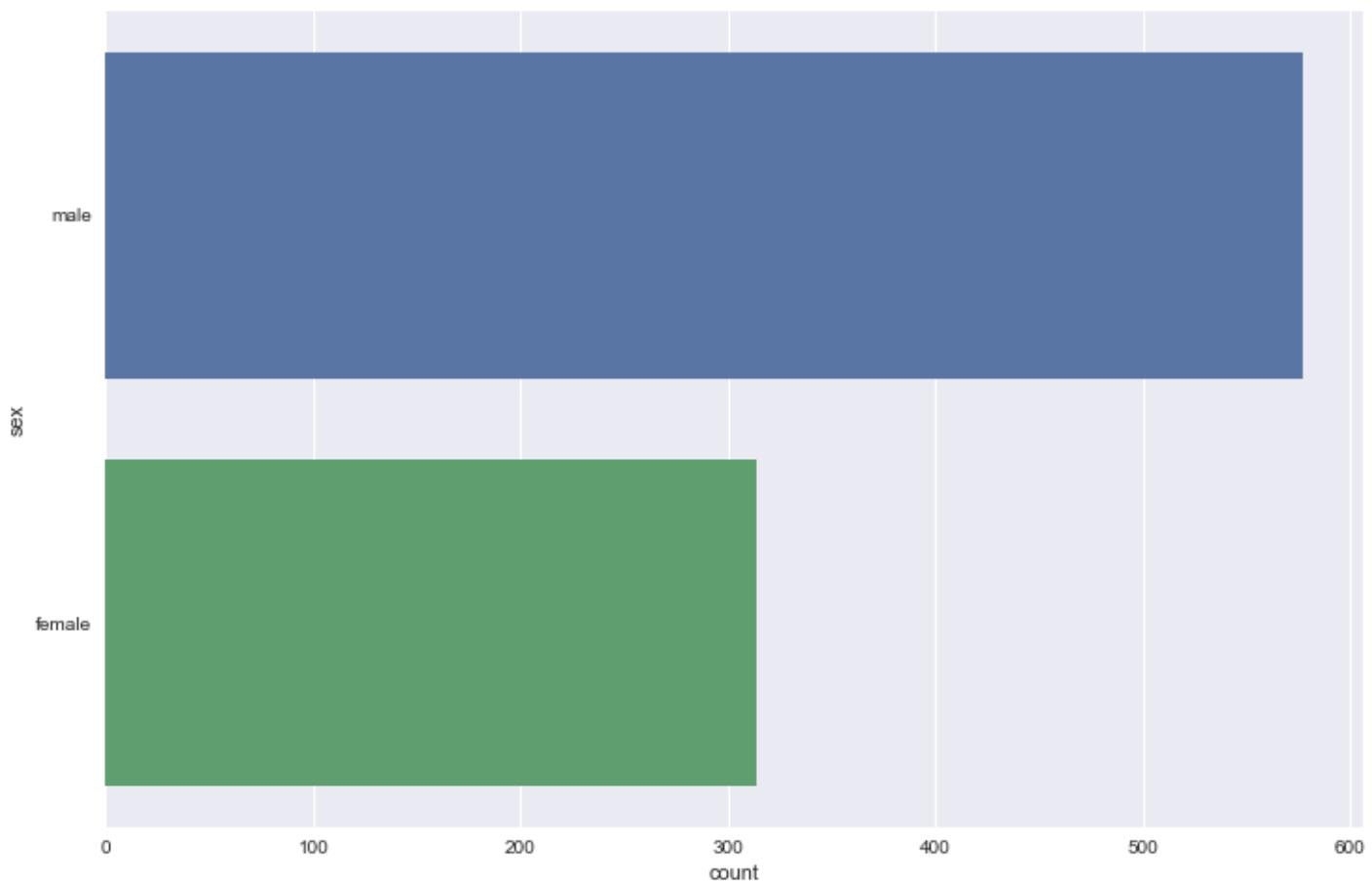
```
data = the_name_of_csv
x = "characteristic_measured"
```

In [240...]

```
plt.figure(figsize = (12,8))
sns.countplot(data = titanic, x = "sex")
plt.show()
```



```
In [241]:  
plt.figure(figsize = (12,8))  
sns.countplot(data = titanic, y = "sex")  
plt.show()
```

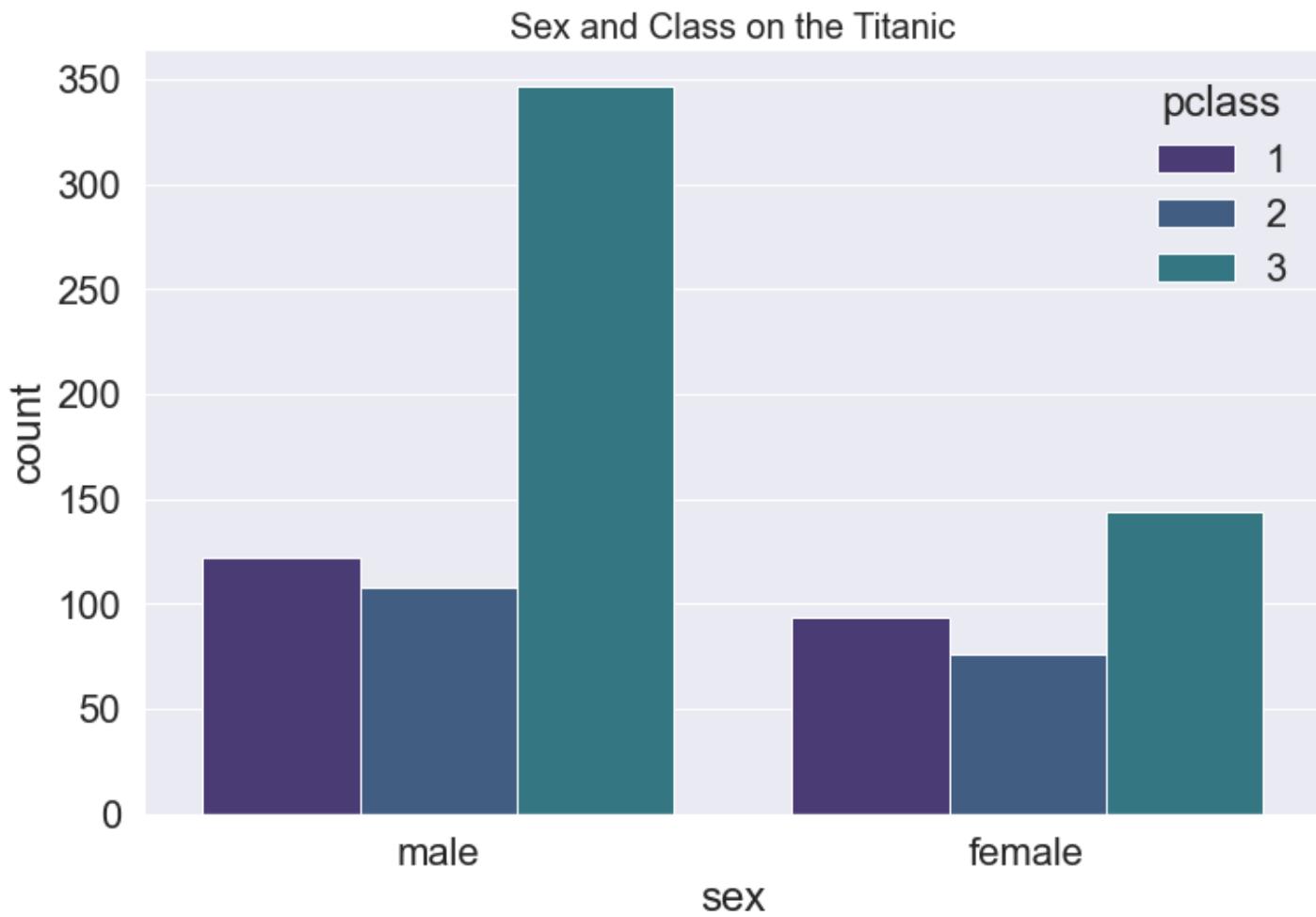


You can pass in other categorical data to be shown on the chart.

```
hue = "characteristic_name"
```

In [253...]

```
# For sns you need to use font_scale
plt.figure(figsize= (12,8))
sns.countplot(data = titanic, x = "sex", hue = "pclass")
sns.set(font_scale =2, palette = "viridis")
plt.title("Sex and Class on the Titanic", fontsize = 20)
plt.show()
```



Categorical Plots

There are other plots besides countplots. There are swarmplots, violinplot, barplot, and pointplot.

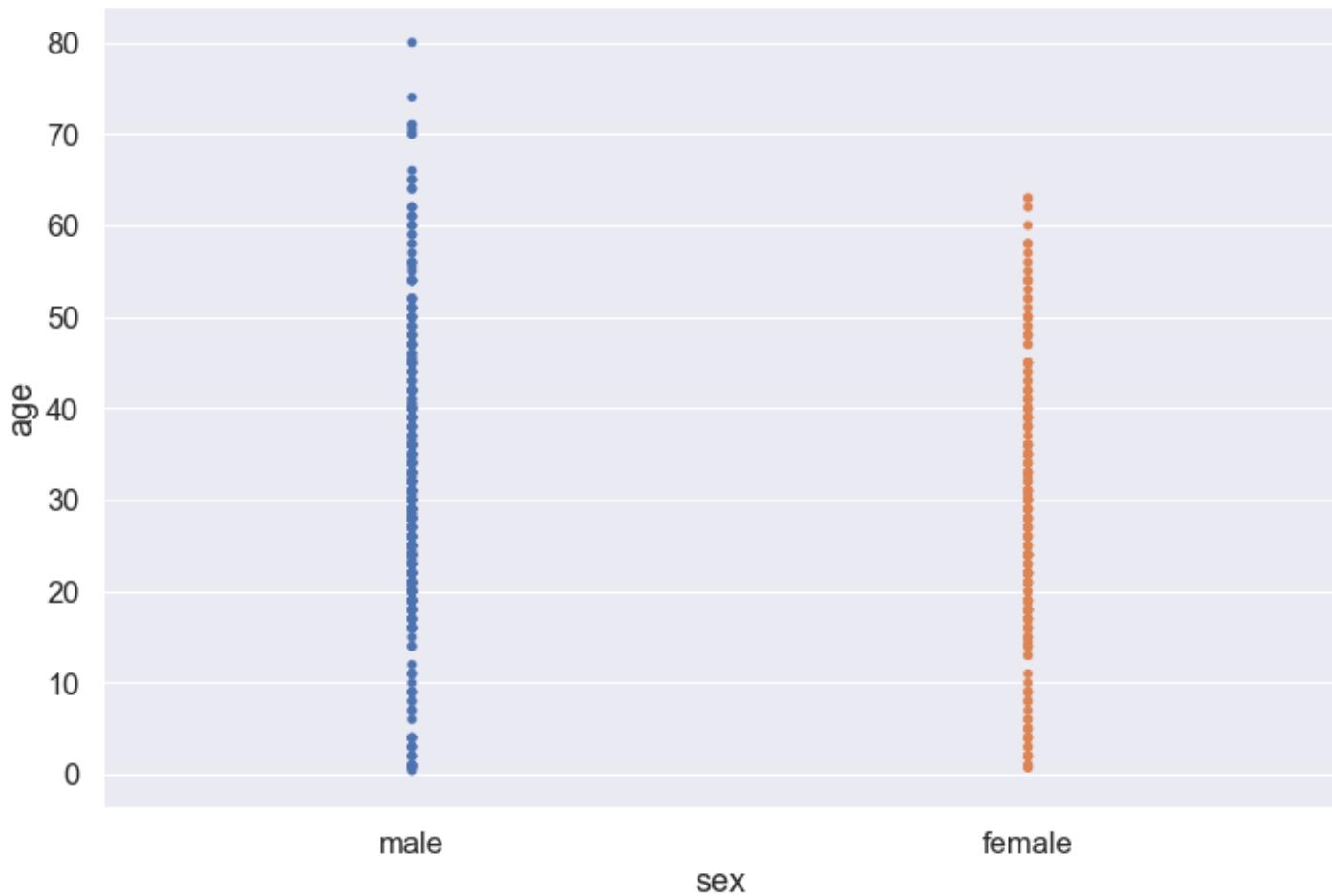
sns.stripplot() combines categorical and numerical values. The categorical value should be the x, and the numerical should be the y.

Each plot represents one person. The color shows the sex of the person, on the x-axis. The placement of the dot on the y-axis shows the age of the individual.

In [268...]

```
plt.figure(figsize = (12,8))
```

```
sns.set(font_scale = 1.5)
sns.stripplot(data = titanic, x = "sex", y = "age", jitter = False, hue = None, dodge = Fa
plt.show()
```

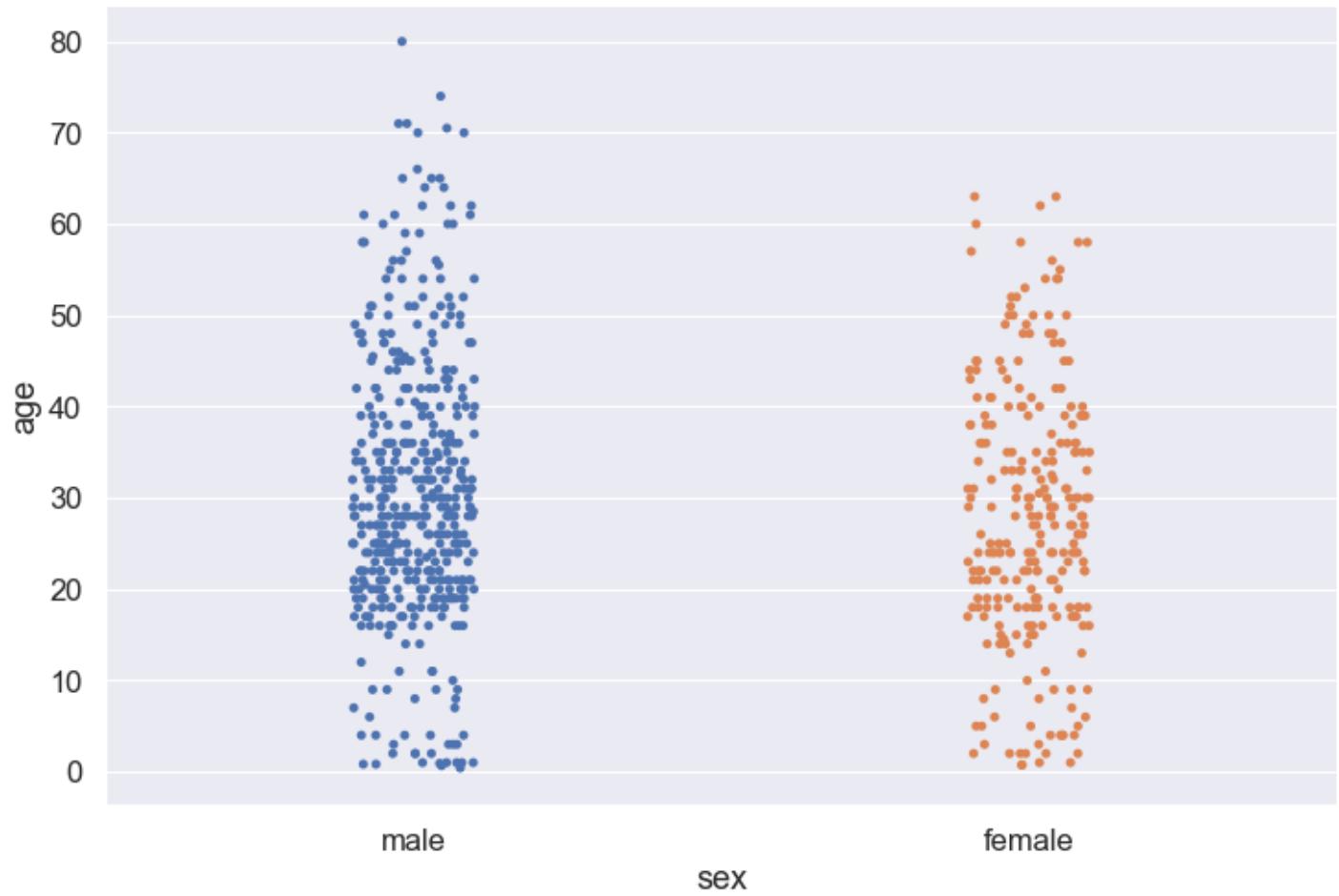


When jitter = True you can individual values.

It can be seen how many points fall within a certain range as the data points do not stack on top of each other.

In [274...]

```
plt.figure(figsize = (12,8))
sns.set(font_scale = 1.5)
sns.stripplot(data = titanic, x = "sex", y = "age", jitter = True, hue = None, dodge = Fal
plt.show()
```



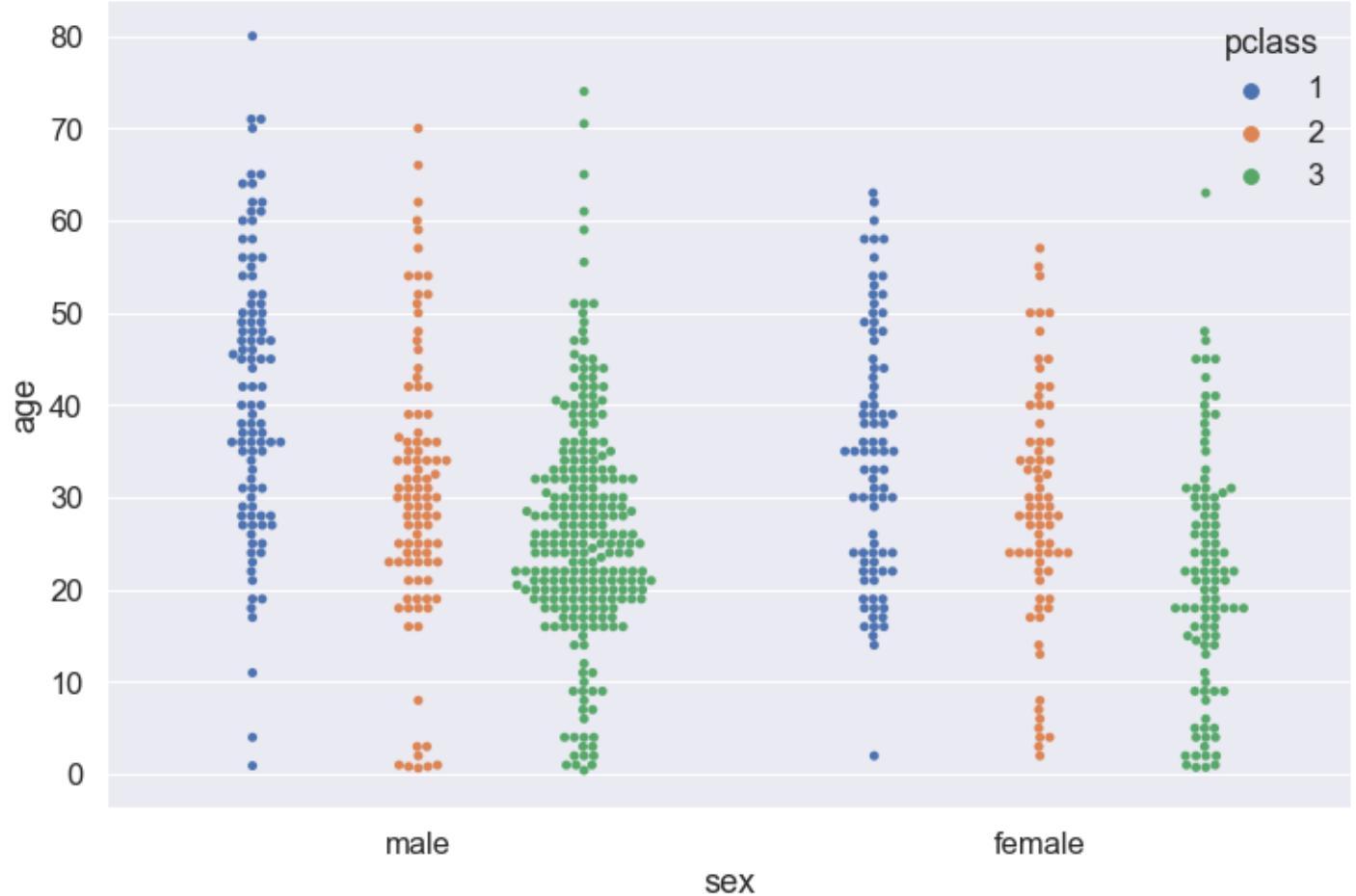
When dodge = True the values are seperated into individual columns.

The hue value is seperated into different columns. The hue can be thought of like the "c" column.

Below a sns.swarmplot is shown. A swarmplot is superior to the sns.stripplot even when the jitter is True because the dots are arranged in a more ordered fashion.

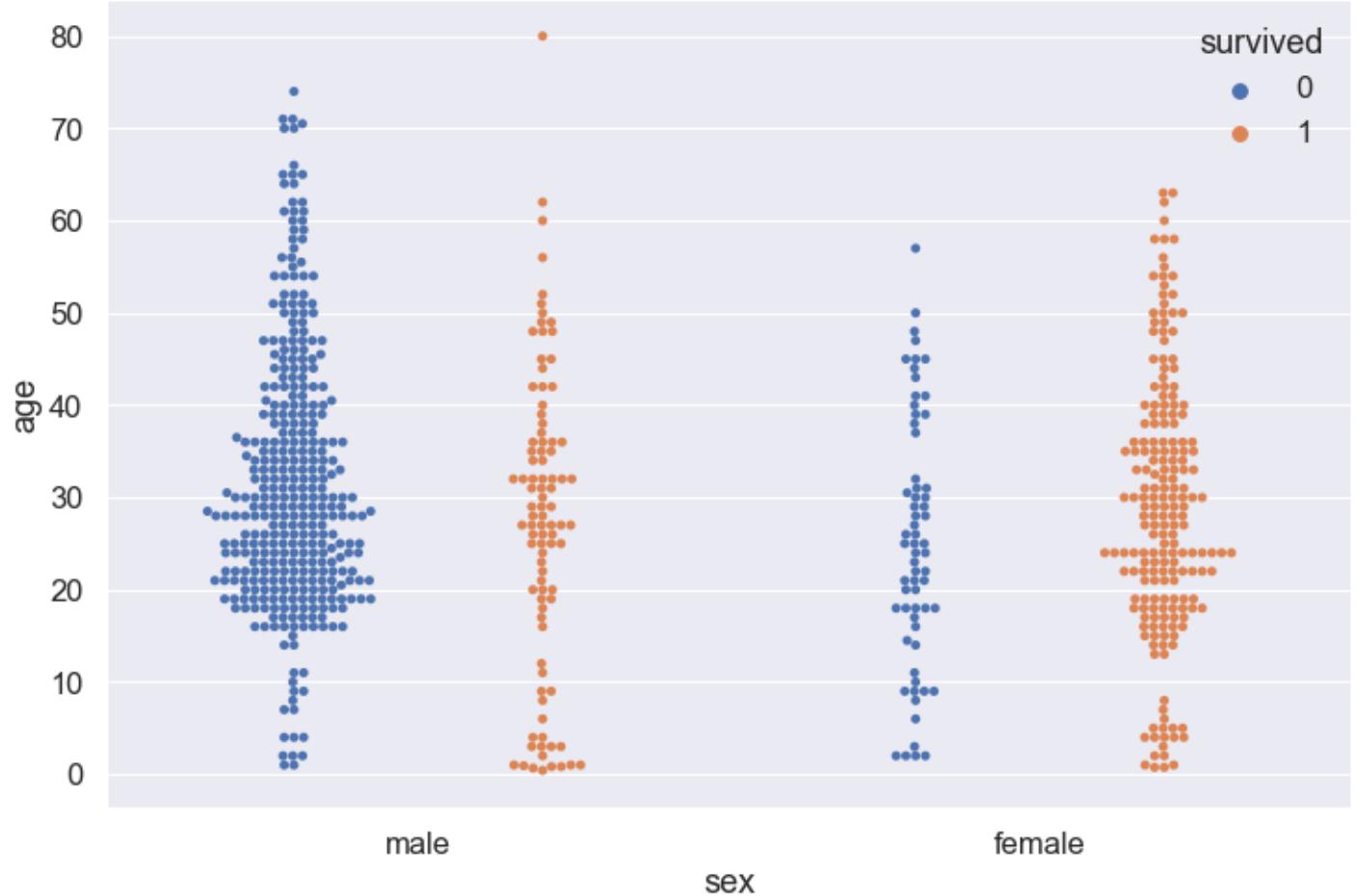
In [276...]

```
plt.figure(figsize =(12,8))
sns.set(font_scale =1.5)
sns.swarmplot(data = titanic, x = "sex", y = "age", hue = "pclass", dodge = True)
plt.show()
```



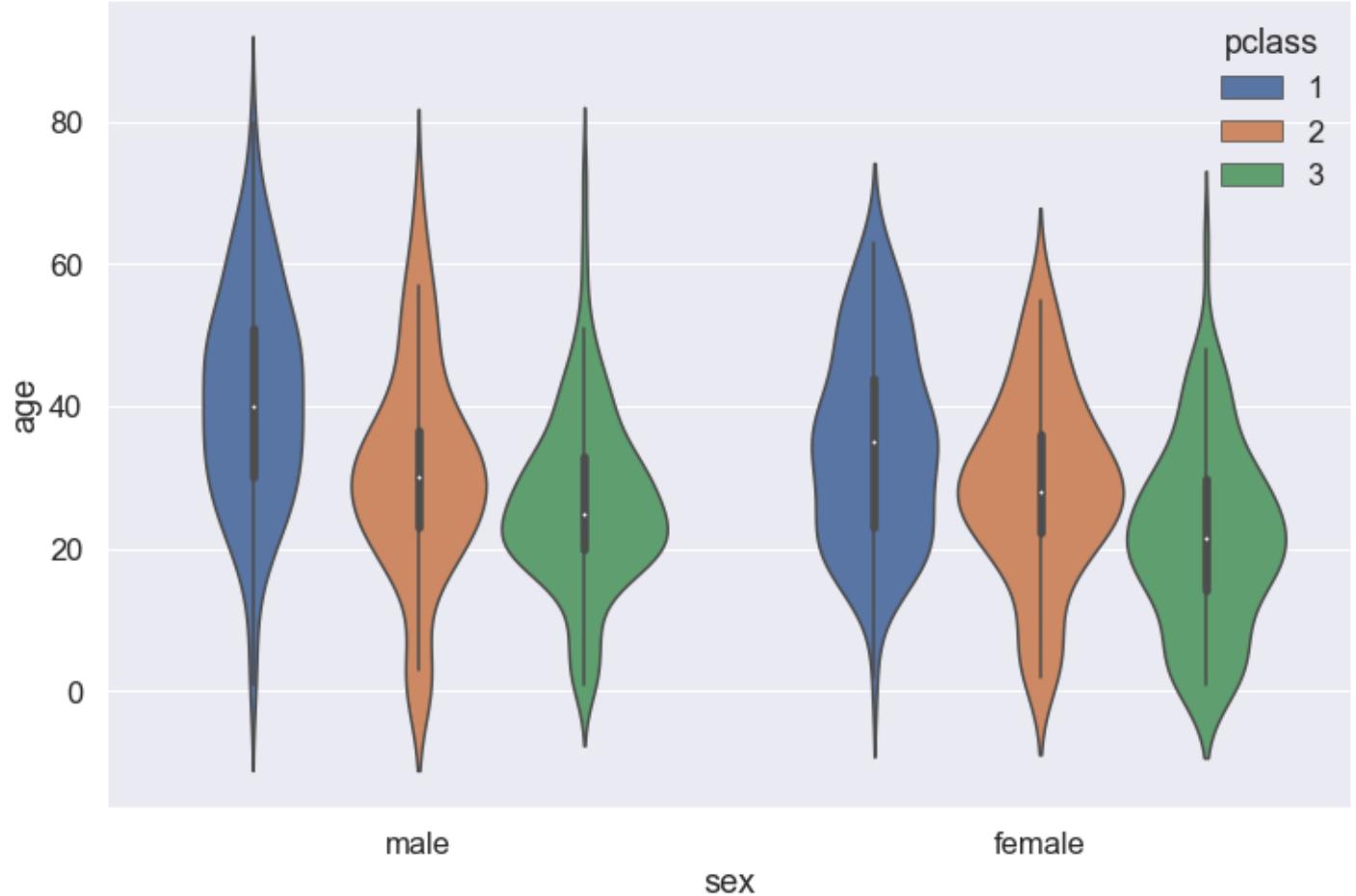
```
In [280...]: ## By using a hue chart we can see more females survived than died. The reverse is true.
```

```
In [281...]: # Sees who survived.  
plt.figure(figsize =(12,8))  
sns.set(font_scale =1.5)  
sns.swarmplot(data = titanic, x = "sex", y = "age", hue = "survived", dodge = True)  
plt.show()
```



sns.violinplot() shows the distribution of data well.

```
In [285]:  
plt.figure(figsize = (12,8))  
sns.set(font_scale = 1.5)  
sns.violinplot(data = titanic, x = "sex", y = "age", hue = "pclass", dodge = True)  
plt.show()
```



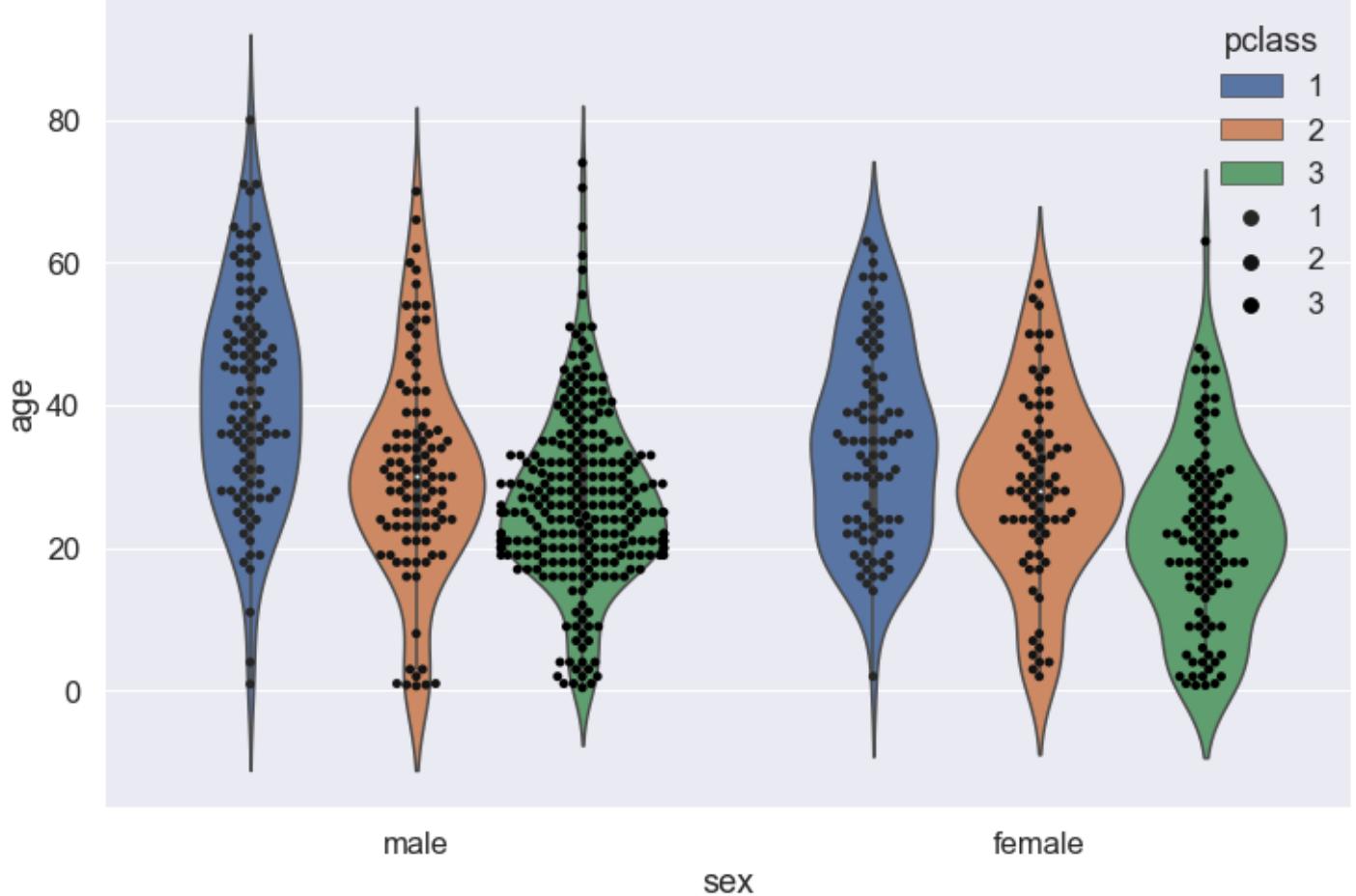
```
In [287]:
```

```
## Dot plots and violin plots can be used in tandem see here.
```

```
In [288]:
```

```
plt.figure(figsize = (12,8))
sns.set(font_scale = 1.5)
sns.violinplot(data = titanic, x = "sex", y = "age", hue = "pclass", dodge = True)
sns.swarmplot(data = titanic, x = "sex", y = "age", hue = "pclass", dodge = True, color = "black")
plt.show()

C:\Users\alonz\anaconda3\lib\site-packages\seaborn\categorical.py:1296: UserWarning: 5.8%
of the points cannot be placed; you may want to decrease the size of the markers or use st
ripplot.
warnings.warn(msg, UserWarning)
```



We can switch a variable into a hue for different results. This is a plain violin.

These violins do not include any of the sns.swarmplot that was used previously.

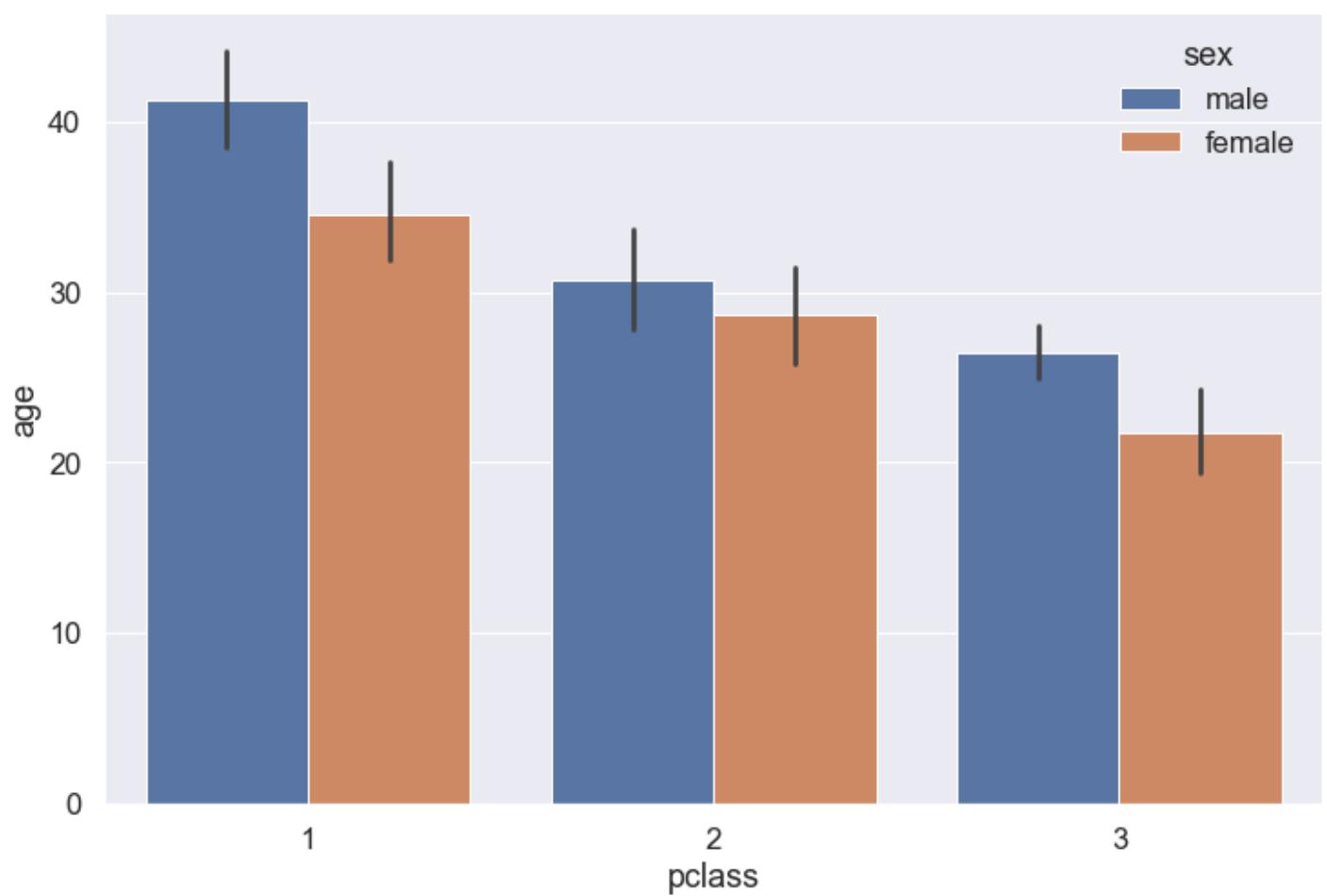
In []:

```
plt.figure(figsize = (12,8))
sns.set(font_scale = 1.5)
sns.violinplot(data = titanic, x = "pclass", y = "age", hue = "sex", dodge = True)
plt.show()
```

Box and whiskers are called sns.barplot() in Seaborn.

In [262...]

```
plt.figure(figsize = (12,8))
sns.set(font_scale = 1.5)
sns.barplot(data = titanic, x= "pclass", y = "age", hue = "sex", dodge = True)
plt.show()
```



In [292]:

```
## sns.barplots can utilize confidence intervals. Shift + tab can give us the default val
```

Signature:

```
sns.barplot(  
    *,  
    x=None,  
    y=None,  
    hue=None,  
    data=None,  
    order=None,  
    hue_order=None,  
    estimator=<function mean at 0x000001C4AED83AF0>,  
    ci=95,  
    n_boot=1000,  
    units=None,  
    seed=None,  
    orient=None,  
    color=None,  
    palette=None,  
    saturation=0.75,  
    errcolor='.26',  
    errwidth=None,  
    caps=None,  
    dodge=True,  
    ax=None,  
    **kwargs,  
)
```

The pointplot can show who tended to be older with statistical significance it was the males.

The relationship between age and pclass, colored by gender.

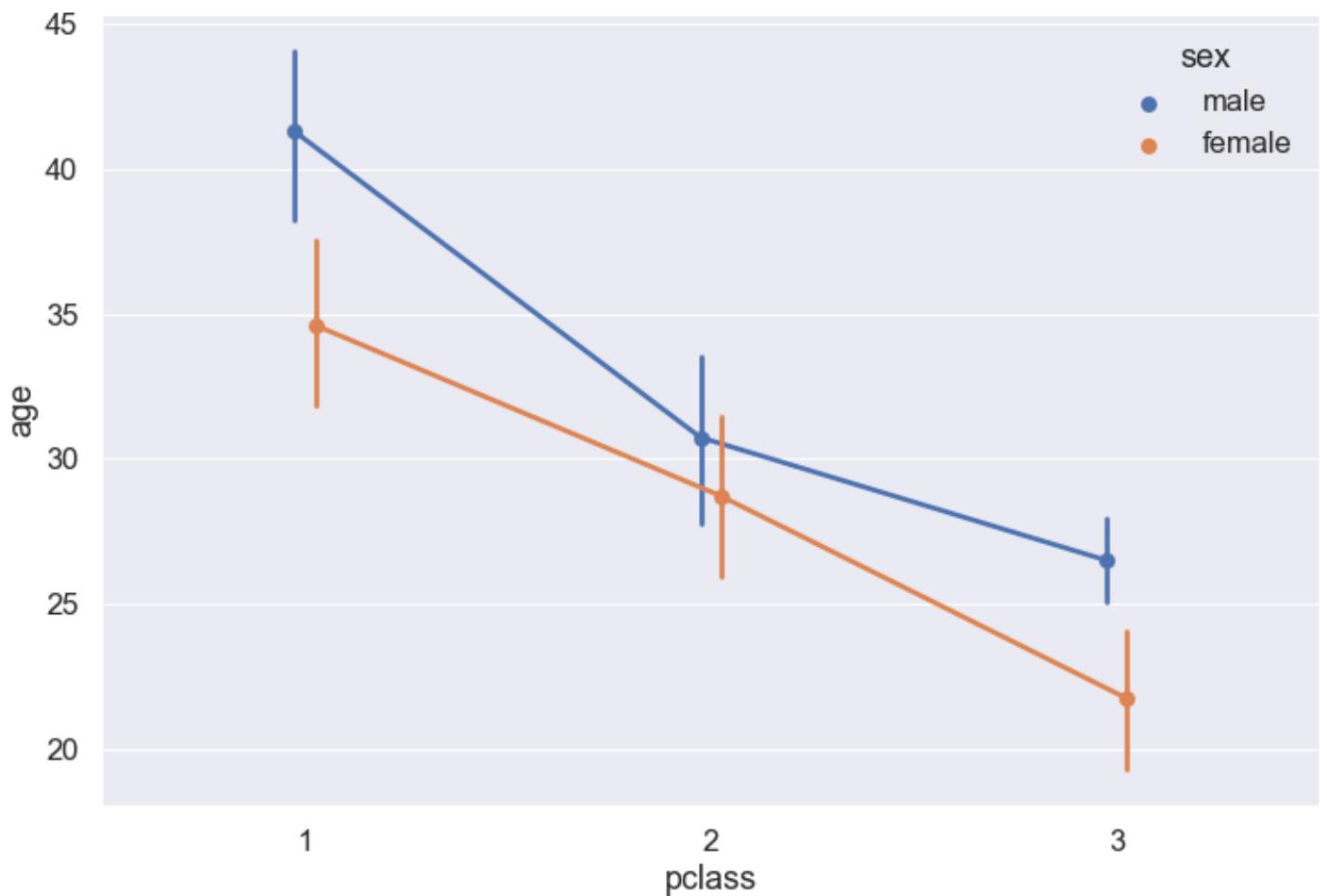
It uses confidence intervals and simulations.

In [297...]

```
plt.figure(figsize = (12,8))  
sns.set(font_scale = 1.5)  
sns.pointplot(data = titanic, x = "pclass", y = "age", hue = "sex", dodge = True )
```

Out[297...]

```
<AxesSubplot: xlabel='pclass', ylabel='age'>
```



Seaborn Regression Plots

Jointplots and Regression

In [301...]

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [302...]

```
titanic= pd.read_csv(r"C:\Users\alonz\Downloads\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

In [303...]

```
titanic.head()
```

Out[303...]

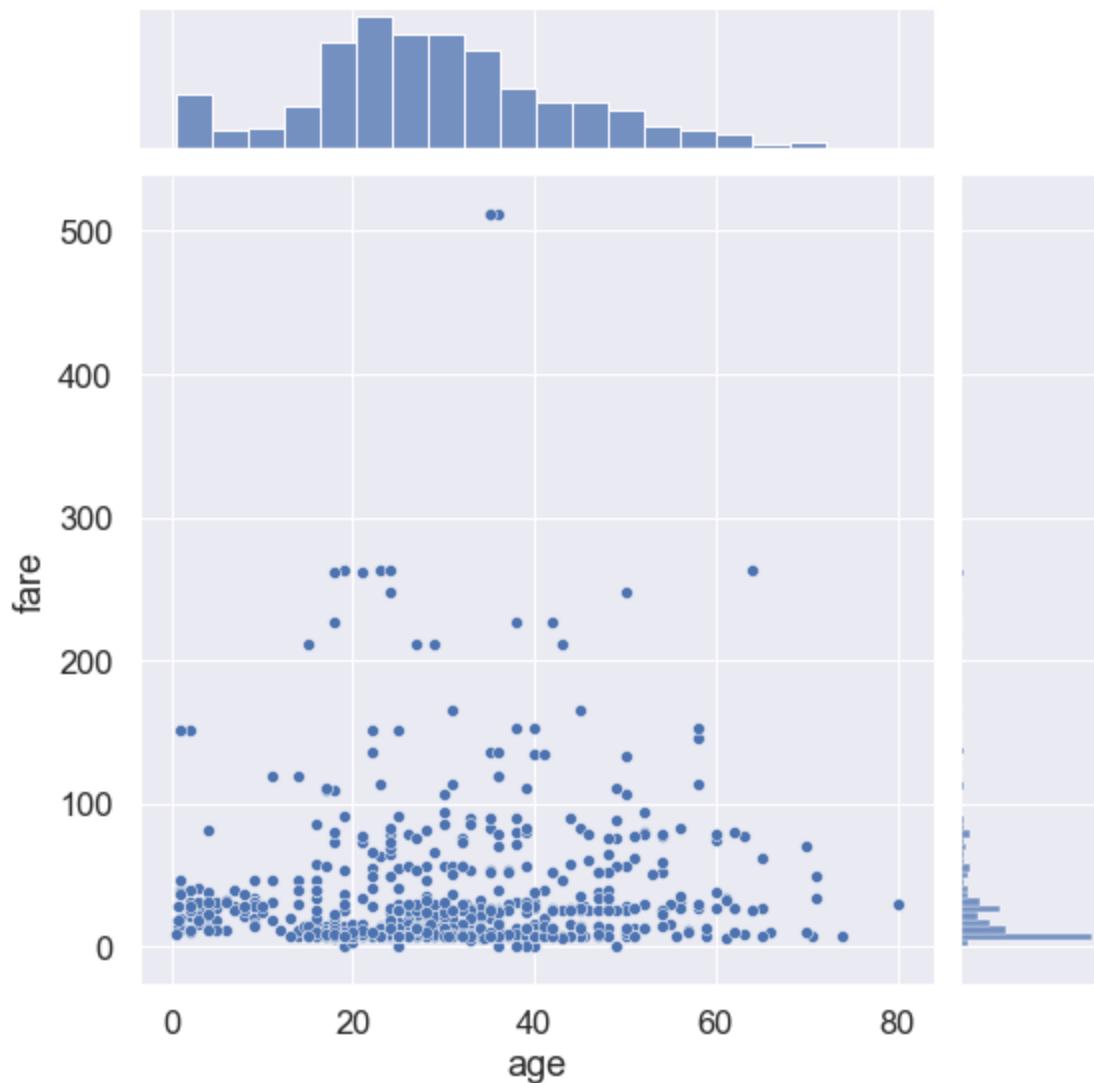
	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN

Jointplot creates a histogram for the X and Y value. In addition to this it creates a scatterplot if kind = "scatter"

It can show a causal relationship between variables.

In [313...]

```
sns.set(font_scale = 1.5)
sns.jointplot(data = titanic, x = "age", y = "fare", height = 8, kind = "scatter")
plt.show()
```

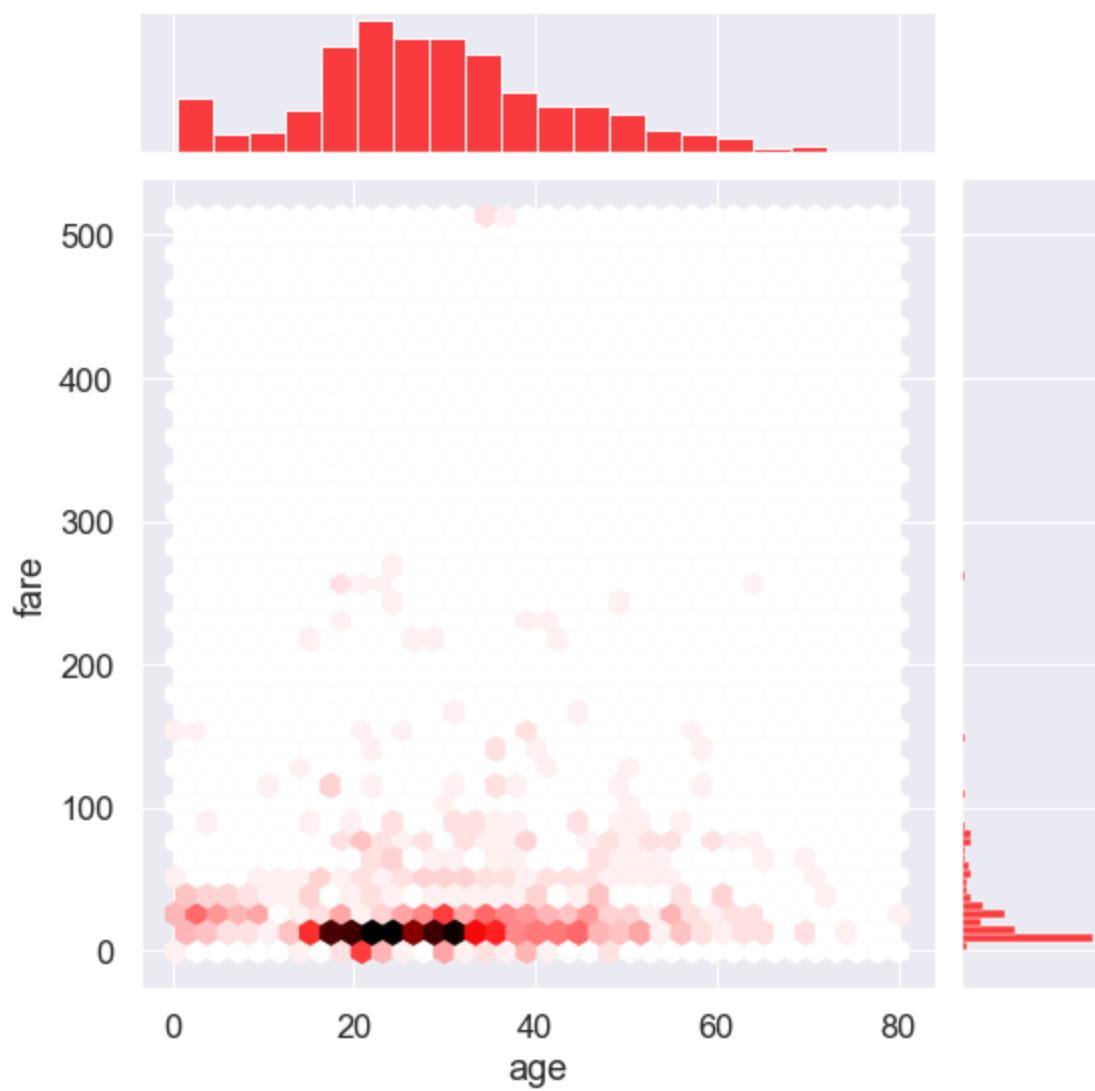


Changing the kind to hex makes the color darker at concentrations of data.

kind = "hex"

In [318...]

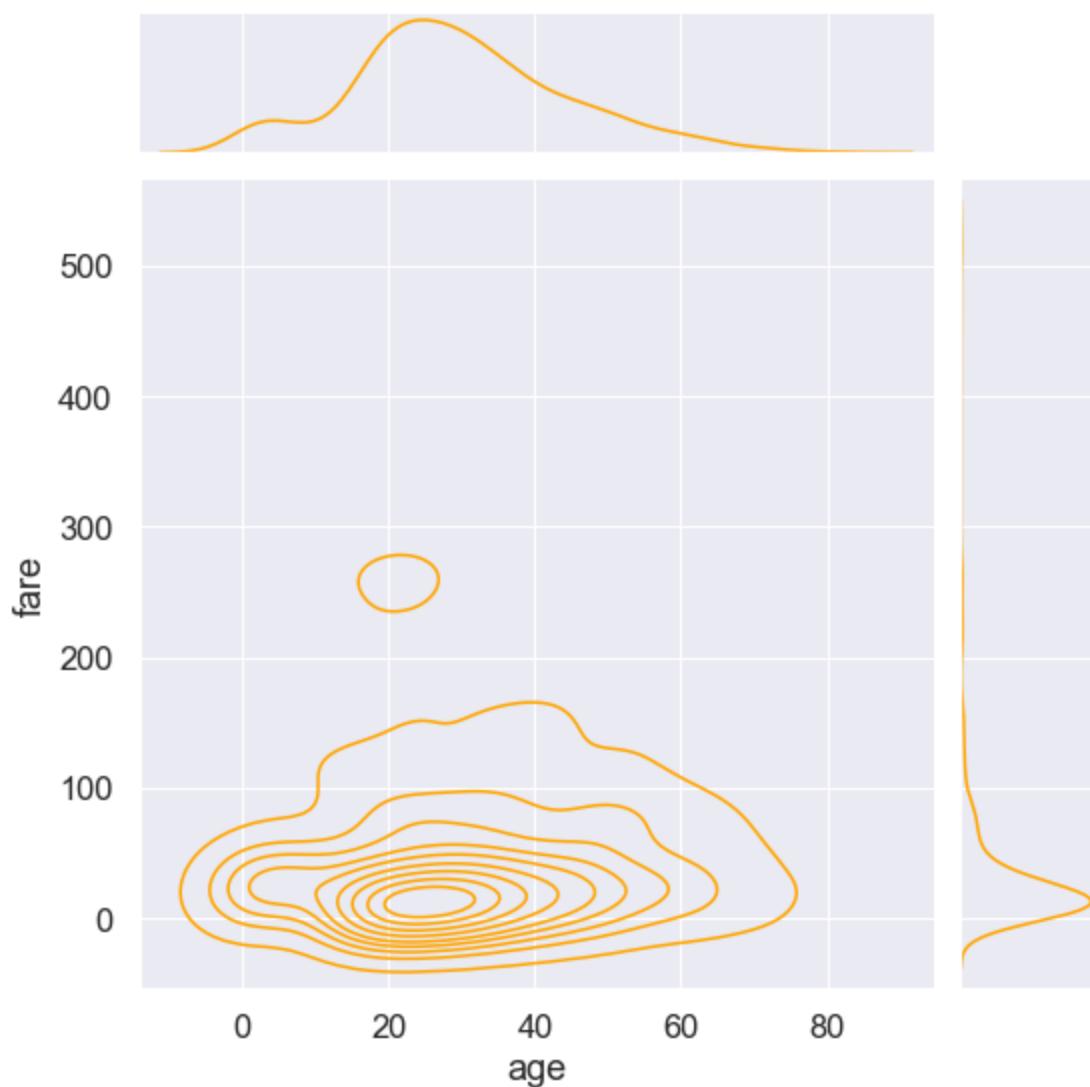
```
sns.set(font_scale = 1.5)
sns.jointplot(data = titanic, x = "age", y = "fare", height = 8, kind = "hex", color = "red")
plt.show()
```



Can use kind = "kde" to get a density map of the jointplot.

In [322]:

```
sns.set(font_scale = 1.5)
sns.jointplot(data = titanic, x = "age", y = "fare", height = 8, kind = "kde", color = "orange")
plt.show()
```



Regression line for sns.jointplot() This is very important.

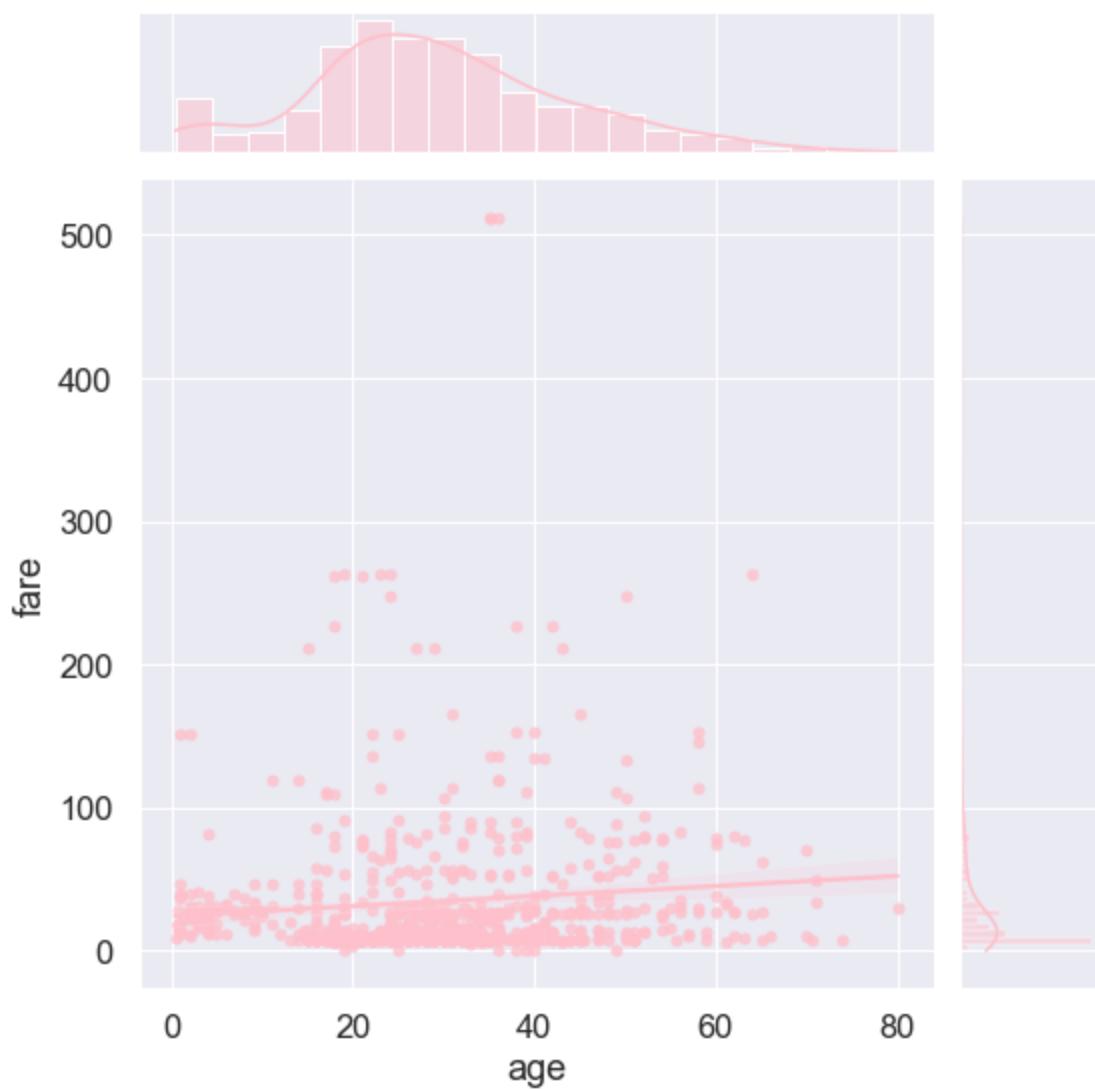
Just change the kind = "reg"

In [326...]

```

sns.set(font_scale = 1.5)
sns.jointplot(data = titanic, x = "age", y = "fare", height = 8, kind = "reg", color = "pink")
plt.show()
# The regression line shows a positive correlation between fare and age.
# The 95% confidence interval tells that out of a thousand simulations the best fit line
# 950 of them.

```



sns.lmplot() is able to get a line of best fit without having to modify so many settings.

Remember to press shift + Tab + Tab (Double tap tab) to see all of the settings.

Confidence interval CI shows that we are 95% confidence that the true relationship or correlation will fall in between certain ranges.

In [324]:

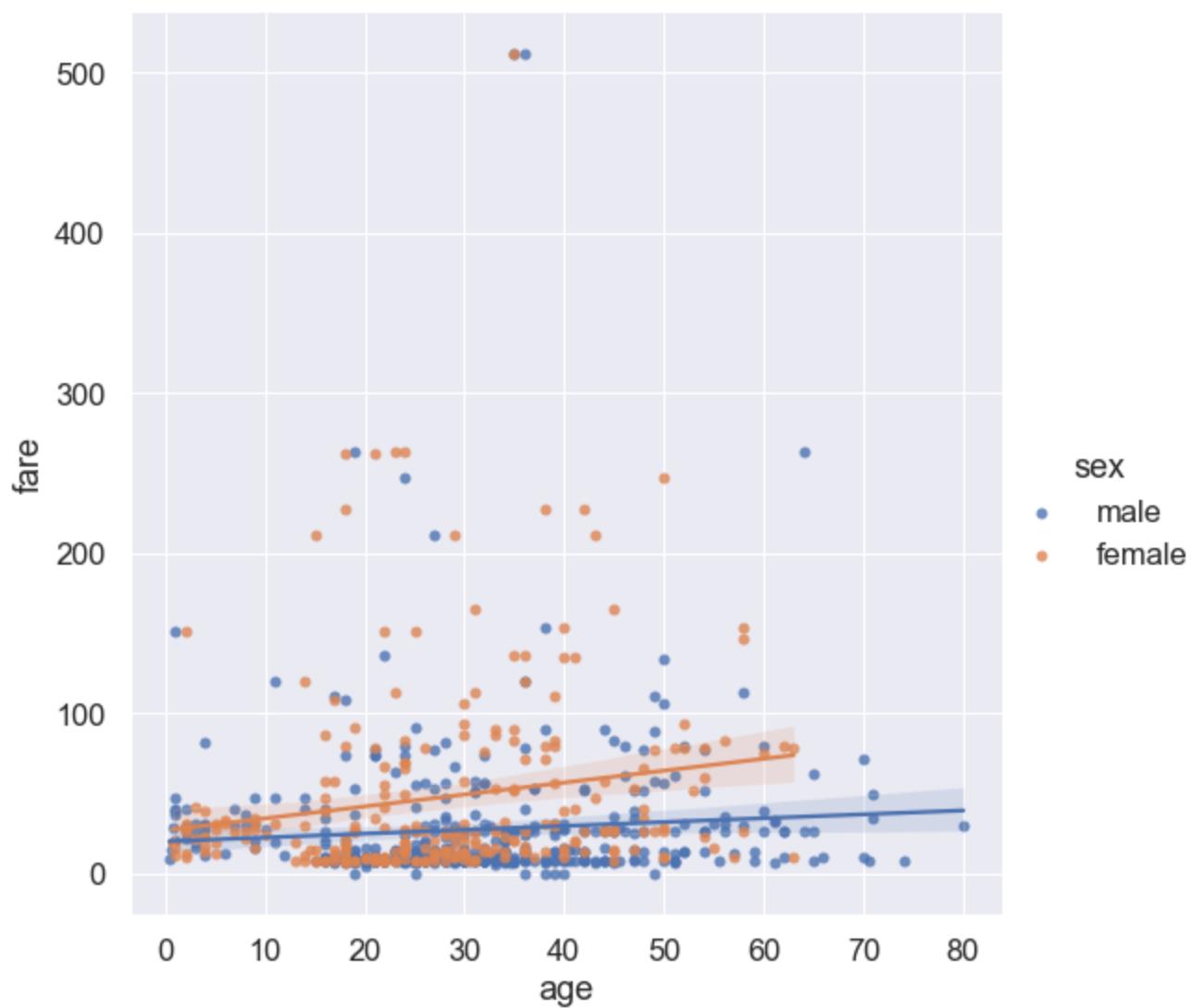
```
sns.set(font_scale = 1.5)
sns.lmplot(data = titanic, x = "age", y = "fare", aspect = 1, height = 8, hue = None)
plt.show()
```



Put an ordinal characteristic into hue for the chart and you will have two lines of best fit for the graph.

In [328...]

```
# Can split up by sex. Just change the hue to sex.  
sns.set(font_scale = 1.5)  
sns.lmplot(data = titanic, x = "age", y = "fare", aspect = 1, height = 8, hue = "sex")  
plt.show()
```

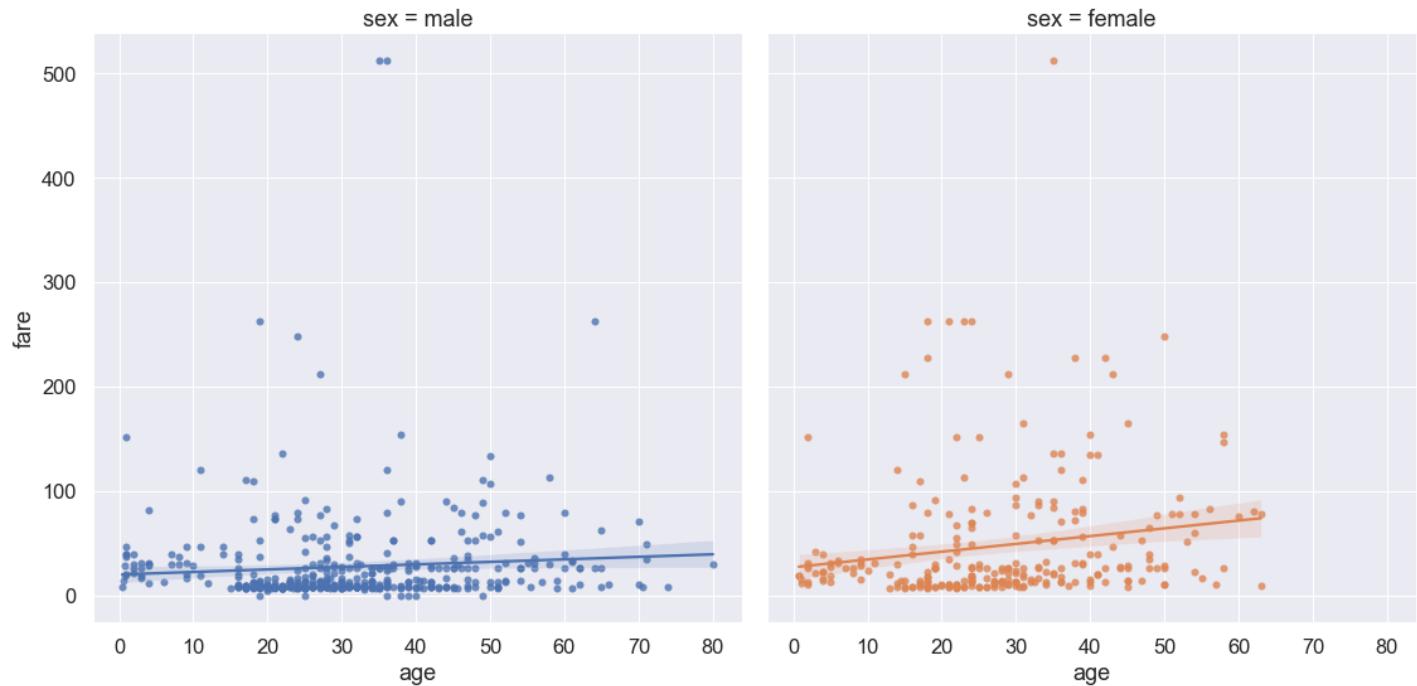


Can make two separate graphs by setting the col = "sex"

This way we can make two separate graphs, instead of separating by color. You can even make the graphs two color in order to make the two in one chart more distinctive.

In [331...]

```
# We can add col to separate the graph into two graphs.
sns.set(font_scale = 1.5)
sns.lmplot(data = titanic, x = "age", y = "fare", aspect = 1, height = 8, hue = "sex",
```



For sns.lmplot() logistic = True when dealing with a categorical variables.

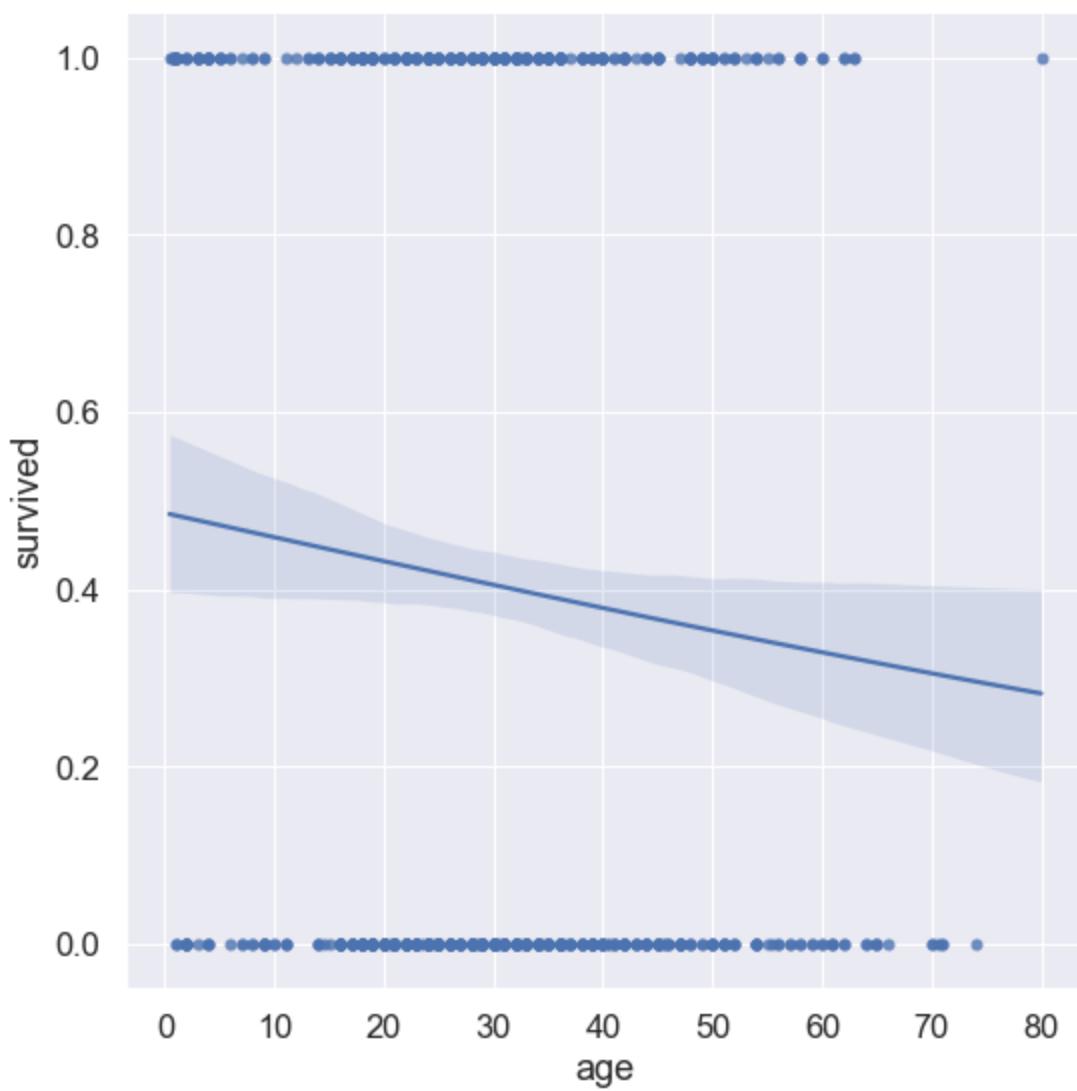
Categorical variables is when zero or one represents two different outcomes.

Note that the shaded area around the line of best fit is the confidence interval. When it is too big it could mean that it is statistically insignificant.

Not being statistically significant means that we can ignore the conclusions that can be drawn from the data, or at least it is less authoritative.

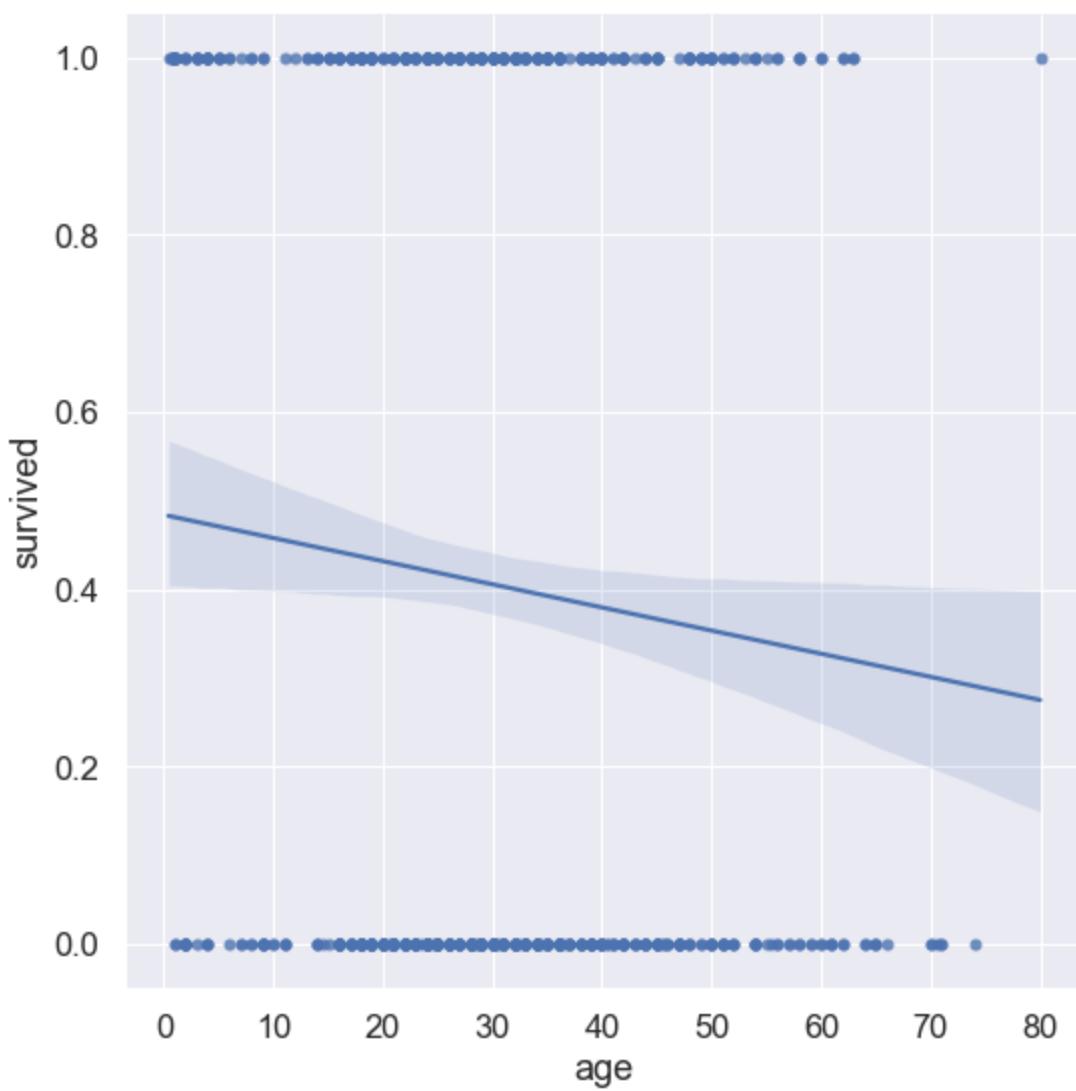
In [335...]

```
# The data shows that the younger you are the more likely you are to survive.
sns.set(font_scale = 1.5)
sns.lmplot(data = titanic, x = "age", y = "survived", aspect = 1, height = 8, col = None,
plt.show()
```



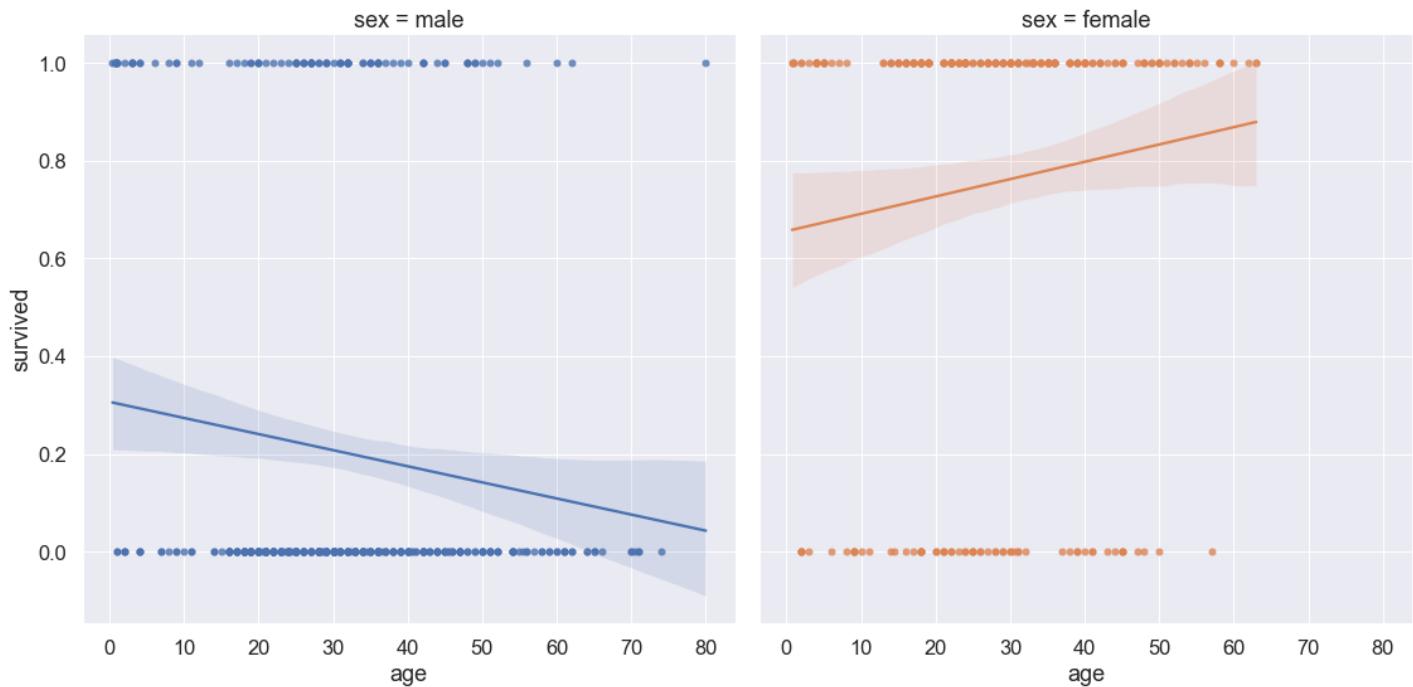
In [312]:

```
sns.set(font_scale = 1.5)
sns.lmplot(data = titanic, x = "age", y = "survived", aspect = 1, height = 8, col = None,
plt.show()
```



In [360]:

```
## The younger the male the more likely he is to survive.
## Women tend to survivor more the older that they get.
sns.set(font_scale = 1.5)
sns.lmplot(data = titanic, x = "age", y = "survived", aspect = 1, height = 8, col = "sex",
plt.show()
```



Pclass split

Note logistic Regression

Logistic regression assumes the dependent variable Y is binary.
Either yes or no.

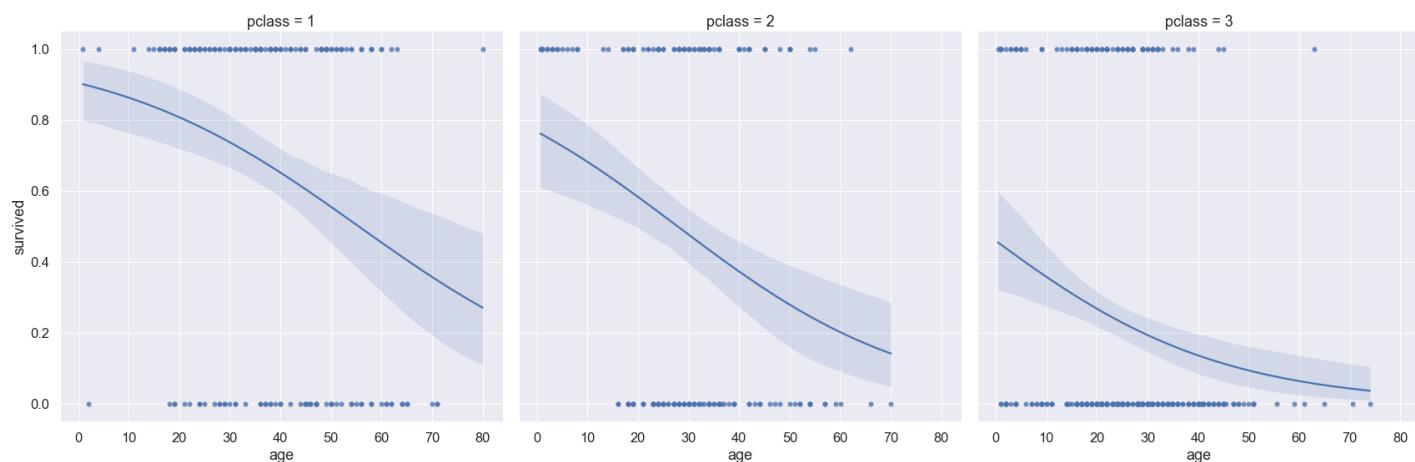
Do more research:

In [367...]

```
# https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-be
# P(Y = 1) as a function of x
# Review more.
```

In [365...]

```
sns.set(font_scale = 1.5)
sns.lmplot(data = titanic, x = "age", y = "survived", aspect = 1, height = 8, col = "pclass")
plt.show()
```



Matrixplots and Heatmaps

In [370...]

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [371...]

```
titanic = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

In [372...]

```
titanic.head()
```

Out[372...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN

pd.crosstab is when you see how many people fit into category, while looking at two facets.

Gets you the count of how many fit into two set of ordinal values.

```
In [373...]: pd.crosstab(titanic.sex, titanic.pclass)
```

```
Out[373...]:
```

	pclass	1	2	3
sex				
female	94	76	144	
male	122	108	347	

Order does matter for pd.crosstab

```
In [377...]: pd.crosstab(titanic.pclass, titanic.sex)
```

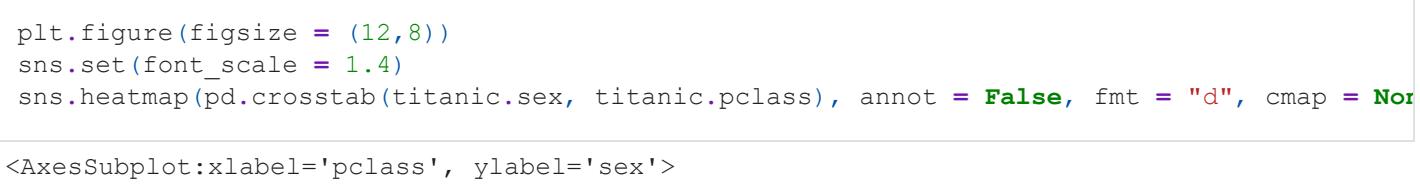
```
Out[377...]:
```

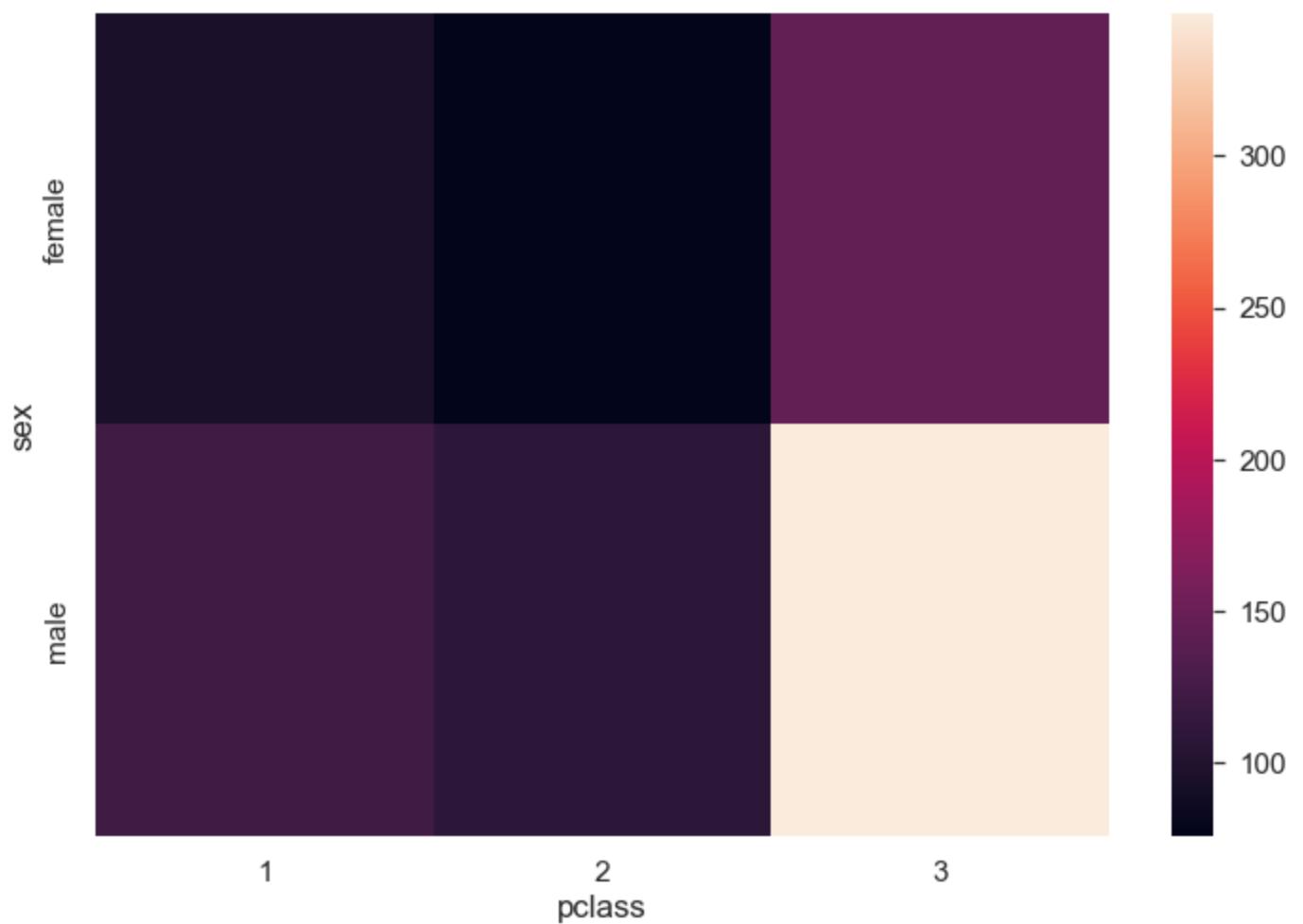
sex	female	male
pclass		
1	94	122
2	76	108
3	144	347

```
In [381...]:
```

```
plt.figure(figsize = (12,8))
sns.set(font_scale = 1.4)
sns.heatmap(pd.crosstab(titanic.sex, titanic.pclass), annot = False, fmt = "d", cmap = "Oranges")
```

```
Out[381...]:
```





Make the `annot = True` in order to see the values displayed on the rectangles.

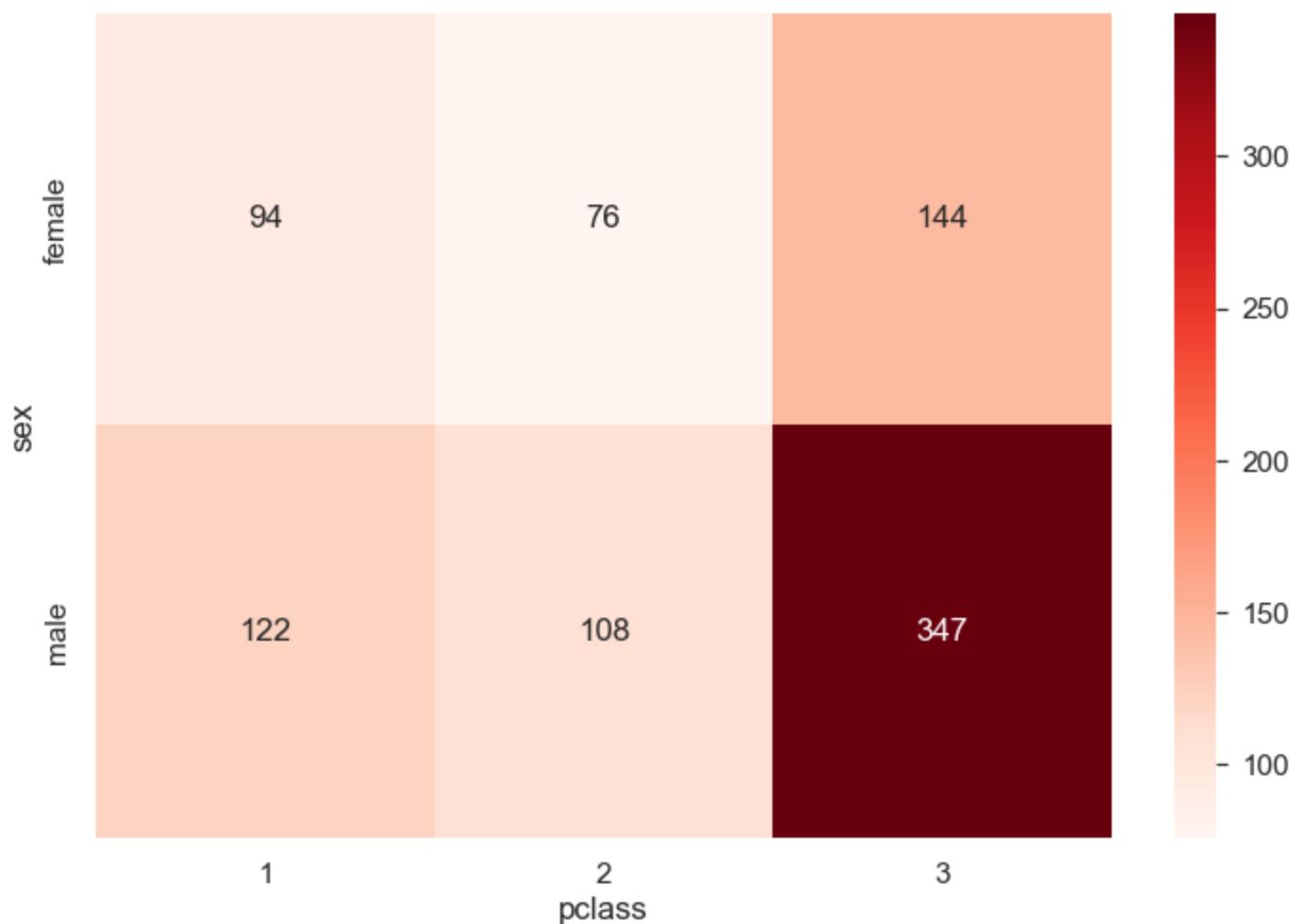
Use `cmap` in order to change the color scheme of the `sns.heatmap()`

In [386...]

```
plt.figure(figsize = (12,8))
sns.set(font_scale = 1.4)
sns.heatmap(pd.crosstab(titanic.sex, titanic.pclass), annot = True, fmt = "d", cmap = "Reds")
```

Out[386...]

```
<AxesSubplot:xlabel='pclass', ylabel='sex'>
```



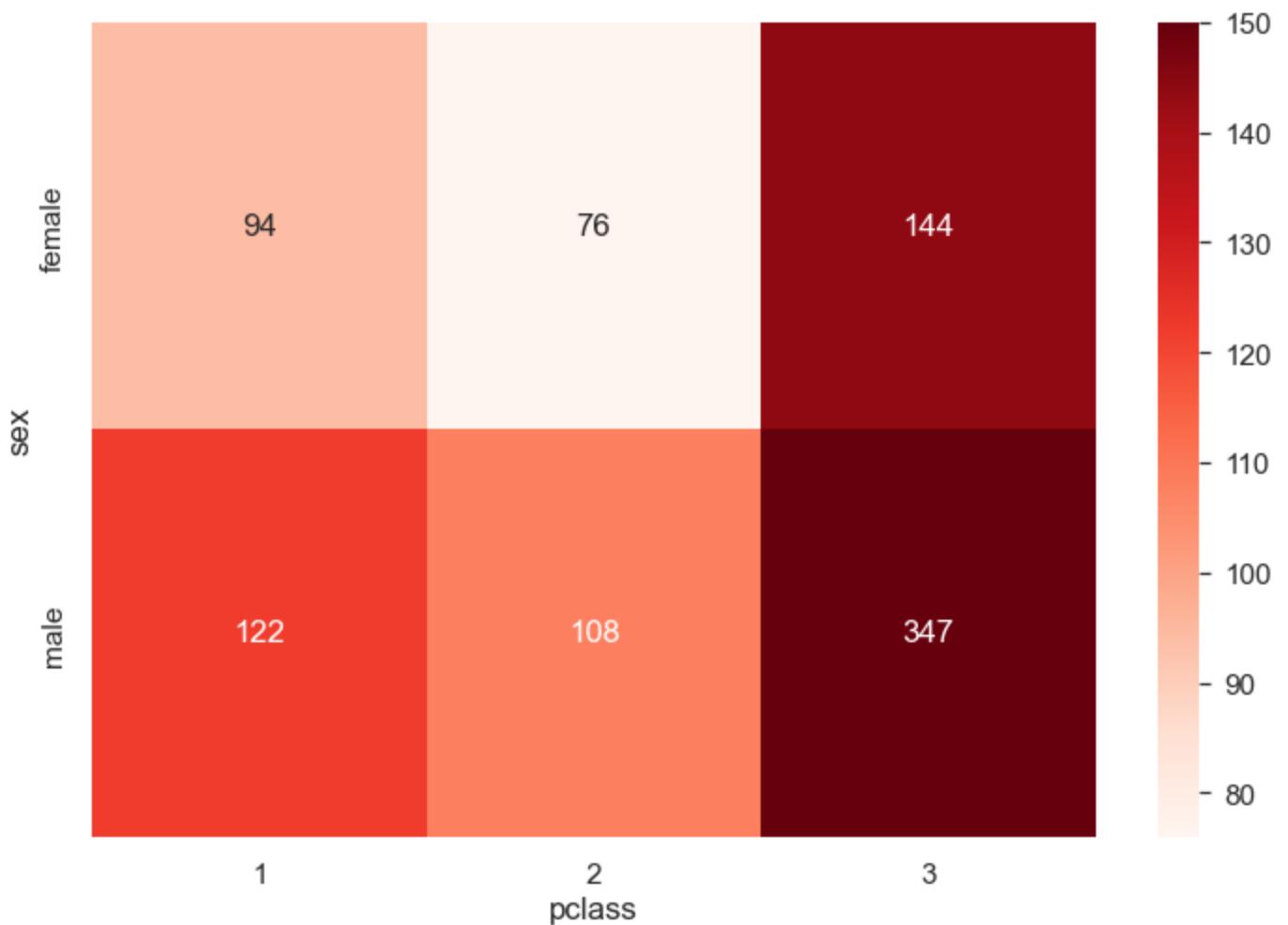
Use vmap to determine the maximum darkness intensity. This can be used to stop an outlier from skewing the color scheme.

In [388...]

```
plt.figure(figsize = (12,8))
sns.set(font_scale = 1.4)
sns.heatmap(pd.crosstab(titanic.sex, titanic.pclass), annot = True, fmt = "d", cmap = "Red
```

Out[388...]

```
<AxesSubplot:xlabel='pclass', ylabel='sex'>
```



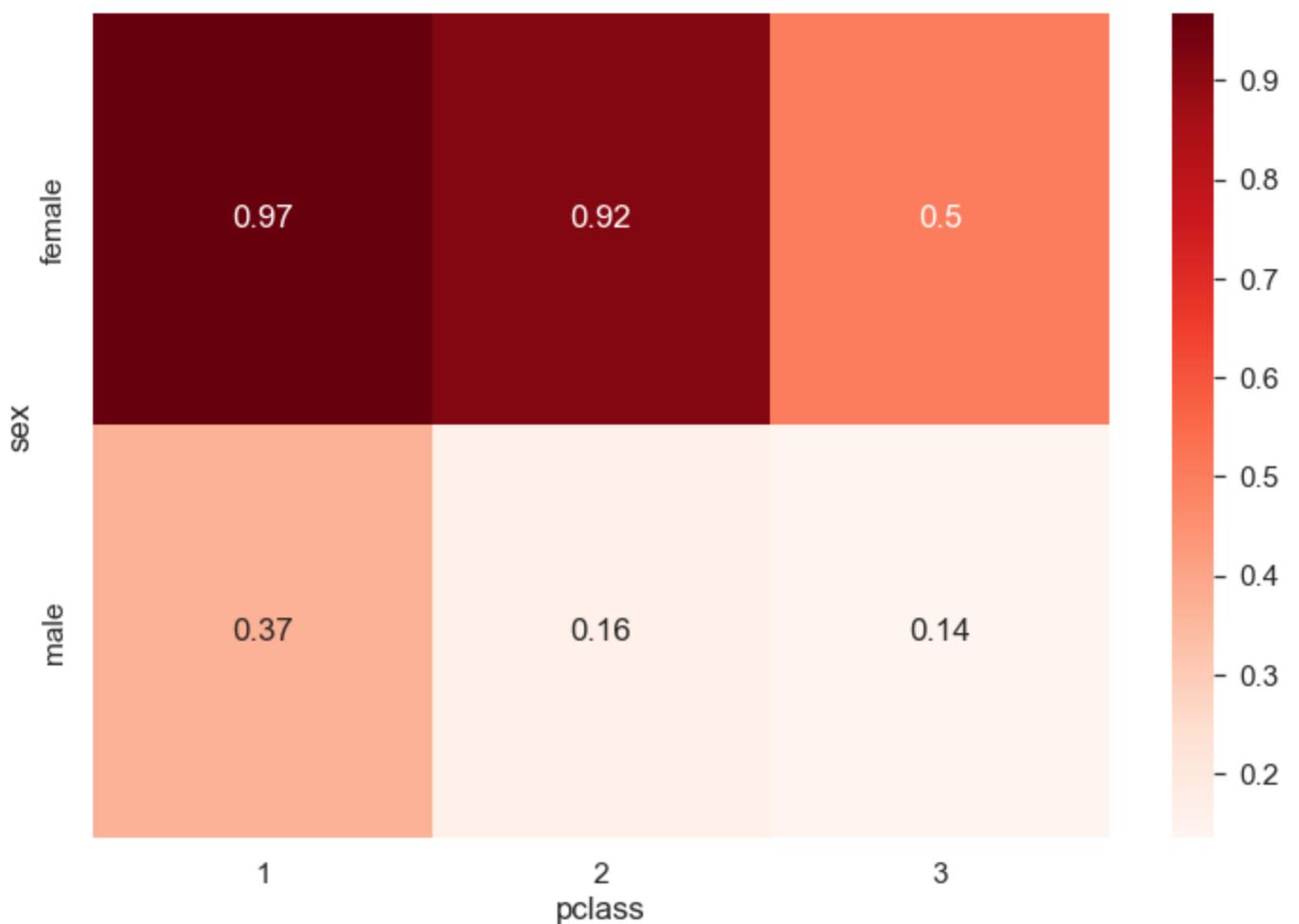
`pd.crosstab(aggfunc = "mean")` can be used to find averages.

```
In [379...]: pd.crosstab(titanic.sex, titanic.pclass, values = titanic.survived, aggfunc = "mean")
```

```
Out[379...]:
```

sex	1	2	3
female	0.968085	0.921053	0.500000
male	0.368852	0.157407	0.135447

```
In [390...]: # This map shows how many people survived using colors.
plt.figure(figsize = (12,8))
sns.set(font_scale = 1.4)
sns.heatmap(pd.crosstab(titanic.sex, titanic.pclass, values = titanic.survived, aggfunc =
```



```
In [391]: titanic.corr()
```

```
Out[391]:
```

	survived	pclass	age	sibsp	parch	fare
survived	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307
pclass	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500
age	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067
sibsp	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651
parch	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225
fare	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000

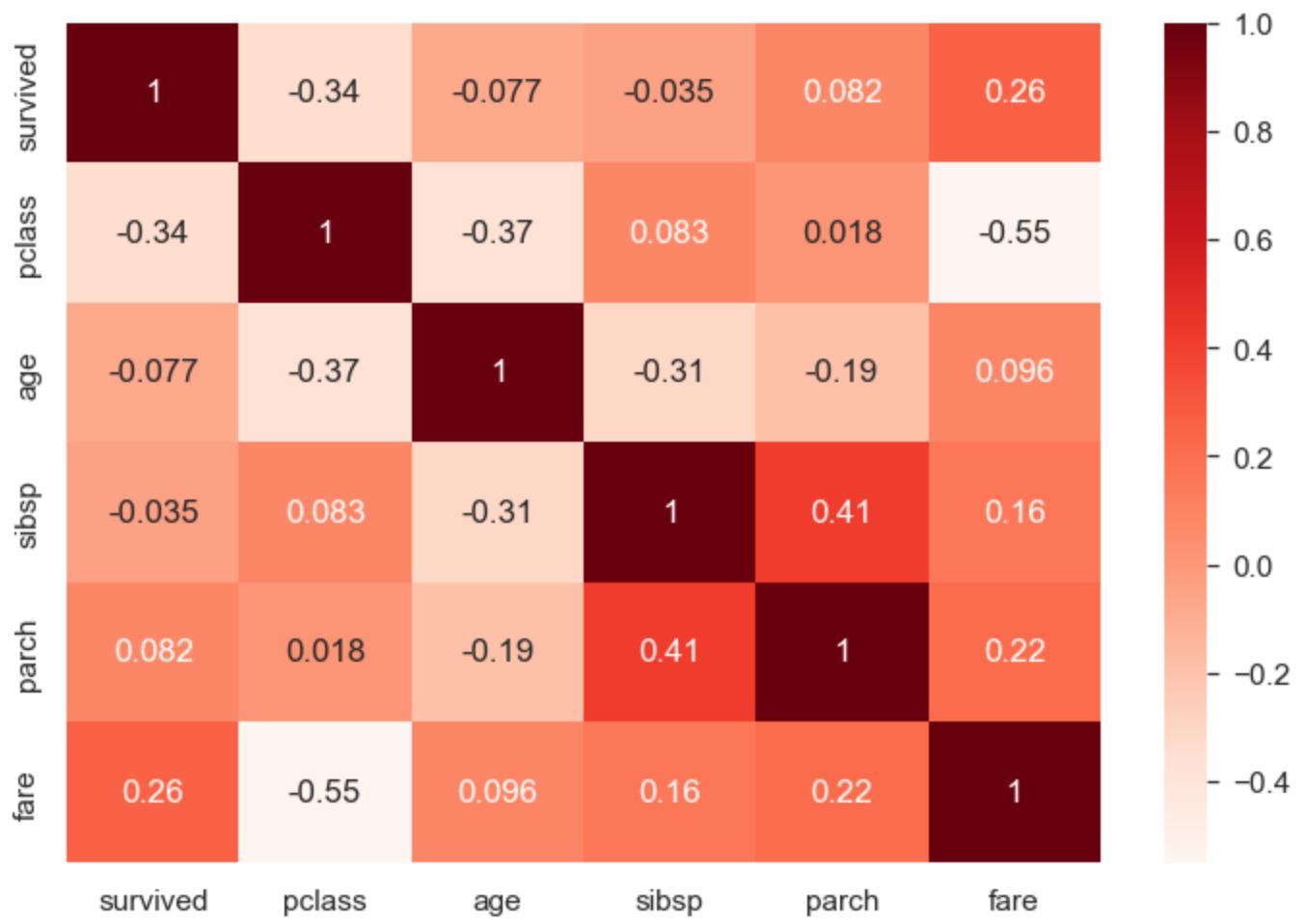
Excellent Feature

Combine sns.heatmap() with data_set.corr()

You can use `sns.heatmap(data_set.corr(), annot = True, cmap = "Reds")` to create a heatmap to see how each characteristic correlates with each other.

```
In [392]:
```

```
## You can pass titanic.corr() to a heatmap.
plt.figure(figsize = (12, 8))
sns.set(font_scale = 1.4)
sns.heatmap(titanic.corr(), annot = True, cmap = "Reds")
plt.show()
```



In []:

In []:

In []:

In []:

Time Series Basics

Importing Time Series Data from csv-Files

In [4]:

```
import pandas as pd
```

parse_dates is when a column is interpreted as a time and date.

YYYY-MM-DD HR:MN:SC

The hierarchy is shown above. It must be year, month, day, hour, minute, and seconds respectively.

Note that there is a space between the date and time. The date is separated by "-" and time by ":".

In [33]:

```
# Press Shift + Tab(3x) to see documentation
# Parse_dates is hard to find without knowing the even after pressing tab.
# It is important to be able to do research to know how to import certain data types.
temp = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\temp.csv")
```

Temp is for temperatures in degrees celsius.

In [34]:

```
temp.head()
```

Out[34]:

LA NY

	datetime	LA	NY
2013-01-01 00:00:00	11.7	-1.1	
2013-01-01 01:00:00	10.7	-1.7	
2013-01-01 02:00:00	9.9	-2.0	
2013-01-01 03:00:00	9.3	-2.1	
2013-01-01 04:00:00	8.8	-2.3	

In [35]:

```
temp.tail()
```

Out[35]:

LA NY

	datetime	LA	NY
2016-12-31 19:00:00	13.5	4.6	
2016-12-31 20:00:00	13.2	5.7	
2016-12-31 21:00:00	12.8	5.8	
2016-12-31 22:00:00	12.3	5.7	
2016-12-31 23:00:00	11.9	5.5	

In [36]:

```
# When the index is the datetime the range of dates can easily be seen.
```

```
temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 35064 entries, 2013-01-01 00:00:00 to 2016-12-31 23:00:00
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  --  
 0   LA      35062 non-null   float64 
 1   NY      35064 non-null   float64 
dtypes: float64(2)
memory usage: 821.8 KB
```

```
In [37]: temp.iloc[0, 0]
```

```
Out[37]: 11.7
```

```
In [41]: temp.index
```

```
DatetimeIndex(['2013-01-01 00:00:00', '2013-01-01 01:00:00',
                 '2013-01-01 02:00:00', '2013-01-01 03:00:00',
                 '2013-01-01 04:00:00', '2013-01-01 05:00:00',
                 '2013-01-01 06:00:00', '2013-01-01 07:00:00',
                 '2013-01-01 08:00:00', '2013-01-01 09:00:00',
                 ...
                 '2016-12-31 14:00:00', '2016-12-31 15:00:00',
                 '2016-12-31 16:00:00', '2016-12-31 17:00:00',
                 '2016-12-31 18:00:00', '2016-12-31 19:00:00',
                 '2016-12-31 20:00:00', '2016-12-31 21:00:00',
                 '2016-12-31 22:00:00', '2016-12-31 23:00:00'],
                dtype='datetime64[ns]', name='datetime', length=35064, freq=None)
```

```
In [42]: temp.index[0]
```

```
Out[42]: Timestamp('2013-01-01 00:00:00')
```

Now the datetime is a timestamp can be clearly seen.

```
In [43]: type(temp.iloc[0, 0])
```

```
Out[43]: numpy.float64
```

Convert strings to datetime objects with pd.to_datetime()

Pandas has a method that can convert the object to a datetime object as seen earlier.

Remember the time format is:

YYYY-MM-DD HR:MN:SC

```
In [46]: date = "2000-01-31 00:10:09"
```

```
In [47]: type(date)
```

```
Out[47]: str
```

```
In [49]: # Convert the date into a timestamp.  
date = pd.to_datetime(date)  
date
```

```
Out[49]: Timestamp('2000-01-31 00:10:09')
```

```
In [51]: # The date is now a pandas Timestamp.  
type(date)
```

```
Out[51]: pandas._libs.tslibs.timestamps.Timestamp
```

Reimport temp data without index and parse_dates

```
In [52]: temp = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\temp.csv")
```

```
In [53]: temp.head()
```

```
Out[53]:
```

	datetime	LA	NY
0	2013-01-01 00:00:00	11.7	-1.1
1	2013-01-01 01:00:00	10.7	-1.7
2	2013-01-01 02:00:00	9.9	-2.0
3	2013-01-01 03:00:00	9.3	-2.1
4	2013-01-01 04:00:00	8.8	-2.3

```
In [55]: # By default the datetime is a regular object.  
temp.datetime
```

```
Out[55]:
```

0	2013-01-01 00:00:00
1	2013-01-01 01:00:00
2	2013-01-01 02:00:00
3	2013-01-01 03:00:00
4	2013-01-01 04:00:00
	...
35059	2016-12-31 19:00:00
35060	2016-12-31 20:00:00
35061	2016-12-31 21:00:00
35062	2016-12-31 22:00:00
35063	2016-12-31 23:00:00

Name: datetime, Length: 35064, dtype: object

```
In [62]: # Press tab after typing pd.to_ to see all of the conversion options  
# The original attribute/column needs to be overwritten as seen below.  
# The cell above showed an object and this cell shows a datetime object.  
temp.datetime = pd.to_datetime(temp.datetime)  
temp.datetime
```

```
Out[62]:
```

0	2013-01-01 00:00:00
1	2013-01-01 01:00:00
2	2013-01-01 02:00:00
3	2013-01-01 03:00:00
4	2013-01-01 04:00:00
	...
35059	2016-12-31 19:00:00

```
35060 2016-12-31 20:00:00
35061 2016-12-31 21:00:00
35062 2016-12-31 22:00:00
35063 2016-12-31 23:00:00
Name: datetime, Length: 35064, dtype: datetime64[ns]
```

Remember use `dataset.set_index()` to change the index.

Knowing how to set the index is very important.

```
In [ ]: # How the instructor coded the solution.
# He converted the date column to a datetime object while making it the index.
# After he did that he dropped the duplicate datetime column
# Remember axis 1 pertain to columns.
temp.set_index(pd.to_datetime(temp.datetime)).drop("datetime", axis = 1)
```

The code used in this project.

```
In [ ]: # The datetime has already been converted to a datetime.
# All that has to be done is for datetime to be set as the index.
# To make the change is permanent use inplace = True or use overwrite the data using "="
temp.set_index("datetime", inplace = True)
```

```
In [74]: temp.head()
```

```
Out[74]:          LA    NY
                datetime
2013-01-01 00:00:00  11.7  -1.1
2013-01-01 01:00:00  10.7  -1.7
2013-01-01 02:00:00   9.9  -2.0
2013-01-01 03:00:00   9.3  -2.1
2013-01-01 04:00:00   8.8  -2.3
```

Remember you can use `dataset.reset_index()` to make the index numerical again.

```
In [76]: temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 35064 entries, 2013-01-01 00:00:00 to 2016-12-31 23:00:00
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  --   --   --   --   --   --   -- 
 0   LA      35062 non-null  float64
 1   NY      35064 non-null  float64
dtypes: float64(2)
memory usage: 821.8 KB
```

```
In [77]: temp.index
```

```
Out[77]: DatetimeIndex(['2013-01-01 00:00:00', '2013-01-01 01:00:00',
 '2013-01-01 02:00:00', '2013-01-01 03:00:00',
```

```
'2013-01-01 04:00:00', '2013-01-01 05:00:00',
'2013-01-01 06:00:00', '2013-01-01 07:00:00',
'2013-01-01 08:00:00', '2013-01-01 09:00:00',
...
'2016-12-31 14:00:00', '2016-12-31 15:00:00',
'2016-12-31 16:00:00', '2016-12-31 17:00:00',
'2016-12-31 18:00:00', '2016-12-31 19:00:00',
'2016-12-31 20:00:00', '2016-12-31 21:00:00',
'2016-12-31 22:00:00', '2016-12-31 23:00:00'],
dtype='datetime64[ns]', name='datetime', length=35064, freq=None)
```

Indexes can be isolated using the:

```
dataset.index[index_number]
```

```
In [79]: temp.index[0]
```

```
Out[79]: Timestamp('2013-01-01 00:00:00')
```

Datetime can be used to convert dates without time.

Remember convert using: pd.to_datetime

```
In [84]: # The default time is midnight without a time specified.
pd.to_datetime("2012-01-24")
```

```
Out[84]: Timestamp('2012-01-24 00:00:00')
```

```
In [85]: # Datetime can also convert the date with only the hours without minutes nor seconds.
pd.to_datetime("2012-01-24 10")
```

```
Out[85]: Timestamp('2012-01-24 10:00:00')
```

```
In [86]: # Strings without spaces nor dashes work also, only numbers.
pd.to_datetime("20120124")
```

```
Out[86]: Timestamp('2012-01-24 00:00:00')
```

Year-month-date is the only acceptable formatting for the date.

YYYY-MM-DD

Any other numerical format will throw an error if MM > 12 or it will give you the wrong date. Dates are expressed in descending order (largest amount of time expressed to the smallest).

Month can be typed in with letters instead of being expressed numerically.

```
In [90]: pd.to_datetime("2015 May 20")
```

```
Out[90]: Timestamp('2015-05-20 00:00:00')
```

```
In [98]:
```

```
# Typing out the month prevents the code from throwing an error. Regardless of when it is
# It does not matter whether the month is uppercased or lowercased.
pd.to_datetime("2015 20 May")
```

```
Out[98]:
```

```
Timestamp('2015-05-20 00:00:00')
```

```
In [99]:
```

```
# Giving a day a proper suffix will prevent errors also.
# The code below is understood to mean the 3rd of January due the "rd" suffix.
pd.to_datetime("2015 01 3rd")
```

```
Out[99]:
```

```
Timestamp('2015-01-03 00:00:00')
```

```
In [102...]:
```

```
# Several dates within a list can be converted into a datetime simultaneously.
pd.to_datetime(["2015 01 3rd", "2015 20 May"])
```

```
Out[102...]:
```

```
DatetimeIndex(['2015-01-03', '2015-05-20'], dtype='datetime64[ns]', freq=None)
```

```
In [103...]:
```

```
# Certain words can be converted into dates such as "today"
# Not really mentioned in the documentation
pd.to_datetime("today")
```

```
Out[103...]:
```

```
Timestamp('2022-03-23 17:30:03.048524')
```

```
In [106...]:
```

```
# You can see additional info here:
# https://pandas.pydata.org/docs/reference/api/pandas.to\_datetime.html
```

```
In [ ]:
```

```
## Passing in a list into .to_dataframe() that is not a date throws an error.
# If the code was ran because elephant is not a date. It does not match any format.
pd.to_datetime(["2015 01 3rd", "2015 20 May", "elephant"])
```

```
In [109...]:
```

```
## To force an error through use errors = "coerce"
# When it is coerced it will show in the list of times elephant will be shown as NaT or no
pd.to_datetime(["2015 01 3rd", "2015 20 May", "elephant"], errors = "coerce")
```

```
Out[109...]:
```

```
DatetimeIndex(['2015-01-03', '2015-05-20', 'NaT'], dtype='datetime64[ns]', freq=None)
```

New Section

Initial Analysis/ Visual Inspection of Time Series

```
In [110...]:
```

```
temp.head()
```

```
Out[110...]:
```

LA NY

	datetime		
2013-01-01 00:00:00	11.7	-1.1	
2013-01-01 01:00:00	10.7	-1.7	
2013-01-01 02:00:00	9.9	-2.0	

LA NY

datetime

datetime	LA	NY
2013-01-01 03:00:00	9.3	-2.1
2013-01-01 04:00:00	8.8	-2.3

In [111...]

```
temp.tail()
```

Out[111...]

LA NY

datetime

datetime	LA	NY
2016-12-31 19:00:00	13.5	4.6
2016-12-31 20:00:00	13.2	5.7
2016-12-31 21:00:00	12.8	5.8
2016-12-31 22:00:00	12.3	5.7
2016-12-31 23:00:00	11.9	5.5

.info() tells the counts and the datatypes

In [112...]

```
temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 35064 entries, 2013-01-01 00:00:00 to 2016-12-31 23:00:00
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
--- 
 0   LA      35062 non-null   float64 
 1   NY      35064 non-null   float64 
dtypes: float64(2)
memory usage: 821.8 KB
```

Describe gives several statistics concerning the data.

In [113...]

```
temp.describe()
```

Out[113...]

LA NY

	LA	NY
count	35062.000000	35064.000000
mean	17.486016	12.068269
std	6.640666	10.466832
min	-6.600000	-22.400000
25%	12.900000	3.900000
50%	17.200000	12.500000
75%	21.900000	20.600000
max	42.300000	37.100000

Making Time Series Graphs

What is time series?

Times series is a type of data collection that measures something over the course of time. An example would be the value of the S&P 500 index price over time. The amount of movie tickets a theater sells etc.

import matlab to make a time series graph

You can press Tab to get suggestions while importing.

In [120...]

```
import matplotlib.pyplot as plt
```

Within `dataset.plot(layout = (x_row, y_columns))` is very important.

It allows you to dictate how the graph looks like.

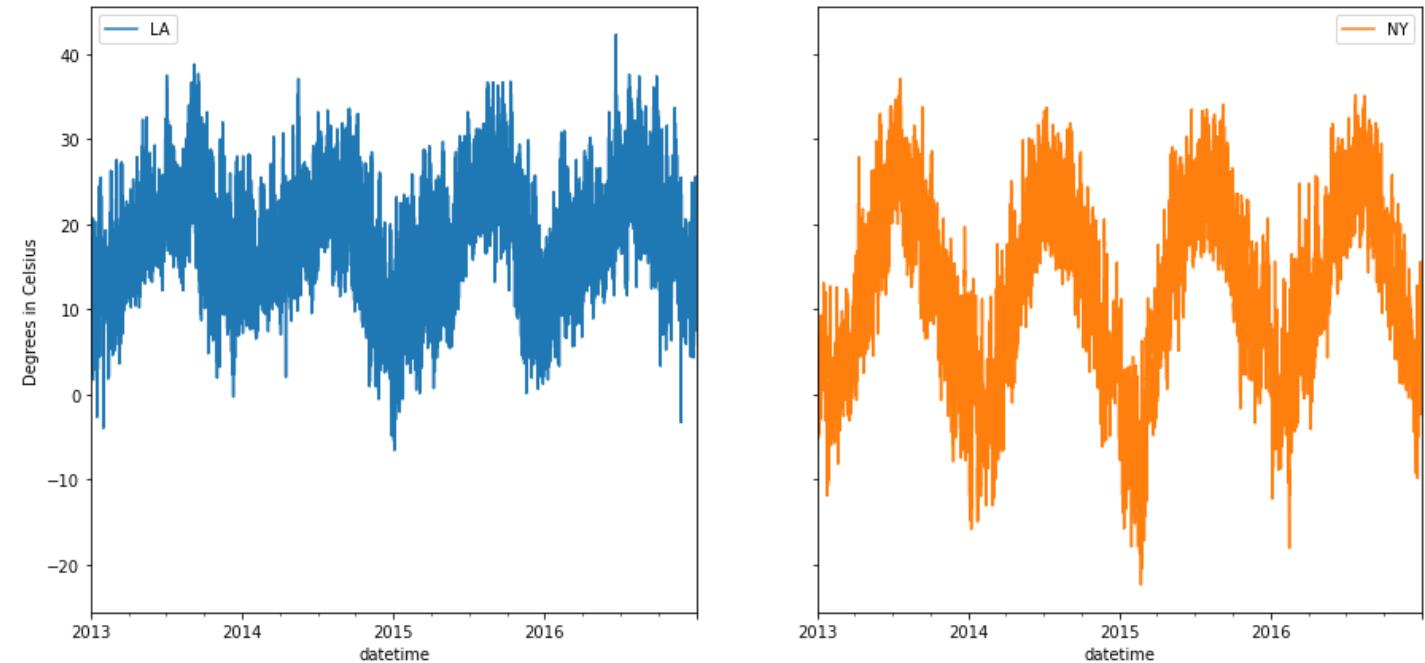
```
dataset.plot(layout = (x_row, y_columns))
```

The main flaw with the data is that there is too many datapoints.

The next graphs will show how to use different time intervals.

In [141...]

```
temp.plot(figsize = (15, 7), subplots = True, sharey = True, layout = (1, 2), ylabel = "Degrees in Celsius")  
plt.show()
```



Indexing and Slicing Time Series

In [143...]

```
import pandas as pd
```

In [144...]

```
# Must use parse_dates = [column_with_time] in order to be able to search by the index of  
temp = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\temp.csv")
```

```
In [145...]
```

```
temp.head()
```

```
Out[145...]
```

LA NY

datetime

datetime	LA	NY
2013-01-01 00:00:00	11.7	-1.1
2013-01-01 01:00:00	10.7	-1.7
2013-01-01 02:00:00	9.9	-2.0
2013-01-01 03:00:00	9.3	-2.1
2013-01-01 04:00:00	8.8	-2.3

```
In [146...]
```

```
temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 35064 entries, 2013-01-01 00:00:00 to 2016-12-31 23:00:00
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
 ---  --     --     --     --    
 0   LA      35062 non-null   float64
 1   NY      35064 non-null   float64
dtypes: float64(2)
memory usage: 821.8 KB
```

```
In [147...]
```

```
temp.loc["2013-01-01 01:00:00"]
```

```
Out[147...]
```

```
LA      10.7
NY     -1.7
Name: 2013-01-01 01:00:00, dtype: float64
```

```
In [148...]
```

```
temp.loc["2015"]
```

```
Out[148...]
```

LA NY

datetime

datetime	LA	NY
2015-01-01 00:00:00	3.8	-5.1
2015-01-01 01:00:00	4.4	-5.1
2015-01-01 02:00:00	3.2	-6.0
2015-01-01 03:00:00	1.2	-6.0
2015-01-01 04:00:00	0.2	-6.0
...
2015-12-31 19:00:00	16.0	8.1
2015-12-31 20:00:00	16.0	8.1
2015-12-31 21:00:00	16.4	7.9
2015-12-31 22:00:00	16.6	7.2
2015-12-31 23:00:00	16.8	6.2

8760 rows × 2 columns

In [149...]

```
temp.loc["2015-05"]
```

Out[149...]

LA NY

datetime	LA	NY
2015-05-01 00:00:00	25.5	13.9
2015-05-01 01:00:00	25.7	13.9
2015-05-01 02:00:00	23.8	10.5
2015-05-01 03:00:00	22.0	10.2
2015-05-01 04:00:00	20.1	8.6
...
2015-05-31 19:00:00	25.4	25.5
2015-05-31 20:00:00	26.0	23.9
2015-05-31 21:00:00	24.9	22.5
2015-05-31 22:00:00	26.0	21.3
2015-05-31 23:00:00	25.5	19.9

744 rows × 2 columns

In [150...]

```
temp.loc["2015-05-20"]
```

Out[150...]

LA NY

datetime	LA	NY
2015-05-20 00:00:00	17.7	19.8
2015-05-20 01:00:00	18.0	19.7
2015-05-20 02:00:00	16.6	19.0
2015-05-20 03:00:00	14.4	19.0
2015-05-20 04:00:00	13.3	19.7
2015-05-20 05:00:00	11.3	17.1
2015-05-20 06:00:00	11.4	17.1
2015-05-20 07:00:00	8.8	16.6
2015-05-20 08:00:00	8.2	12.5
2015-05-20 09:00:00	8.8	13.3
2015-05-20 10:00:00	7.8	13.3
2015-05-20 11:00:00	8.1	13.6
2015-05-20 12:00:00	9.7	13.6
2015-05-20 13:00:00	9.5	16.3
2015-05-20 14:00:00	10.7	16.4
2015-05-20 15:00:00	12.6	16.7

LA NY

datetime

2015-05-20 16:00:00	13.6	18.1
2015-05-20 17:00:00	15.3	18.3
2015-05-20 18:00:00	15.5	18.1
2015-05-20 19:00:00	17.7	18.1
2015-05-20 20:00:00	18.4	17.8
2015-05-20 21:00:00	18.0	17.8
2015-05-20 22:00:00	19.1	14.2
2015-05-20 23:00:00	19.1	14.2

In [159...]

```
# This dataset shows 24 hours the two columns show the weather for the two cities.  
temp.loc["2015-05-20"].shape
```

Out[159...]

(24, 2)

In [151...]

```
temp.loc["2015-05-20 10:00:00"]
```

Out[151...]

```
LA      7.8  
NY     13.3  
Name: 2015-05-20 10:00:00, dtype: float64
```

In []:

```
# Note that this data won't work because the data only gives hourly. It does not give minutes.  
temp.loc["2015-05-20 10:30:00"]
```

In [160...]

```
temp.loc["2015-01-01" : "2015-12-31"]
```

Out[160...]

LA NY

datetime

2015-01-01 00:00:00	3.8	-5.1
2015-01-01 01:00:00	4.4	-5.1
2015-01-01 02:00:00	3.2	-6.0
2015-01-01 03:00:00	1.2	-6.0
2015-01-01 04:00:00	0.2	-6.0
...
2015-12-31 19:00:00	16.0	8.1
2015-12-31 20:00:00	16.0	8.1
2015-12-31 21:00:00	16.4	7.9
2015-12-31 22:00:00	16.6	7.2
2015-12-31 23:00:00	16.8	6.2

8760 rows × 2 columns

```
In [162...]  
# There are different ways to express different time intervals.  
# The first shows the year from the first day to the last day of the year.  
# The second just passes the year by itself.  
temp.loc["2015-01-01" : "2015-12-31"].equals(temp.loc["2015"])
```

```
Out[162...]  
True
```

```
In [163...]  
temp.loc[:, "2015-05-20"]
```

```
Out[163...]  
LA    NY  
datetime  
2013-01-01 00:00:00  11.7  -1.1  
2013-01-01 01:00:00  10.7  -1.7  
2013-01-01 02:00:00  9.9   -2.0  
2013-01-01 03:00:00  9.3   -2.1  
2013-01-01 04:00:00  8.8   -2.3  
...    ...  ...  
2015-05-20 19:00:00  17.7  18.1  
2015-05-20 20:00:00  18.4  17.8  
2015-05-20 21:00:00  18.0  17.8  
2015-05-20 22:00:00  19.1  14.2  
2015-05-20 23:00:00  19.1  14.2
```

20880 rows × 2 columns

DDMonthYYYY

This is an alternative format of typing in dates.

```
In [165...]  
# Can pass in the date in with the DDMonthYYYY no spaces.  
temp.loc["20February2015"]
```

```
Out[165...]  
LA    NY  
datetime  
2015-02-20 00:00:00  16.4  -12.4  
2015-02-20 01:00:00  17.5  -12.4  
2015-02-20 02:00:00  14.6  -14.5  
2015-02-20 03:00:00  13.9  -14.5  
2015-02-20 04:00:00  10.3  -14.5  
2015-02-20 05:00:00  8.9   -15.9  
2015-02-20 06:00:00  9.0   -15.9  
2015-02-20 07:00:00  7.1   -15.9
```

LA NY

datetime

2015-02-20 08:00:00	6.6	-16.8
2015-02-20 09:00:00	6.3	-16.8
2015-02-20 10:00:00	5.8	-16.8
2015-02-20 11:00:00	5.5	-17.4
2015-02-20 12:00:00	5.8	-17.4
2015-02-20 13:00:00	5.2	-17.4
2015-02-20 14:00:00	5.3	-14.0
2015-02-20 15:00:00	10.5	-14.2
2015-02-20 16:00:00	8.9	-14.0
2015-02-20 17:00:00	12.4	-10.4
2015-02-20 18:00:00	12.0	-10.2
2015-02-20 19:00:00	16.8	-9.9
2015-02-20 20:00:00	17.7	-9.3
2015-02-20 21:00:00	17.2	-9.3
2015-02-20 22:00:00	18.5	-9.3
2015-02-20 23:00:00	18.5	-14.0

In [167...]

```
# Shows the tenth hour of the day and the 12th hour of the day.
temp.loc[['2015-05-20 10:00:00', '2015-05-20 12:00:00']]
```

Out[167...]

LA NY

datetime

2015-05-20 10:00:00	7.8	13.3
2015-05-20 12:00:00	9.7	13.6

Can pass in a list with the two time stamps, saved to one variable.

In [176...]

```
two_timestamps = pd.to_datetime(['2015-05-20 10:00:00', '2015-05-20 12:00:00'])
two_timestamps
```

Out[176...]

```
DatetimeIndex(['2015-05-20 10:00:00', '2015-05-20 12:00:00'], dtype='datetime64[ns]', freq=None)
```

In [177...]

```
temp.loc[two_timestamps]
```

Out[177...]

LA NY

2015-05-20 10:00:00	7.8	13.3
2015-05-20 12:00:00	9.7	13.6

Important

Create a Customized Isolated DatetimeIndex with:

pd.date_range()

pd.date_range(start = x, end = y) freq can go inside a pd.date_range() It can equal "D", "W", "M" which signifies day, week, and month respectively.

```
In [185... pd.to_datetime(["2015-05-20", "Feb 20 2015"])
```

```
Out[185... DatetimeIndex(['2015-05-20', '2015-02-20'], dtype='datetime64[ns]', freq=None)
```

```
In [186... # freq = "D" corresponds with frequency every day that is the default
pd.date_range(start = "2015-07-01", end = "2015-07-31")
```

```
Out[186... DatetimeIndex(['2015-07-01', '2015-07-02', '2015-07-03', '2015-07-04',
 '2015-07-05', '2015-07-06', '2015-07-07', '2015-07-08',
 '2015-07-09', '2015-07-10', '2015-07-11', '2015-07-12',
 '2015-07-13', '2015-07-14', '2015-07-15', '2015-07-16',
 '2015-07-17', '2015-07-18', '2015-07-19', '2015-07-20',
 '2015-07-21', '2015-07-22', '2015-07-23', '2015-07-24',
 '2015-07-25', '2015-07-26', '2015-07-27', '2015-07-28',
 '2015-07-29', '2015-07-30', '2015-07-31'],
 dtype='datetime64[ns]', freq='D')
```

```
In [187... # freq = "W" means that each week will be recorded
pd.date_range(start = "2015-07-01", end = "2015-07-31", freq = "W")
```

```
Out[187... DatetimeIndex(['2015-07-05', '2015-07-12', '2015-07-19', '2015-07-26'],
 dtype='datetime64[ns]', freq='W-SUN')
```

```
In [189... # freq = "M" means that each month will be recorded
pd.date_range(start = "2015-07-01", end = "2015-07-31", freq = "M")
```

```
Out[189... DatetimeIndex(['2015-07-31'],
 dtype='datetime64[ns]', freq='M')
```

Important

Note that you can simply type pd.d and hit tab and it will finish pd.date_range

Make a Date range without using an end.

Can utilize pd.date_range(start = x, periods = y, freq = "z")

A note on formating:

Note that periods has to have an "s" at the end and that the freq can be capital or lowercased.

```
In [195... # Periods defines how many of the frequencies should be created from the start date.
pd.date_range(start = "2015-07-01", periods = 31, freq = "D")
```

```
Out[195... DatetimeIndex(['2015-07-01', '2015-07-02', '2015-07-03', '2015-07-04',
```

```
'2015-07-05', '2015-07-06', '2015-07-07', '2015-07-08',
'2015-07-09', '2015-07-10', '2015-07-11', '2015-07-12',
'2015-07-13', '2015-07-14', '2015-07-15', '2015-07-16',
'2015-07-17', '2015-07-18', '2015-07-19', '2015-07-20',
'2015-07-21', '2015-07-22', '2015-07-23', '2015-07-24',
'2015-07-25', '2015-07-26', '2015-07-27', '2015-07-28',
'2015-07-29', '2015-07-30', '2015-07-31'],
dtype='datetime64[ns]', freq='D')
```

In [201...]

```
# For this let's assume that the user wants 24 months from 'new years day' 2015 for interval
pd.date_range(start = "2015-01-01", periods = 24, freq = "M")
```

Out[201...]

```
DatetimeIndex(['2015-01-31', '2015-02-28', '2015-03-31', '2015-04-30',
                '2015-05-31', '2015-06-30', '2015-07-31', '2015-08-31',
                '2015-09-30', '2015-10-31', '2015-11-30', '2015-12-31',
                '2016-01-31', '2016-02-29', '2016-03-31', '2016-04-30',
                '2016-05-31', '2016-06-30', '2016-07-31', '2016-08-31',
                '2016-09-30', '2016-10-31', '2016-11-30', '2016-12-31'],
                dtype='datetime64[ns]', freq='M')
```

freq has a great option that signifies business days. freq = "B"

```
pd.date_range(freq = "B")
```

In [214...]

```
# In 2015 December 25th or "Christmas" fell on a Friday.
# According to this calendar it is still a buisiness day.
# The Business day does not recognize holidays, as tested with Christmas and independence day
# Only weekends are considered non-business days.
pd.date_range(start = "2015-12-01", periods = 31, freq = "B")
```

Out[214...]

```
DatetimeIndex(['2015-12-01', '2015-12-02', '2015-12-03', '2015-12-04',
                '2015-12-07', '2015-12-08', '2015-12-09', '2015-12-10',
                '2015-12-11', '2015-12-14', '2015-12-15', '2015-12-16',
                '2015-12-17', '2015-12-18', '2015-12-21', '2015-12-22',
                '2015-12-23', '2015-12-24', '2015-12-25', '2015-12-28',
                '2015-12-29', '2015-12-30', '2015-12-31', '2016-01-01',
                '2016-01-04', '2016-01-05', '2016-01-06', '2016-01-07',
                '2016-01-08', '2016-01-11', '2016-01-12'],
                dtype='datetime64[ns]', freq='B')
```

Hourly is an option for these pd.date_range() as well.

```
pd.date_range(freq = "H")
```

In [216...]

```
# Can map out a whole day using the hourly frequency. freq = "H"
pd.date_range(start = "2016-08-25", periods = 24, freq = "H")
```

Out[216...]

```
DatetimeIndex(['2016-08-25 00:00:00', '2016-08-25 01:00:00',
                '2016-08-25 02:00:00', '2016-08-25 03:00:00',
                '2016-08-25 04:00:00', '2016-08-25 05:00:00',
                '2016-08-25 06:00:00', '2016-08-25 07:00:00',
                '2016-08-25 08:00:00', '2016-08-25 09:00:00',
                '2016-08-25 10:00:00', '2016-08-25 11:00:00',
                '2016-08-25 12:00:00', '2016-08-25 13:00:00',
                '2016-08-25 14:00:00', '2016-08-25 15:00:00',
                '2016-08-25 16:00:00', '2016-08-25 17:00:00',
                '2016-08-25 18:00:00', '2016-08-25 19:00:00',
                '2016-08-25 20:00:00', '2016-08-25 21:00:00',
                '2016-08-25 22:00:00', '2016-08-25 23:00:00'],
                dtype='datetime64[ns]', freq='H')
```

In []:

```
## Periods should be +1 more than their intended value.
```

For example `if` you want to have a daterange of 12 hours put periods = 13 . This should be

In [223...]

```
# The Hour does not have to start at midnight if you define it.  
# Now this shows a list of times from 9 am to 9 pm.  
pd.date_range(start = "2016-08-25 09:00:00", periods = 13, freq = "H")
```

Out[223...]

```
DatetimeIndex(['2016-08-25 09:00:00', '2016-08-25 10:00:00',  
                '2016-08-25 11:00:00', '2016-08-25 12:00:00',  
                '2016-08-25 13:00:00', '2016-08-25 14:00:00',  
                '2016-08-25 15:00:00', '2016-08-25 16:00:00',  
                '2016-08-25 17:00:00', '2016-08-25 18:00:00',  
                '2016-08-25 19:00:00', '2016-08-25 20:00:00',  
                '2016-08-25 21:00:00'],  
               dtype='datetime64[ns]', freq='H')
```

In [225...]

```
# Note that the date_range() saves into a list.  
# This list can be tapped into using brackets.  
pd.date_range(start = "2016-08-25 09:00:00", periods = 13, freq = "H")[-1]
```

Out[225...]

```
Timestamp('2016-08-25 21:00:00', freq='H')
```

In [227...]

```
# Weekly Frequency freq = "W" by default it will pick Sunday to count from.  
# Each period would go from one Sunday to the next.  
pd.date_range(start = "2015-07-01", end = "2015-07-31", freq = "W")
```

Out[227...]

```
DatetimeIndex(['2015-07-05', '2015-07-12', '2015-07-19', '2015-07-26'], dtype='datetime64[ns]', freq='W-SUN')
```

In [228...]

```
# A new reference point from the week can be chosen by putting freq = "W-Mon" etc.  
# Just put a dash connecting W and the three letter weekday abbreviation.  
pd.date_range(start = "2015-07-01", end = "2015-07-31", freq = "W-Wed")
```

Out[228...]

```
DatetimeIndex(['2015-07-01', '2015-07-08', '2015-07-15', '2015-07-22',  
                '2015-07-29'],  
               dtype='datetime64[ns]', freq='W-WED')
```

In [229...]

```
# The Month categories counts counts to the closes corresponding month value of the next month.  
# The start date determines the dates of the next month dates.  
# If you start on the 31st the program will try to get as close as it can to that day. For example:  
pd.date_range(start = "2015-07-14", periods = 6, freq = "M")
```

Out[229...]

```
DatetimeIndex(['2015-07-31', '2015-08-31', '2015-09-30', '2015-10-31',  
                '2015-11-30', '2015-12-31'],  
               dtype='datetime64[ns]', freq='M')
```

In [231...]

```
# freq = "MS" will count months from the 1st of the new month every time.  
# The same code as last cell is used except with MS.  
pd.date_range(start = "2015-07-14", periods = 6, freq = "MS")
```

Out[231...]

```
DatetimeIndex(['2015-08-01', '2015-09-01', '2015-10-01', '2015-11-01',  
                '2015-12-01', '2016-01-01'],  
               dtype='datetime64[ns]', freq='MS')
```

Using pd.DateOffset()

In [232...]

```
pd.date_range(start = "2015-07-14", periods = 6, freq = pd.DateOffset(months = 1))
```

Out[232...]

```
DatetimeIndex(['2015-07-14', '2015-08-14', '2015-09-14', '2015-10-14',  
                '2015-11-14', '2015-12-14'],  
               dtype='datetime64[ns]', freq='MS')
```

```
'2015-11-14', '2015-12-14'],  
dtype='datetime64[ns]', freq='<DateOffset: months=1>')
```

Use freq = "Q" for the last day of quarters.

Gives the closest end to each quarter. There are four quarters of three months in every year.

```
In [234... pd.date_range(start = "2015-07-14", periods = 6, freq = "Q")
```

```
Out[234... DatetimeIndex(['2015-09-30', '2015-12-31', '2016-03-31', '2016-06-30',  
'2016-09-30', '2016-12-31'],  
dtype='datetime64[ns]', freq='Q-DEC')
```

Use freq = "QS" this will give the first date of the quarter.

Gives the closest start to each quarter. Only retrieves quarter start dates that come after the start date.

```
In [237... pd.date_range(start = "2015-07-14", periods = 6, freq = "QS")
```

```
Out[237... DatetimeIndex(['2015-10-01', '2016-01-01', '2016-04-01', '2016-07-01',  
'2016-10-01', '2017-01-01'],  
dtype='datetime64[ns]', freq='QS-JAN')
```

```
In [239... # Can Use different months to determine the results.  
# Use freq = "QS-Abbreviated_month"  
pd.date_range(start = "2015-07-14", periods = 6, freq = "QS-May")
```

```
Out[239... DatetimeIndex(['2015-08-01', '2015-11-01', '2016-02-01', '2016-05-01',  
'2016-08-01', '2016-11-01'],  
dtype='datetime64[ns]', freq='QS-MAY')
```

```
In [240... pd.date_range(start = "2015-07-14", periods = 6, freq = "Q-May")
```

```
Out[240... DatetimeIndex(['2015-08-31', '2015-11-30', '2016-02-29', '2016-05-31',  
'2016-08-31', '2016-11-30'],  
dtype='datetime64[ns]', freq='Q-MAY')
```

Annual Year freq = "A" or freq = "Y"

Either options will get the end of the year.

```
In [244... pd.date_range(start = "2015-07-14", periods = 6, freq = "A")
```

```
Out[244... DatetimeIndex(['2015-12-31', '2016-12-31', '2017-12-31', '2018-12-31',  
'2019-12-31', '2020-12-31'],  
dtype='datetime64[ns]', freq='A-DEC')
```

Annual Year Start freq = "AS" or freq = "YS"

Note that S stands for start in this instance.

```
In [246... pd.date_range(start = "2015-07-14", periods = 6, freq = "AS")
```

```
Out[246... DatetimeIndex(['2016-01-01', '2017-01-01', '2018-01-01', '2019-01-01',  
'2020-01-01', '2021-01-01'],  
dtype='datetime64[ns]', freq='AS-JAN')
```

Seeing Annual days accross targeting a specific month.

An example would be seeing the last day of June across several years.

```
In [248... pd.date_range(start = "2015-07-14", periods = 6, freq = "A-Jun")
```

```
Out[248... DatetimeIndex(['2016-06-30', '2017-06-30', '2018-06-30', '2019-06-30',
   '2020-06-30', '2021-06-30'],
  dtype='datetime64[ns]', freq='A-JUN')
```

```
In [249... pd.date_range(start = "2015-07-14", periods = 6, freq = "AS-Jun")
```

```
Out[249... DatetimeIndex(['2016-06-01', '2017-06-01', '2018-06-01', '2019-06-01',
   '2020-06-01', '2021-06-01'],
  dtype='datetime64[ns]', freq='AS-JUN')
```

pd.DateOffset is very important as it lets you see the past dates.

Use pd.Offset() to see events that happened in the past.

pd.DateOffset("years/months = x")

```
In [251... # Can see your last few work anniversaries etc.
pd.date_range(end = "2018-11-24", periods = 10, freq = pd.DateOffset(years = 1))
```

```
Out[251... DatetimeIndex(['2009-11-24', '2010-11-24', '2011-11-24', '2012-11-24',
   '2013-11-24', '2014-11-24', '2015-11-24', '2016-11-24',
   '2017-11-24', '2018-11-24'],
  dtype='datetime64[ns]', freq='<DateOffset: years=1>')
```

Important

pd.date_range(freq = "H")

The pd.date_range() can be used for hourly data as well.

```
In [252... pd.date_range(start = "2015-07-01", periods = 10, freq = "H")
```

```
Out[252... DatetimeIndex(['2015-07-01 00:00:00', '2015-07-01 01:00:00',
   '2015-07-01 02:00:00', '2015-07-01 03:00:00',
   '2015-07-01 04:00:00', '2015-07-01 05:00:00',
   '2015-07-01 06:00:00', '2015-07-01 07:00:00',
   '2015-07-01 08:00:00', '2015-07-01 09:00:00'],
  dtype='datetime64[ns]', freq='H')
```

Can pass in a quantifier before the freq unit.

Every two hours can be counted instead of one.

```
In [255... # Frequency of 2 hours
pd.date_range(start = "2015-07-01", periods = 10, freq = "2H")
```

```
Out[255... DatetimeIndex(['2015-07-01 00:00:00', '2015-07-01 02:00:00',
   '2015-07-01 04:00:00', '2015-07-01 06:00:00',
   '2015-07-01 08:00:00', '2015-07-01 10:00:00',
   '2015-07-01 12:00:00', '2015-07-01 14:00:00',
   '2015-07-01 16:00:00', '2015-07-01 18:00:00'],
  dtype='datetime64[ns]', freq='2H')
```

```
In [263... # Frequency is showing periods of 3 days and 8 hours.
pd.date_range(start = "2015-07-01", periods = 10, freq = "3D8H")
```

```
Out[263... DatetimeIndex(['2015-07-01 00:00:00', '2015-07-04 08:00:00',
   '2015-07-07 16:00:00', '2015-07-11 00:00:00',
   '2015-07-14 08:00:00', '2015-07-17 16:00:00',
   '2015-07-21 00:00:00', '2015-07-24 08:00:00',
   '2015-07-27 16:00:00', '2015-07-31 00:00:00'],
  dtype='datetime64[ns]', freq='80H')
```

Important

Downsampling

Convert the frequency of the data. An example would be converting data from hourly to daily. Currently the data looks bad because the hourly data is too much to chart.

```
In [265... import pandas as pd
import matplotlib.pyplot as plt
plt.style.use("seaborn")
```

```
In [266... temp = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\temp.csv")
```

```
In [267... temp.head()
```

```
Out[267...          LA    NY
      datetime
2013-01-01 00:00:00  11.7  -1.1
2013-01-01 01:00:00  10.7  -1.7
2013-01-01 02:00:00   9.9  -2.0
2013-01-01 03:00:00   9.3  -2.1
2013-01-01 04:00:00   8.8  -2.3
```

```
In [268... temp.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 35064 entries, 2013-01-01 00:00:00 to 2016-12-31 23:00:00
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  --     --     --     --    
 0   LA      35062 non-null   float64
 1   NY      35064 non-null   float64
dtypes: float64(2)
memory usage: 821.8 KB
```

Very important

The `.resample(rule = "day_month_or_year")` can be used to change the data to fit a different interval. I.e hour to days.

`column_name.resample(rule = "day_month_or_year")`

Pass in the type of data that you need to convert using rule.

In [274...]

```
temp.resample(rule = "D")
```

Out[274...]

```
<pandas.core.resample.DatetimeIndexResampler object at 0x000001E6658A7BB0>
```

In [407...]

```
# This put the data into a daily aggregation instead of an hourly one.
list(temp.resample(rule = "D"))[0:2]
```

Out[407...]

```
[Timestamp('2013-01-01 00:00:00', freq='D'),
```

```
LA NY
```

datetime	LA	NY
2013-01-01 00:00:00	11.7	-1.1
2013-01-01 01:00:00	10.7	-1.7
2013-01-01 02:00:00	9.9	-2.0
2013-01-01 03:00:00	9.3	-2.1
2013-01-01 04:00:00	8.8	-2.3
2013-01-01 05:00:00	8.7	-2.5
2013-01-01 06:00:00	6.9	-3.2
2013-01-01 07:00:00	7.8	-3.4
2013-01-01 08:00:00	6.7	-3.0
2013-01-01 09:00:00	6.6	-1.8
2013-01-01 10:00:00	6.1	-1.4
2013-01-01 11:00:00	5.6	-1.8
2013-01-01 12:00:00	5.1	-1.7
2013-01-01 13:00:00	5.2	-1.5
2013-01-01 14:00:00	4.6	-1.0
2013-01-01 15:00:00	5.1	-0.0
2013-01-01 16:00:00	6.2	1.2
2013-01-01 17:00:00	9.3	2.2
2013-01-01 18:00:00	11.4	3.3
2013-01-01 19:00:00	12.0	3.5
2013-01-01 20:00:00	12.9	3.1
2013-01-01 21:00:00	13.9	2.4
2013-01-01 22:00:00	14.1	2.4
2013-01-01 23:00:00	14.0	2.7),

```
(Timestamp('2013-01-02 00:00:00', freq='D'),
```

```
LA NY
```

datetime	LA	NY
2013-01-02 00:00:00	13.2	2.6
2013-01-02 01:00:00	11.8	2.7
2013-01-02 02:00:00	10.5	2.9
2013-01-02 03:00:00	9.5	2.9
2013-01-02 04:00:00	8.3	2.9
2013-01-02 05:00:00	8.0	3.5
2013-01-02 06:00:00	7.5	3.7
2013-01-02 07:00:00	7.1	3.5
2013-01-02 08:00:00	6.4	3.7
2013-01-02 09:00:00	6.0	3.6
2013-01-02 10:00:00	5.9	3.5
2013-01-02 11:00:00	6.1	3.6
2013-01-02 12:00:00	5.8	3.5
2013-01-02 13:00:00	5.6	3.5

```
2013-01-02 14:00:00  5.8  3.7
2013-01-02 15:00:00  5.9  4.0
2013-01-02 16:00:00  6.4  4.0
2013-01-02 17:00:00  9.0  3.6
2013-01-02 18:00:00  11.5 3.1
2013-01-02 19:00:00  13.3 3.2
2013-01-02 20:00:00  14.2 3.2
2013-01-02 21:00:00  15.0 2.9
2013-01-02 22:00:00  14.9 2.0
2013-01-02 23:00:00  15.1 1.2) ]
```

```
In [277...]: len(list(temp.resample(rule = "D")))
```

```
Out[277...]: 1461
```

```
In [284...]: # The first number taps into the first day. The second number [1] taps into the actual list
# Now each day is summarized into a list of temperature fluctuations throughout the day.
list(temp.resample(rule = "D"))[0][1]
```

```
Out[284...]:
```

LA NY

datetime	LA	NY
2013-01-01 00:00:00	11.7	-1.1
2013-01-01 01:00:00	10.7	-1.7
2013-01-01 02:00:00	9.9	-2.0
2013-01-01 03:00:00	9.3	-2.1
2013-01-01 04:00:00	8.8	-2.3
2013-01-01 05:00:00	8.7	-2.5
2013-01-01 06:00:00	6.9	-3.2
2013-01-01 07:00:00	7.8	-3.4
2013-01-01 08:00:00	6.7	-3.0
2013-01-01 09:00:00	6.6	-1.8
2013-01-01 10:00:00	6.1	-1.4
2013-01-01 11:00:00	5.6	-1.8
2013-01-01 12:00:00	5.1	-1.7
2013-01-01 13:00:00	5.2	-1.5
2013-01-01 14:00:00	4.6	-1.0
2013-01-01 15:00:00	5.1	-0.0
2013-01-01 16:00:00	6.2	1.2
2013-01-01 17:00:00	9.3	2.2
2013-01-01 18:00:00	11.4	3.3
2013-01-01 19:00:00	12.0	3.5
2013-01-01 20:00:00	12.9	3.1
2013-01-01 21:00:00	13.9	2.4
2013-01-01 22:00:00	14.1	2.4

LA NY

datetime

2013-01-01 23:00:00 14.0 2.7

In [288...]

```
# Tap into the dataframe the hours are grouped into days using the resample.  
# The first entry per day or the temperature at midnight is taken for each day via the .first()  
temp.resample("D").first()
```

Out[288...]

LA NY

datetime

2013-01-01 11.7 -1.1

2013-01-02 13.2 2.6

2013-01-03 15.1 0.3

2013-01-04 16.3 -1.2

2013-01-05 18.1 -1.2

...

2016-12-27 15.1 4.1

2016-12-28 19.9 11.2

2016-12-29 23.3 2.1

2016-12-30 25.5 3.0

2016-12-31 15.7 0.8

1461 rows × 2 columns

Condensing Data with resampling

column.resample("D").mean()

Day could be replaced with any other interval. Mean could be replaced with median etc.

In []:

```
## Stringing along methods to get better results.  
The data that is gathered from hours to days can be further processed.  
## Method string together .mean() etc.
```

In [291...]

temp.resample("D").mean()

Out[291...]

LA NY

datetime

2013-01-01 8.858333 -0.404167

2013-01-02 9.283333 3.208333

2013-01-03 10.304167 -2.425000

LA NY

datetime	LA	NY
2013-01-04	11.512500	-2.070833
2013-01-05	11.083333	0.816667
...
2016-12-27	12.154167	10.579167
2016-12-28	14.433333	4.016667
2016-12-29	16.045833	1.312500
2016-12-30	15.933333	2.204167
2016-12-31	13.275000	1.204167

1461 rows × 2 columns

More examples of method stringing with resampling.

In [293...]

```
# Samples the first two hours.
temp.resample("2H").first()
```

Out[293...]

LA NY

datetime	LA	NY
2013-01-01 00:00:00	11.7	-1.1
2013-01-01 02:00:00	9.9	-2.0
2013-01-01 04:00:00	8.8	-2.3
2013-01-01 06:00:00	6.9	-3.2
2013-01-01 08:00:00	6.7	-3.0
...
2016-12-31 14:00:00	12.7	-1.3
2016-12-31 16:00:00	12.6	1.1
2016-12-31 18:00:00	13.2	3.4
2016-12-31 20:00:00	13.2	5.7
2016-12-31 22:00:00	12.3	5.7

17532 rows × 2 columns

In [308...]

```
# Samples by two hours instead of one. Takes the first hour.
temp.resample("2H").first().head(15)
```

Out[308...]

LA NY

datetime	LA	NY
2013-01-01 00:00:00	11.7	-1.1
2013-01-01 02:00:00	9.9	-2.0

LA NY

datetime

2013-01-01 04:00:00	8.8	-2.3
2013-01-01 06:00:00	6.9	-3.2
2013-01-01 08:00:00	6.7	-3.0
2013-01-01 10:00:00	6.1	-1.4
2013-01-01 12:00:00	5.1	-1.7
2013-01-01 14:00:00	4.6	-1.0
2013-01-01 16:00:00	6.2	1.2
2013-01-01 18:00:00	11.4	3.3
2013-01-01 20:00:00	12.9	3.1
2013-01-01 22:00:00	14.1	2.4
2013-01-02 00:00:00	13.2	2.6
2013-01-02 02:00:00	10.5	2.9
2013-01-02 04:00:00	8.3	2.9

In [295...]

```
temp.resample("W-Wed").mean()
```

Out[295...]

LA NY

datetime

2013-01-02	9.070833	1.402083
2013-01-09	11.033333	1.033929
2013-01-16	8.870238	6.001190
2013-01-23	14.678571	1.010714
2013-01-30	12.554762	-4.382738
...
2016-12-07	13.205357	5.964286
2016-12-14	14.490476	1.228571
2016-12-21	13.209524	-2.248810
2016-12-28	11.930357	4.688095
2017-01-04	15.084722	1.573611

210 rows × 2 columns

In [307...]

```
# Sort by the end of the month
temp.resample("M").mean().head()
```

Out[307...]

LA NY

datetime

LA NY

datetime

2013-01-31	11.596237	1.129570
2013-02-28	12.587202	0.617857
2013-03-31	15.069946	3.719220
2013-04-30	16.487361	10.699306
2013-05-31	19.005780	15.824328

In [306...]

```
# By the start of the month
temp.resample("MS").mean().head()
```

Out[306...]

LA NY

datetime

2013-01-01	11.596237	1.129570
2013-02-01	12.587202	0.617857
2013-03-01	15.069946	3.719220
2013-04-01	16.487361	10.699306
2013-05-01	19.005780	15.824328

In [298...]

```
# Deprecated:
temp.resample("M", loffset = "15D").mean()
```

C:\Users\alonz\AppData\Local\Temp\ipykernel_2928\1291155925.py:1: FutureWarning: 'loffset' in .resample() and in Grouper() is deprecated.

```
>>> df.resample(freq="3s", loffset="8H")
```

becomes:

```
>>> from pandas.tseries.frequencies import to_offset
>>> df = df.resample(freq="3s").mean()
>>> df.index = df.index.to_timestamp() + to_offset("8H")
>>> temp.resample("M", loffset = "15D").mean()
```

Out[298...]

LA NY

datetime

2013-02-15	11.596237	1.129570
2013-03-15	12.587202	0.617857
2013-04-15	15.069946	3.719220
2013-05-15	16.487361	10.699306
2013-06-15	19.005780	15.824328
2013-07-15	19.905417	22.225694
2013-08-15	22.093952	26.329704
2013-09-15	21.513172	22.480376

LA NY

datetime	LA	NY
2013-10-15	22.404861	18.291806
2013-11-15	16.620430	14.335215
2013-12-15	15.107917	7.111944
2014-01-15	13.416935	2.776210
2014-02-15	16.247715	-2.210349
2014-03-15	14.326637	-1.596280
2014-04-15	15.836156	1.994758
2014-05-15	16.783472	10.128611
2014-06-15	20.041398	16.829167
2014-07-15	19.810556	21.784861
2014-08-15	23.056183	24.019624
2014-09-15	22.146102	21.848387
2014-10-15	21.620278	19.267222
2014-11-15	16.968280	14.265860
2014-12-15	11.322083	5.719167
2015-01-15	8.523925	3.513844
2015-02-15	9.527016	-4.179301
2015-03-15	11.587649	-7.239732
2015-04-15	14.465995	1.020161
2015-05-15	13.527917	10.180417
2015-06-15	14.540054	18.731586
2015-07-15	20.995000	20.347500
2015-08-15	22.899059	24.252151
2015-09-15	24.846909	24.046774
2015-10-15	24.586250	21.589861
2015-11-15	21.785753	12.656452
2015-12-15	15.255278	10.117500
2016-01-15	11.919758	8.662500
2016-02-15	12.509274	0.168952
2016-03-15	16.600431	2.069971
2016-04-15	15.686425	8.070430
2016-05-15	17.726111	10.535556
2016-06-15	17.375403	15.874462
2016-07-15	22.536111	21.462639
2016-08-15	24.541532	24.767608

LA NY

datetime	LA	NY
2016-09-15	23.983199	25.351882
2016-10-15	22.379306	21.032778
2016-11-15	16.137903	14.321640
2016-12-15	17.196111	8.539722
2017-01-15	13.390457	2.327285

```
# Nondeprecated code >>> df.resample(freq="3s", loffset="8H") becomes: >>> from pandas.tseries.frequencies import  
to_offset >>> df = df.resample(freq="3s").mean() >>> df.index = df.index.to_timestamp() + to_offset("8H") temp.resample("M",  
loffset = "15D").mean()
```

In [299...]
Gives the quarterly averages of the data.
temp.resample("Q").mean()

Out[299...]

LA NY

datetime	LA	NY
2013-03-31	13.099212	1.862361
2013-06-30	18.472115	16.245101
2013-09-30	21.999638	22.411594
2013-12-31	15.047781	8.084918
2014-03-31	15.508287	-0.570880
2014-06-30	18.891255	16.253938
2014-09-30	22.281295	21.738315
2014-12-31	12.281748	7.855933
2015-03-31	11.869306	-3.340509
2015-06-30	16.334386	16.445238
2015-09-30	24.105571	23.314810
2015-12-31	16.331839	10.482745
2016-03-31	14.895375	3.466484
2016-06-30	19.192353	15.956639
2016-09-30	23.648324	23.746603
2016-12-31	15.557201	8.394656

In [300...]
Makes Feb the last quarter. Resamples base off of that.
temp.resample("Q-Feb").mean()

Out[300...]

LA NY

datetime	LA	NY
2013-02-28	12.066525	0.886723
2013-05-31	16.859973	10.074230

LA NY

datetime	LA	NY
2013-08-31	21.184601	23.694384
2013-11-30	18.028755	13.258288
2014-02-28	14.675000	-0.301713
2014-05-31	17.562047	9.645652
2014-08-31	21.691168	22.559284
2014-11-30	16.640522	13.097070
2015-02-28	9.822593	-2.481574
2015-05-31	14.185054	9.975181
2015-08-31	22.934511	22.909692
2015-11-30	20.556090	14.764515
2016-02-29	13.612225	3.668178
2016-05-31	16.920652	11.503895
2016-08-31	23.699457	23.886775
2016-11-30	18.544368	14.627976
2017-02-28	13.390457	2.327285

Resample by year by using freq = "Y" or freq = "A"

In [304...]: `temp.resample("Y").mean()`

Out[304...]:

LA NY

datetime	LA	NY
2013-12-31	17.174229	12.196153
2014-12-31	17.245616	11.370959
2015-12-31	17.191530	11.795194
2016-12-31	18.330305	12.908470

Downsampling Time series with .resample() pt 2

Need to be mindul of how data can change based on resampling methods.

Notice how the labels of the time can change depending on the resampling method.

In [305...]: `temp.resample("M").mean().head()`

Out[305...]:

LA NY

datetime	LA	NY
----------	----	----

LA NY

datetime

2013-01-31	11.596237	1.129570
2013-02-28	12.587202	0.617857
2013-03-31	15.069946	3.719220
2013-04-30	16.487361	10.699306
2013-05-31	19.005780	15.824328

In [316...]

```
# Samples the start of the month rather than the end.  
# The first resample shows the last day of Jan. The one below shows the first due to "MS".  
temp.resample("MS").mean().head()
```

Out[316...]

LA NY

datetime

2013-01-01	11.596237	1.129570
2013-02-01	12.587202	0.617857
2013-03-01	15.069946	3.719220
2013-04-01	16.487361	10.699306
2013-05-01	19.005780	15.824328

Do NOT use:

.resample(kind = "timestamp")

Don't use this default value. It makes the intervals that were originally listed on the datachart change due to the resampling method.

In [320...]

```
# Notice how the dates change.  
temp.resample("MS", kind = "timestamp").mean().head(2)
```

Out[320...]

LA NY

datetime

2013-01-01	11.596237	1.129570
2013-02-01	12.587202	0.617857

In [321...]

```
# This shows the end of the month while the other shows the start of the month.  
temp.resample("M", kind = "timestamp").mean().head(2)
```

Out[321...]

LA NY

datetime

2013-01-31	11.596237	1.129570
-------------------	-----------	----------

LA NY

datetime

2013-02-28 12.587202 0.617857

Use the kind = "period" for a consistent regardless of "M" vs "MS" etc.

`column.resample("M", kind = "period")`

In [326...]

```
# For the start of the month.  
# Now it has the first month of 2013.  
temp.resample("M", kind = "period").mean().head(2)
```

Out[326...]

LA NY

datetime

2013-01 11.596237 1.129570
2013-02 12.587202 0.617857

Note that when kind = "period" you can no longer put things like: "MS"

That is a good thing.

You can only put things such as "M" this is because the start of the month and the end of the month no longer matters.

In []:

```
# This code will throw an error  
temp.resample("MS", kind = "period").mean().head(2)
```

In [332...]

```
# Another example for year. Without kind = "period" or kind = "timestamp"  
# Notice how a whole date is given instead of the year.  
temp.resample("Y", kind = "timestamp").mean().head(2)
```

Out[332...]

LA NY

datetime

2013-12-31 17.174229 12.196153
2014-12-31 17.245616 11.370959

The good exemplar.

Be sure to save it to a variable.

Saving it automatically recreates the rest of the table faithfully, automatically.

In [342...]

```
# This is how it is done.  
# Notice that is just the year only that is shown.
```

```
# The interval defines what is shown.  
temp_n = temp.resample("Y", kind = "period").mean()
```

In [344...]

```
temp_n
```

Out[344...]

LA NY

datetime	LA	NY
2013	17.174229	12.196153
2014	17.245616	11.370959
2015	17.191530	11.795194
2016	18.330305	12.908470

Now to Make temp_n months with more datapoints

In [347...]

```
temp_n = temp.resample("M", kind = "period").mean()
```

In [349...]

```
# Still works because it shows month and years. Enough to distinguish themselves but not  
temp_n.head()
```

Out[349...]

LA NY

datetime	LA	NY
2013-01	11.596237	1.129570
2013-02	12.587202	0.617857
2013-03	15.069946	3.719220
2013-04	16.487361	10.699306
2013-05	19.005780	15.824328

In [350...]

```
temp_n.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
PeriodIndex: 48 entries, 2013-01 to 2016-12  
Freq: M  
Data columns (total 2 columns):  
 #   Column  Non-Null Count  Dtype     
---  --     
 0   LA      48 non-null    float64  
 1   NY      48 non-null    float64  
dtypes: float64(2)  
memory usage: 1.1 KB
```

In [351...]

```
temp_n.index
```

Out[351...]

```
PeriodIndex(['2013-01', '2013-02', '2013-03', '2013-04', '2013-05', '2013-06',  
            '2013-07', '2013-08', '2013-09', '2013-10', '2013-11', '2013-12',  
            '2014-01', '2014-02', '2014-03', '2014-04', '2014-05', '2014-06',  
            '2014-07', '2014-08', '2014-09', '2014-10', '2014-11', '2014-12',  
            '2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06',  
            '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12',  
            '2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06'],
```

```
'2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12'],
dtype='period[M]', name='datetime')
```

In [352...]

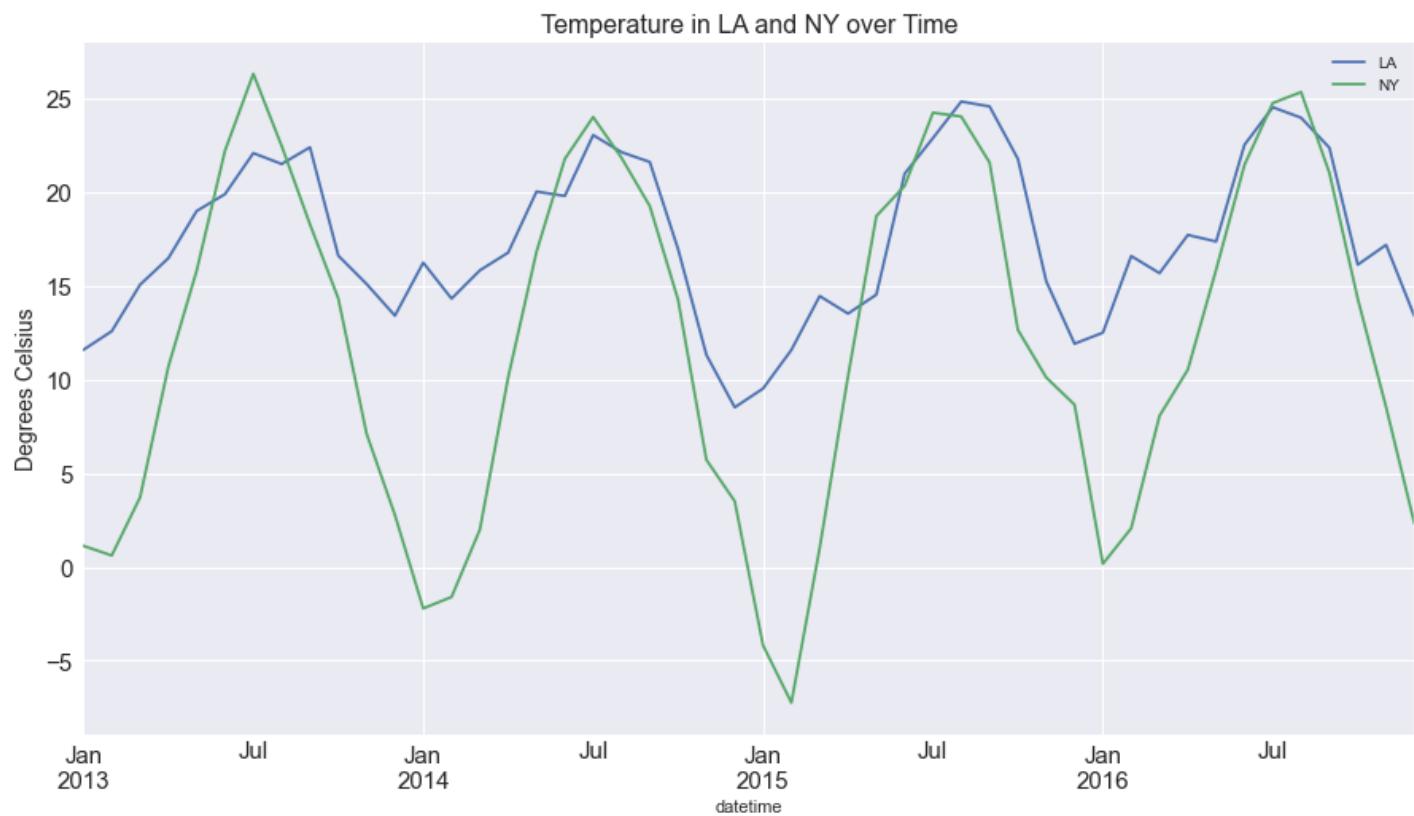
```
# Indexes can be selected individually.
temp_n.index[0]
```

Out[352...]

Chart with less datapoints.

In [360...]

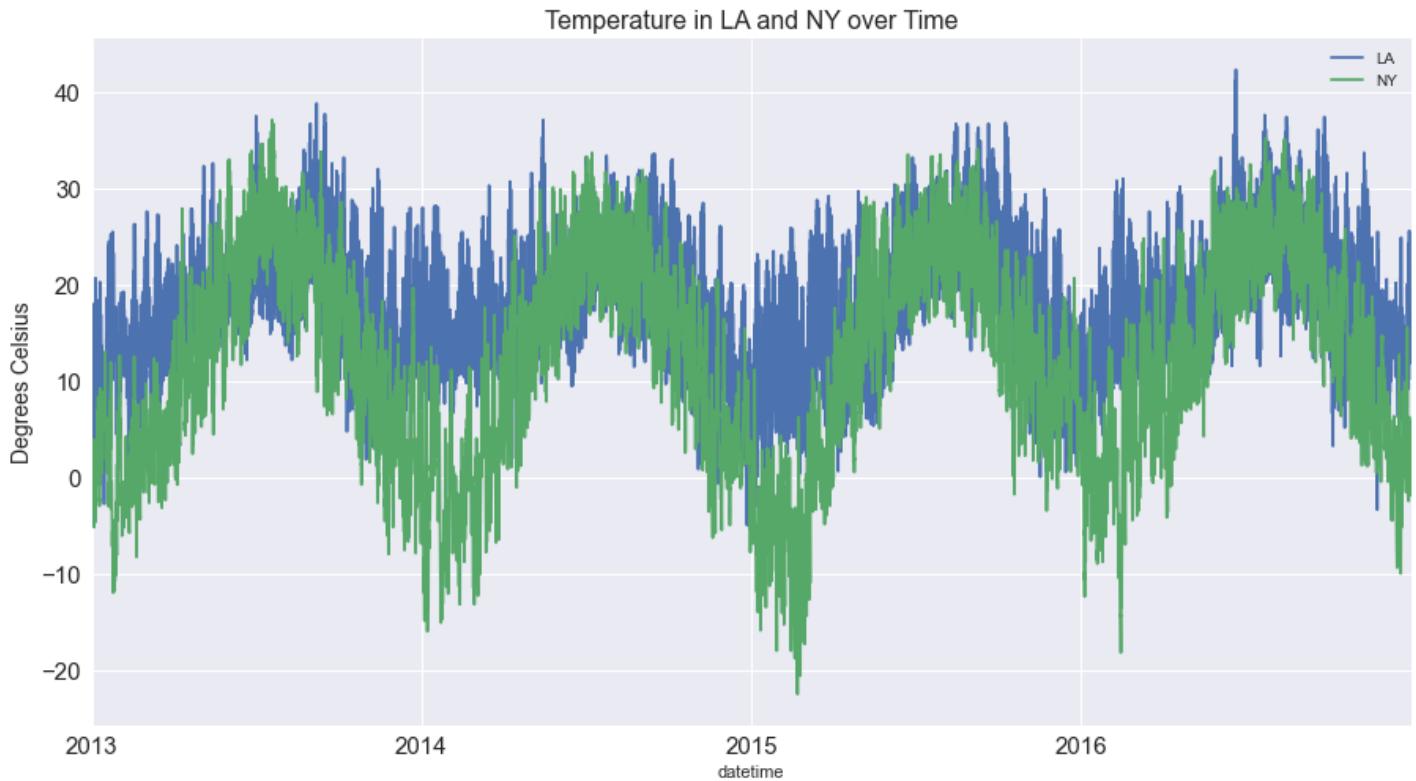
```
temp_n.plot(figsize = (15, 8), fontsize = 15)
plt.title("Temperature in LA and NY over Time", fontsize = 16)
plt.ylabel("Degrees Celsius", fontsize = 14)
plt.show()
```



The original chart.

In [361...]

```
temp.plot(figsize = (15, 8), fontsize = 15)
plt.title("Temperature in LA and NY over Time", fontsize = 16)
plt.ylabel("Degrees Celsius", fontsize = 14)
plt.show()
```



New Section

The Period Index Object

Stick with Datetime objects

In [363...]

```
import pandas as pd
```

In [364...]

```
temp = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\temp.csv")
```

In [365...]

```
temp.head()
```

Out[365...]

LA NY

datetime	LA	NY
2013-01-01 00:00:00	11.7	-1.1
2013-01-01 01:00:00	10.7	-1.7
2013-01-01 02:00:00	9.9	-2.0
2013-01-01 03:00:00	9.3	-2.1
2013-01-01 04:00:00	8.8	-2.3

In [366...]

```
temp.tail()
```

Out[366...]

LA NY

datetime

LA NY

datetime

	LA	NY
2016-12-31 19:00:00	13.5	4.6
2016-12-31 20:00:00	13.2	5.7
2016-12-31 21:00:00	12.8	5.8
2016-12-31 22:00:00	12.3	5.7
2016-12-31 23:00:00	11.9	5.5

In [367...]

```
temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 35064 entries, 2013-01-01 00:00:00 to 2016-12-31 23:00:00
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  --  
 0   LA       35062 non-null   float64 
 1   NY       35064 non-null   float64 
dtypes: float64(2)
memory usage: 821.8 KB
```

In [377...]

```
temp_n = temp.resample("M", kind = "period").mean()
temp_n.head(12)
```

Out[377...]

LA NY

	LA	NY
2013-01	11.596237	1.129570
2013-02	12.587202	0.617857
2013-03	15.069946	3.719220
2013-04	16.487361	10.699306
2013-05	19.005780	15.824328
2013-06	19.905417	22.225694
2013-07	22.093952	26.329704
2013-08	21.513172	22.480376
2013-09	22.404861	18.291806
2013-10	16.620430	14.335215
2013-11	15.107917	7.111944
2013-12	13.416935	2.776210

In [378...]

```
temp_n.index
```

Out[378...]

```
PeriodIndex(['2013-01', '2013-02', '2013-03', '2013-04', '2013-05', '2013-06',
 '2013-07', '2013-08', '2013-09', '2013-10', '2013-11', '2013-12',
 '2014-01', '2014-02', '2014-03', '2014-04', '2014-05', '2014-06',
 '2014-07', '2014-08', '2014-09', '2014-10', '2014-11', '2014-12',
 '2015-01', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06',
 '2015-07', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12'],
```

```
'2016-01', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06',
'2016-07', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12'],
dtype='period[M]', name='datetime')
```

```
In [379... temp_n.loc["2013-01"]]
```

```
Out[379... LA      11.596237
          NY      1.129570
          Name: 2013-01, dtype: float64
```

```
In [380... temp_n.loc["2013-05":"2013-08"]]
```

```
Out[380...          LA          NY
          datetime
2013-05  19.005780  15.824328
2013-06  19.905417  22.225694
2013-07  22.093952  26.329704
2013-08  21.513172  22.480376
```

```
In [381... temp_n.loc["2013"]]
```

```
Out[381...          LA          NY
          datetime
2013-01  11.596237  1.129570
2013-02  12.587202  0.617857
2013-03  15.069946  3.719220
2013-04  16.487361  10.699306
2013-05  19.005780  15.824328
2013-06  19.905417  22.225694
2013-07  22.093952  26.329704
2013-08  21.513172  22.480376
2013-09  22.404861  18.291806
2013-10  16.620430  14.335215
2013-11  15.107917  7.111944
2013-12  13.416935  2.776210
```

```
In [382... temp_n.to_timestamp(how = "start")]
```

```
Out[382...          LA          NY
          datetime
2013-01-01  11.596237  1.129570
2013-02-01  12.587202  0.617857
```

LA NY

datetime	LA	NY
2013-03-01	15.069946	3.719220
2013-04-01	16.487361	10.699306
2013-05-01	19.005780	15.824328
2013-06-01	19.905417	22.225694
2013-07-01	22.093952	26.329704
2013-08-01	21.513172	22.480376
2013-09-01	22.404861	18.291806
2013-10-01	16.620430	14.335215
2013-11-01	15.107917	7.111944
2013-12-01	13.416935	2.776210
2014-01-01	16.247715	-2.210349
2014-02-01	14.326637	-1.596280
2014-03-01	15.836156	1.994758
2014-04-01	16.783472	10.128611
2014-05-01	20.041398	16.829167
2014-06-01	19.810556	21.784861
2014-07-01	23.056183	24.019624
2014-08-01	22.146102	21.848387
2014-09-01	21.620278	19.267222
2014-10-01	16.968280	14.265860
2014-11-01	11.322083	5.719167
2014-12-01	8.523925	3.513844
2015-01-01	9.527016	-4.179301
2015-02-01	11.587649	-7.239732
2015-03-01	14.465995	1.020161
2015-04-01	13.527917	10.180417
2015-05-01	14.540054	18.731586
2015-06-01	20.995000	20.347500
2015-07-01	22.899059	24.252151
2015-08-01	24.846909	24.046774
2015-09-01	24.586250	21.589861
2015-10-01	21.785753	12.656452
2015-11-01	15.255278	10.117500
2015-12-01	11.919758	8.662500
2016-01-01	12.509274	0.168952

datetime	LA	NY
2016-02-01	16.600431	2.069971
2016-03-01	15.686425	8.070430
2016-04-01	17.726111	10.535556
2016-05-01	17.375403	15.874462
2016-06-01	22.536111	21.462639
2016-07-01	24.541532	24.767608
2016-08-01	23.983199	25.351882
2016-09-01	22.379306	21.032778
2016-10-01	16.137903	14.321640
2016-11-01	17.196111	8.539722
2016-12-01	13.390457	2.327285

New Section

Advanced Indexing with reindex()

In [383...]

```
import pandas as pd
```

In [384...]

```
temp = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\temp.csv")
```

Summary

There are several ways to fill in missing values using reindex()

In column.reindex(method = "nearest") also "bfill" or "ffill" or "nearest"

Revisit Advanced Indexing with reindex

Documentation:

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.reindex.html>

In [385...]

```
temp.head()
```

Out[385...]

LA NY

datetime	LA	NY
2013-01-01 00:00:00	11.7	-1.1
2013-01-01 01:00:00	10.7	-1.7
2013-01-01 02:00:00	9.9	-2.0

LA NY

datetime

datetime	LA	NY
2013-01-01 03:00:00	9.3	-2.1
2013-01-01 04:00:00	8.8	-2.3

In [386...]

```
temp.tail()
```

Out[386...]

LA NY

datetime

datetime	LA	NY
2016-12-31 19:00:00	13.5	4.6
2016-12-31 20:00:00	13.2	5.7
2016-12-31 21:00:00	12.8	5.8
2016-12-31 22:00:00	12.3	5.7
2016-12-31 23:00:00	11.9	5.5

In [395...]

```
# Convert to daily from hourly
temp_d = temp.resample("D").mean()
temp_d
```

Out[395...]

LA NY

datetime

datetime	LA	NY
2013-01-01	8.858333	-0.404167
2013-01-02	9.283333	3.208333
2013-01-03	10.304167	-2.425000
2013-01-04	11.512500	-2.070833
2013-01-05	11.083333	0.816667
...
2016-12-27	12.154167	10.579167
2016-12-28	14.433333	4.016667
2016-12-29	16.045833	1.312500
2016-12-30	15.933333	2.204167
2016-12-31	13.275000	1.204167

1461 rows × 2 columns

In [396...]

```
# Go back in time using freq = pd.DateOffset(years = 1) inside the pd.date_range()
birthd = pd.date_range(end = "2016-12-24", periods = 3, freq = pd.DateOffset(years = 1))
birthd
```

Out[396...]

```
DatetimeIndex(['2014-12-24', '2015-12-24', '2016-12-24'], dtype='datetime64[ns]', freq='<DateOffset: years=1>')
```

In [397...]

```
temp_d.loc[birthd]
```

Out[397...]

LA **NY**

2014-12-24	10.712500	8.045833
2015-12-24	10.716667	17.462500
2016-12-24	11.820833	4.045833

In [401...]

```
temp_d.reindex(birthd, method = "bfill")
```

Out[401...]

LA **NY**

2014-12-24	10.712500	8.045833
2015-12-24	10.716667	17.462500
2016-12-24	11.820833	4.045833

In [399...]

```
temp_d.head()
```

Out[399...]

LA **NY**

datetime		
2013-01-01	8.858333	-0.404167
2013-01-02	9.283333	3.208333
2013-01-03	10.304167	-2.425000
2013-01-04	11.512500	-2.070833
2013-01-05	11.083333	0.816667

In [393...]

```
temp_d.tail()
```

Out[393...]

LA **NY**

datetime		
2016-12-27	12.154167	10.579167
2016-12-28	14.433333	4.016667
2016-12-29	16.045833	1.312500
2016-12-30	15.933333	2.204167
2016-12-31	13.275000	1.204167

In []:

In []:

In []:

Removing Columns

In [57]:	<pre>import pandas as pd</pre>																																																												
In [58]:	<pre>summer = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\summer.csv")</pre>																																																												
In [59]:	<pre>summer.head()</pre>																																																												
Out[59]:	<table><thead><tr><th></th><th>Year</th><th>City</th><th>Sport</th><th>Discipline</th><th>Athlete</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr></thead><tbody><tr><td>0</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HAJOS, Alfred</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr><tr><td>1</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>HERSCHMANN, Otto</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr><tr><td>2</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>DRIVAS, Dimitrios</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr><tr><td>3</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>MALOKINIS, Ioannis</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr><tr><td>4</td><td>1896</td><td>Athens</td><td>Aquatics</td><td>Swimming</td><td>CHASAPIS, Spiridon</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr></tbody></table>		Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal	0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold	1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver	2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze	3	1896	Athens	Aquatics	Swimming	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold	4	1896	Athens	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver
	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal																																																				
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold																																																				
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver																																																				
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze																																																				
3	1896	Athens	Aquatics	Swimming	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold																																																				
4	1896	Athens	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver																																																				

Use the `.drop(columns = "characteristic_to_delete")` to get rid of a column.

In [60]:	<pre>summer.drop(columns = "Sport")</pre>																																																																																																												
Out[60]:	<table><thead><tr><th></th><th>Year</th><th>City</th><th>Discipline</th><th>Athlete</th><th>Country</th><th>Gender</th><th>Event</th><th>Medal</th></tr></thead><tbody><tr><td>0</td><td>1896</td><td>Athens</td><td>Swimming</td><td>HAJOS, Alfred</td><td>HUN</td><td>Men</td><td>100M Freestyle</td><td>Gold</td></tr><tr><td>1</td><td>1896</td><td>Athens</td><td>Swimming</td><td>HERSCHMANN, Otto</td><td>AUT</td><td>Men</td><td>100M Freestyle</td><td>Silver</td></tr><tr><td>2</td><td>1896</td><td>Athens</td><td>Swimming</td><td>DRIVAS, Dimitrios</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Bronze</td></tr><tr><td>3</td><td>1896</td><td>Athens</td><td>Swimming</td><td>MALOKINIS, Ioannis</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Gold</td></tr><tr><td>4</td><td>1896</td><td>Athens</td><td>Swimming</td><td>CHASAPIS, Spiridon</td><td>GRE</td><td>Men</td><td>100M Freestyle For Sailors</td><td>Silver</td></tr><tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr><tr><td>31160</td><td>2012</td><td>London</td><td>Wrestling Freestyle</td><td>JANIKOWSKI, Damian</td><td>POL</td><td>Men</td><td>Wg 84 KG</td><td>Bronze</td></tr><tr><td>31161</td><td>2012</td><td>London</td><td>Wrestling Freestyle</td><td>REZAEI, Ghasem Gholamreza</td><td>IRI</td><td>Men</td><td>Wg 96 KG</td><td>Gold</td></tr><tr><td>31162</td><td>2012</td><td>London</td><td>Wrestling Freestyle</td><td>TOTROV, Rustam</td><td>RUS</td><td>Men</td><td>Wg 96 KG</td><td>Silver</td></tr><tr><td>31163</td><td>2012</td><td>London</td><td>Wrestling Freestyle</td><td>ALEKSANYAN, Artur</td><td>ARM</td><td>Men</td><td>Wg 96 KG</td><td>Bronze</td></tr><tr><td>31164</td><td>2012</td><td>London</td><td>Wrestling Freestyle</td><td>LIDBERG, Jimmy</td><td>SWE</td><td>Men</td><td>Wg 96 KG</td><td>Bronze</td></tr></tbody></table>		Year	City	Discipline	Athlete	Country	Gender	Event	Medal	0	1896	Athens	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold	1	1896	Athens	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver	2	1896	Athens	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze	3	1896	Athens	Swimming	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold	4	1896	Athens	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver	31160	2012	London	Wrestling Freestyle	JANIKOWSKI, Damian	POL	Men	Wg 84 KG	Bronze	31161	2012	London	Wrestling Freestyle	REZAEI, Ghasem Gholamreza	IRI	Men	Wg 96 KG	Gold	31162	2012	London	Wrestling Freestyle	TOTROV, Rustam	RUS	Men	Wg 96 KG	Silver	31163	2012	London	Wrestling Freestyle	ALEKSANYAN, Artur	ARM	Men	Wg 96 KG	Bronze	31164	2012	London	Wrestling Freestyle	LIDBERG, Jimmy	SWE	Men	Wg 96 KG	Bronze
	Year	City	Discipline	Athlete	Country	Gender	Event	Medal																																																																																																					
0	1896	Athens	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold																																																																																																					
1	1896	Athens	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver																																																																																																					
2	1896	Athens	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze																																																																																																					
3	1896	Athens	Swimming	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold																																																																																																					
4	1896	Athens	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver																																																																																																					
...																																																																																																					
31160	2012	London	Wrestling Freestyle	JANIKOWSKI, Damian	POL	Men	Wg 84 KG	Bronze																																																																																																					
31161	2012	London	Wrestling Freestyle	REZAEI, Ghasem Gholamreza	IRI	Men	Wg 96 KG	Gold																																																																																																					
31162	2012	London	Wrestling Freestyle	TOTROV, Rustam	RUS	Men	Wg 96 KG	Silver																																																																																																					
31163	2012	London	Wrestling Freestyle	ALEKSANYAN, Artur	ARM	Men	Wg 96 KG	Bronze																																																																																																					
31164	2012	London	Wrestling Freestyle	LIDBERG, Jimmy	SWE	Men	Wg 96 KG	Bronze																																																																																																					

31165 rows × 8 columns

Notice that by default the `.drop()` method does not influence the main dataset by default. Change the `inplace = True` in order to change the main data set.

In [61]:

```
summer.head()
```

Out[61]:

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze
3	1896	Athens	Aquatics	Swimming	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold
4	1896	Athens	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver

Example of getting rid of the characteristic permanently.

Remember hold shift + Tab + Tab will pull up all of the parameters that you can change.

In [62]:

```
## To drop multiple values use a list in the column section.  
summer.drop(columns = ["Sport", "Discipline"], inplace = True)
```

In [63]:

```
summer.head()
```

Out[63]:

	Year	City	Athlete	Country	Gender	Event	Medal
0	1896	Athens	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold
1	1896	Athens	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver
2	1896	Athens	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze
3	1896	Athens	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold
4	1896	Athens	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver

Drop a column by using `.drop(labels = "characteristic_name", axis = "columns")`

In [64]:

```
# Set the inplace parameter to True to make it permanent.  
summer.drop(labels = "Event", axis = "columns", inplace = True)  
summer.head()
```

Out[64]:

	Year	City	Athlete	Country	Gender	Medal
0	1896	Athens	HAJOS, Alfred	HUN	Men	Gold
1	1896	Athens	HERSCHMANN, Otto	AUT	Men	Silver
2	1896	Athens	DRIVAS, Dimitrios	GRE	Men	Bronze
3	1896	Athens	MALOKINIS, Ioannis	GRE	Men	Gold
4	1896	Athens	CHASAPIS, Spiridon	GRE	Men	Silver

Another way to delete a column: `del dataset_name["column_name"]`

The `.drop()` method is superior as the `del` way is less malleable.

In []:

```
# This cell is not run as Event has already been deleted.  
del summer["Event"]
```

Reimport original data.

In [71]:

```
summer = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\summer.csv")
```

In [72]:

```
### Original Data  
summer.head()
```

Out[72]:

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze
3	1896	Athens	Aquatics	Swimming	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold
4	1896	Athens	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver

Using the `.loc[:,["List_of_selected_columns"]]` we can choose which columns will remain.

`data_set.loc[:,["List_of_selected_columns"]]` is a great way to select data that is actually pertinent to the purpose of the analysis. It is superior to deleting columns, if you need a few columns out of a dataset with a hundred columns. If there are a few columns the drop method will work better.

In [74]:

```
summer = summer.loc[:,["Year", "City", "Athlete", "Country", "Gender", "Medal"]]
```

In [75]:

```
summer.head()
```

Out[75]:

	Year	City	Athlete	Country	Gender	Medal
0	1896	Athens	HAJOS, Alfred	HUN	Men	Gold
1	1896	Athens	HERSCHMANN, Otto	AUT	Men	Silver
2	1896	Athens	DRIVAS, Dimitrios	GRE	Men	Bronze
3	1896	Athens	MALOKINIS, Ioannis	GRE	Men	Gold
4	1896	Athens	CHASAPIS, Spiridon	GRE	Men	Silver

Removing Rows with pandas

In [86]:

```
# This time import with athlete column as an index.
```

```
summer = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\summer.csv")
```

```
In [80]: summer.head(10)
```

Out[80]:

	Year	City	Sport	Discipline	Country	Gender	Event	Medal
Athlete								
HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold
HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver
DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze
MALOKINIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold
CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver
CHOROPHAS, Efstatios	1896	Athens	Aquatics	Swimming	GRE	Men	1200M Freestyle	Bronze
HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	1200M Freestyle	Gold
ANDREOU, Joannis	1896	Athens	Aquatics	Swimming	GRE	Men	1200M Freestyle	Silver
CHOROPHAS, Efstatios	1896	Athens	Aquatics	Swimming	GRE	Men	400M Freestyle	Bronze
NEUMANN, Paul	1896	Athens	Aquatics	Swimming	AUT	Men	400M Freestyle	Gold

Setting an index is very important. It allows you to be able to get rid of indexes that are set to a certain value.

```
.drop(index = "name_of_index_entry")
```

In [87]:

```
# Observe what happens when the first name of the index is deleted.
summer.drop(index = "HAJOS, Alfred", inplace = True)
summer.head(5)
```

Out[87]:

	Year	City	Sport	Discipline	Country	Gender	Event	Medal
Athlete								
HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver
DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze
MALOKINIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold
CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver
CHOROPHAS, Efstatios	1896	Athens	Aquatics	Swimming	GRE	Men	1200M Freestyle	Bronze

Use a list for the index in order to get rid of several names.

In [89]:

```
summer.drop(index = ["HERSCHMANN, Otto", "DRIVAS, Dimitrios"], inplace = True)
summer.head(5)
```

Out[89]:

	Year	City	Sport	Discipline	Country	Gender	Event	Medal
Athlete								
MALOKINIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold
CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver
CHOROPHAS, Efstatios	1896	Athens	Aquatics	Swimming	GRE	Men	1200M Freestyle	Bronze

Year	City	Sport	Discipline	Country	Gender	Event	Medal
Athlete							
1896	Athens	Aquatics	Swimming	GRE	Men	1200M Freestyle	Silver
1896	Athens	Aquatics	Swimming	GRE	Men	400M Freestyle	Bronze

In []:

```
# Another way to get rid of an index.
summer.drop(labels = "DRIVAS, Dimitrios", axis = 0, inplace = True)
# Note that axis = index is also a valid input.
```

Use the `dataset.loc[dataset.Year == Value]` to hone in on specific time periods that needs to be focused on.

In [97]:

```
summer = summer.loc[summer.Year == 1996]
```

In [98]:

```
summer.head()
```

Out[98]:

Year	City	Sport	Discipline	Country	Gender	Event	Medal
Athlete							
1996	Atlanta	Aquatics	Diving	CHN	Men	10M Platform	Bronze
1996	Atlanta	Aquatics	Diving	RUS	Men	10M Platform	Gold
1996	Atlanta	Aquatics	Diving	GER	Men	10M Platform	Silver
1996	Atlanta	Aquatics	Diving	USA	Women	10M Platform	Bronze
1996	Atlanta	Aquatics	Diving	CHN	Women	10M Platform	Gold

Quickly make a chart by filtering via a subcategory.

In [100]:

```
summer = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\summer.csv")
```

In [104]:

```
summer = summer.loc[summer.City == "Beijing"]
```

In [105]:

```
summer.head()
```

Out[105]:

Year	City	Sport	Discipline	Country	Gender	Event	Medal
Athlete							
2008	Beijing	Aquatics	Diving	RUS	Men	10M Platform	Bronze
2008	Beijing	Aquatics	Diving	AUS	Men	10M Platform	Gold
2008	Beijing	Aquatics	Diving	CHN	Men	10M Platform	Silver
2008	Beijing	Aquatics	Diving	CHN	Women	10M Platform	Bronze
2008	Beijing	Aquatics	Diving	CHN	Women	10M Platform	Gold

Can filter a characteristic and only show other characteristics

specified.

In [108...]

```
summer = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\summer.csv")
```

In [115...]

```
summer.loc[summer.Medal == "Gold", "Sport": "Medal"]
```

Out[115...]

	Sport	Discipline	Country	Gender	Event	Medal
Athlete						
HAJOS, Alfred	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold
MALOKINIS, Ioannis	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold
HAJOS, Alfred	Aquatics	Swimming	HUN	Men	1200M Freestyle	Gold
NEUMANN, Paul	Aquatics	Swimming	AUT	Men	400M Freestyle	Gold
BURKE, Thomas	Athletics	Athletics	USA	Men	100M	Gold
...
NOROOZI, Omid Haji	Wrestling	Wrestling Freestyle	IRI	Men	Wg 60 KG	Gold
KIM, Hyeonwoo	Wrestling	Wrestling Freestyle	KOR	Men	Wg 66 KG	Gold
VLASOV, Roman	Wrestling	Wrestling Freestyle	RUS	Men	Wg 74 KG	Gold
KHUGAEV, Alan	Wrestling	Wrestling Freestyle	RUS	Men	Wg 84 KG	Gold
REZAEI, Ghasem Gholamreza	Wrestling	Wrestling Freestyle	IRI	Men	Wg 96 KG	Gold

10486 rows × 6 columns

In [111...]

```
summer = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\summer.csv")
```

In [117...]

```
# Or a list Can be used to include more values.  
summer.loc[summer.Medal == "Gold", ["Year", "Country", "Sport", "Medal"]]
```

Out[117...]

	Year	Country	Sport	Medal
Athlete				
HAJOS, Alfred	1896	HUN	Aquatics	Gold
MALOKINIS, Ioannis	1896	GRE	Aquatics	Gold
HAJOS, Alfred	1896	HUN	Aquatics	Gold
NEUMANN, Paul	1896	AUT	Aquatics	Gold
BURKE, Thomas	1896	USA	Athletics	Gold
...
NOROOZI, Omid Haji	2012	IRI	Wrestling	Gold
KIM, Hyeonwoo	2012	KOR	Wrestling	Gold
VLASOV, Roman	2012	RUS	Wrestling	Gold
KHUGAEV, Alan	2012	RUS	Wrestling	Gold
REZAEI, Ghasem Gholamreza	2012	IRI	Wrestling	Gold

10486 rows × 4 columns

Dropping Certain Characteristic values (shown later)

If a characteristic/column entry equals a certain value it can be dropped automatically to filter out the data. For example if there is a column named fruits, the data scientist can have any entry that is named "apple" able to be dropped.

First filtering will be observed for data using masks.

Here is an example of filtering values that meet certain criteria.

For example a chart can be made that fit one or two different criteria.

To filter out using two criteria a mask characteristic must be used to work with `.loc`

Symbols such as "|", "&" - represents or and respectively.

In [121...]

```
mask1 = summer.Year == 1996
mask2 = summer.Sport == "Aquatics"
```

In []:

```
# NOTE: This code will throw an error. Masks need to be used to avoid errors.
# Masks are filters to data that can make sure certain characteristics equal certain values
summer.loc[(summer.Year == 1996 | summer.Sport == "Aquatics")]
```

In [126...]

```
# Takes place in the year 1996 or is an aquatic game, it could also be both.
# Note the masks are out into a par
summer.loc[(mask1 | mask2)]
```

Out[126...]

	Year	City	Sport	Discipline	Country	Gender	Event	Medal
	Athlete							
HAJOS, Alfred	1896	Athens	Aquatics	Swimming	HUN	Men	100M Freestyle	Gold
HERSCHMANN, Otto	1896	Athens	Aquatics	Swimming	AUT	Men	100M Freestyle	Silver
DRIVAS, Dimitrios	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Bronze
MALOKINIS, Ioannis	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Gold
CHASAPIS, Spiridon	1896	Athens	Aquatics	Swimming	GRE	Men	100M Freestyle For Sailors	Silver
...
RIPPON, Melissa	2012	London	Aquatics	Water Polo	AUS	Women	Water Polo	Bronze
SMITH, Sophie	2012	London	Aquatics	Water Polo	AUS	Women	Water Polo	Bronze
SOUTHERN, Ash	2012	London	Aquatics	Water Polo	AUS	Women	Water Polo	Bronze
WEBSTER, Rowie	2012	London	Aquatics	Water Polo	AUS	Women	Water Polo	Bronze
ZAGAME, Nicola	2012	London	Aquatics	Water Polo	AUS	Women	Water Polo	Bronze

5767 rows × 8 columns

In [128...]

```
#This is a chart that has two characteristics.
summer.loc[(mask1 & mask2)]
```

Out[128...]

Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal
XIAO, Hailiang	1996	Atlanta	Aquatics	Diving	CHN	Men	10M Platform	Bronze
SAUTIN, Dmitry	1996	Atlanta	Aquatics	Diving	RUS	Men	10M Platform	Gold
HEMPEL, Jan	1996	Atlanta	Aquatics	Diving	GER	Men	10M Platform	Silver
CLARK, Mary Ellen	1996	Atlanta	Aquatics	Diving	USA	Women	10M Platform	Bronze
FU, Mingxia	1996	Atlanta	Aquatics	Diving	CHN	Women	10M Platform	Gold
...
SKOLNEKOVIC, Sinisa	1996	Atlanta	Aquatics	Water polo	CRO	Men	Water Polo	Silver
STRITOF, Ratko	1996	Atlanta	Aquatics	Water polo	CRO	Men	Water Polo	Silver
VEGAR, Tino	1996	Atlanta	Aquatics	Water polo	CRO	Men	Water Polo	Silver
VRBICIC, Renato	1996	Atlanta	Aquatics	Water polo	CRO	Men	Water Polo	Silver
VRDOLJAK, Zdeslav	1996	Atlanta	Aquatics	Water polo	CRO	Men	Water Polo	Silver

262 rows × 8 columns

Exclusion this is done by using the " ~ " symbol which tells the program to exclude certain values.

In [130...]

```
mask1 = summer.Year == 1996
mask2 = summer.Sport == "Aquatics"
```

In [137...]

```
# Count of 1996 before mask is applied.
(summer.Year == 1996).value_counts()
```

Out[137...]

```
False    29306
True     1859
Name: Year, dtype: int64
```

In [140...]

```
#Only show values that are neither aquatics nor in the year 1996.
summer = summer.loc[~(mask1 | mask2)]
```

In [141...]

```
summer.head()
```

Out[141...]

Athlete	Year	City	Sport	Discipline	Country	Gender	Event	Medal
LANE, Francis	1896	Athens	Athletics	Athletics	USA	Men	100M	Bronze
SZOKOLYI, Alajos	1896	Athens	Athletics	Athletics	HUN	Men	100M	Bronze
BURKE, Thomas	1896	Athens	Athletics	Athletics	USA	Men	100M	Gold

Year	City	Sport	Discipline	Country	Gender	Event	Medal	
Athlete								
HOFMANN, Fritz	1896	Athens	Athletics	Athletics	GER	Men	100M	Silver
CURTIS, Thomas	1896	Athens	Athletics	Athletics	USA	Men	110M Hurdles	Gold

Double checking to see if the items were removed.

See how many items were removed from a list by using `.values_counts()`

This only works after the mask has been applied.

```
In [147...]: (summer.Year == 1996).value_counts()
```

```
Out[147...]: False    25398
Name: Year, dtype: int64
```

```
In [148...]: # Alternate method.
1996 in summer.Year.values
```

```
Out[148...]: False
```

Use the `.isin` method for a characteristic to see if it appears in the column followed with a `.any()` for a boolean.

```
dataset.Column_name(["Characterisitc_entry"]).any()
```

```
In [151...]: summer.Sport.isin(["Aquatics"]).any()
```

```
Out[151...]: False
```

New Section

Adding New Columns to a DataFrame

```
In [155...]: import pandas as pd
```

```
In [158...]: titanic = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

```
In [159...]: titanic.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN

Add a column simply by typing `data_set["new_column_name"] = value`

```
data_set["new_column_name"] = value
```

```
In [164... # An string can be passed in instead of an integer.
titanic["Zeroes"] = 0
```

```
In [165... titanic.head()
```

```
Out[165...   survived  pclass  sex  age  sibsp  parch  fare  embarked  deck  Zeroes
  0          0      3  male  22.0      1      0  7.2500      S  NaN      0
  1          1      1  female  38.0      1      0  71.2833      C  C      0
  2          1      3  female  26.0      0      0  7.9250      S  NaN      0
  3          1      1  female  35.0      1      0  53.1000      S  C      0
  4          0      3  male  35.0      0      0  8.0500      S  NaN      0
```

```
In [167... # Does not create a column, but this can create an attributte.
titanic.Ones = 1
```

```
In [169... # Notice how a Ones column does not appear on the chart.
titanic.head()
```

```
Out[169...   survived  pclass  sex  age  sibsp  parch  fare  embarked  deck  Zeroes
  0          0      3  male  22.0      1      0  7.2500      S  NaN      0
  1          1      1  female  38.0      1      0  71.2833      C  C      0
  2          1      3  female  26.0      0      0  7.9250      S  NaN      0
  3          1      1  female  35.0      1      0  53.1000      S  C      0
  4          0      3  male  35.0      0      0  8.0500      S  NaN      0
```

```
In [171... # However it is inside of the system.
titanic.Ones
```

```
Out[171... 1
```

New Section

Data Preparation and Feature Creation

Arithmetic Operations

In [173...]

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

In [183...]

```
titanic = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

In [186...]

```
# Notice how index 5 age is missing. With nan which is can be filled in with .fillna()
titanic.head(7)
```

Out[186...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN
5	0	3	male	NaN	0	0	8.4583	Q	NaN
6	0	1	male	54.0	0	0	51.8625	S	E

In [176...]

```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
---  --  
 0   survived    891 non-null    int64  
 1   pclass      891 non-null    int64  
 2   sex         891 non-null    object  
 3   age         714 non-null    float64 
 4   sibsp       891 non-null    int64  
 5   parch       891 non-null    int64  
 6   fare         891 non-null    float64 
 7   embarked    889 non-null    object  
 8   deck         203 non-null    object  
dtypes: float64(2), int64(4), object(3)
memory usage: 62.8+ KB
```

Use `data.column.fillna(data.column.mean(), inplace = True)` to fill in the empty values with the mean age.

`.fillna` is a very important tool to fill in the blank values.

In [188...]

```
titanic.age.fillna(titanic.age.mean(), inplace = True)
```

In [189...]

```
titanic.age.mean()
```

```
Out[189... 29.699117647058763
```

```
In [190... # Notice how index 5's age NaN is filled in with the mean age.
```

```
titanic.head(10)
```

```
Out[190... 0 1 2 3 4 5 6 7 8 9
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.000000	1	0	7.2500	S	NaN
1	1	1	female	38.000000	1	0	71.2833	C	C
2	1	3	female	26.000000	0	0	7.9250	S	NaN
3	1	1	female	35.000000	1	0	53.1000	S	C
4	0	3	male	35.000000	0	0	8.0500	S	NaN
5	0	3	male	29.699118	0	0	8.4583	Q	NaN
6	0	1	male	54.000000	0	0	51.8625	S	E
7	0	3	male	2.000000	3	1	21.0750	S	NaN
8	1	3	female	27.000000	0	2	11.1333	S	NaN
9	1	2	female	14.000000	1	0	30.0708	C	NaN

Example: the missing value of the deck can be changed with the .fillna() command also.

```
In [193... titanic.deck.fillna("none", inplace = True)
```

```
In [197... # Notice how the NaN values have been replaced with none.
```

```
titanic
```

```
Out[197... 0 1 2 3 4 ... 886 887 888 889 890
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.000000	1	0	7.2500	S	none
1	1	1	female	38.000000	1	0	71.2833	C	C
2	1	3	female	26.000000	0	0	7.9250	S	none
3	1	1	female	35.000000	1	0	53.1000	S	C
4	0	3	male	35.000000	0	0	8.0500	S	none
...
886	0	2	male	27.000000	0	0	13.0000	S	none
887	1	1	female	19.000000	0	0	30.0000	S	B
888	0	3	female	29.699118	1	2	23.4500	S	none
889	1	1	male	26.000000	0	0	30.0000	C	C
890	0	3	male	32.000000	0	0	7.7500	Q	none

891 rows × 9 columns

Changing the value back to the .fillna() back to NaN

You cannot use .fillna() to revert a previous NaN value back.

In [198...]

```
titanic.deck.fillna("NaN")
```

Out[198...]

```
0      none
1        C
2      none
3        C
4      none
...
886    none
887      B
888    none
889      C
890    none
Name: deck, Length: 891, dtype: object
```

In [199...]

```
titanic.head()
```

Out[199...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	none
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	S	none
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	none

Convering values back to NaN

Note this can only be done when all of the values have been replaced with a singular value fillna.

Insted you have to use the .replace() method.

A condition is needed. If a boolean value is true then the new value will replace the old value.

See below:

DataFrame.replace(to_replace=None, value=NoDefault.no_default, inplace=False, limit=None, regex=False, method=NoDefault.no_default)

In [205...]

```
# See more info:
# https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.replace.html#pandas.DataFrame.replace
titanic.deck.replace(to_replace = "none", value = "NaN", inplace = True)
```

In [208...]

```
titanic.head()
```

Out[208...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN

New section

Arithmetic Operations

Addition, subtraction, multiplication and division.

Adding two columns.

Two columns can be added up by simply putting a + sign in between them.

In [215...]

```
titanic = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

In [221...]

```
# You need to pass in a list for only displaying desireable columns.
titanic[["sibsp", "pclass"]]
```

Out[221...]

	sibsp	pclass
0	1	3
1	1	1
2	0	3
3	1	1
4	0	3
...
886	0	2
887	0	1
888	1	3
889	0	1
890	0	3

891 rows × 2 columns

Notice how the sum ends up in the second column.

In [235...]

```
# Here this table is adding the number of spouses and children that were onboard
(titanic.sibsp + titanic.pclass).head(10)
# Below is the same operation without the head.
# titanic.sibsp + titanic.pclass
```

Out[235...]

0	4
1	2

```
2    3
3    2
4    3
5    3
6    1
7    6
8    3
9    3
dtype: int64
```

Alternative way to add.

This method is considered superior.

Use the `data_set.column.add(second_characteristic)`

```
In [238...]: titanic.sibsp.add(titanic.parch)
```

```
Out[238...]: 0      1
1      1
2      0
3      1
4      0
...
886    0
887    0
888    3
889    0
890    0
Length: 891, dtype: int64
```

Add a new column with a sum.

```
In [240...]: titanic["No_of_Rel"] = titanic.sibsp.add(titanic.parch)
```

```
In [243...]: titanic
```

```
Out[243...]:   survived  pclass  sex    age  sibsp  parch    fare  embarked  deck  No_of_Rel
  0         0      3  male  22.0      1      0  7.2500        S   NaN        1
  1         1      1  female  38.0      1      0  71.2833       C     C        1
  2         1      3  female  26.0      0      0  7.9250        S   NaN        0
  3         1      1  female  35.0      1      0  53.1000       S     C        1
  4         0      3  male  35.0      0      0  8.0500        S   NaN        0
...
886        0      2  male  27.0      0      0  13.0000        S   NaN        0
887        1      1  female  19.0      0      0  30.0000       S     B        0
888        0      3  female   NaN      1      2  23.4500        S   NaN        3
889        1      1  male  26.0      0      0  30.0000       C     C        0
890        0      3  male  32.0      0      0  7.7500        Q   NaN        0
```

891 rows × 10 columns

Import sales data.

In [245...]

```
sales = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\sales.csv")
```

In [247...]

```
sales
```

Out[247...]

	Unnamed: 0	Mon	Tue	Wed	Thu	Fri
0	Steven	34	27	15	NaN	33
1	Mike	45	9	74	87.0	12
2	Andi	17	33	54	8.0	29
3	Paul	87	67	27	45.0	7

When you add a missing value by default the sum will be NaN.

In [248...]

```
sales.Mon + sales.Thu
```

Out[248...]

0	NaN
1	132.0
2	25.0
3	132.0

dtype: float64

Can make the NaN fill value equal to zero. This would make addition possible.

In [252...]

```
sales.Mon.add(sales.Thu, fill_value = 0)
```

Out[252...]

0	34.0
1	132.0
2	25.0
3	132.0

dtype: float64

In [260...]

```
sales["perc_Bonus"] = [0.12, 0.15, 0.10, 0.20]
```

In [259...]

```
sales
```

Out[259...]

	Unnamed: 0	Mon	Tue	Wed	Thu	Fri	per_Bonus	perc_Bonus
0	Steven	34	27	15	NaN	33	0.12	0.12
1	Mike	45	9	74	87.0	12	0.15	0.15
2	Andi	17	33	54	8.0	29	0.10	0.10
3	Paul	87	67	27	45.0	7	0.20	0.20

Multiplication

```
In [261... sales.Thu * sales.perc_Bonus
```

```
Out[261... 0      NaN
1      13.05
2      0.80
3      9.00
dtype: float64
```

.mul is how to do multiplication.

```
In [265... sales.Thu.mul(sales.perc_Bonus, fill_value = 0)
```

```
Out[265... 0      0.00
1      13.05
2      0.80
3      9.00
dtype: float64
```

```
In [266... sales.iloc[:, :-1].sum(axis = 1).mul(sales.perc_Bonus)
```

```
C:\Users\alonz\AppData\Local\Temp\ipykernel_27368\1359361647.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.
```

```
    sales.iloc[:, :-1].sum(axis = 1).mul(sales.perc_Bonus)
```

```
Out[266... 0      13.1088
1      34.0950
2      14.1200
3      46.6800
dtype: float64
```

```
In [268... sales["Bonus"] = sales.iloc[:, :-1].sum(axis = 1).mul(sales.perc_Bonus)
```

```
C:\Users\alonz\AppData\Local\Temp\ipykernel_27368\4000653404.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.
```

```
    sales["Bonus"] = sales.iloc[:, :-1].sum(axis = 1).mul(sales.perc_Bonus)
```

```
In [269... sales
```

	Unnamed: 0	Mon	Tue	Wed	Thu	Fri	per_Bonus	perc_Bonus	perc_Bonus	Bonus
0	Steven	34	27	15	NaN	33	0.12	0.12	0.12	13.1088
1	Mike	45	9	74	87.0	12	0.15	0.15	0.15	34.0950
2	Andi	17	33	54	8.0	29	0.10	0.10	0.10	14.1200
3	Paul	87	67	27	45.0	7	0.20	0.20	0.20	46.6800

Add Subtract, multiplication, with scalar value

```
In [271... titanic.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck	No_of_Rel
0	0	3	male	22.0	1	0	7.2500	S	NaN	1

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck	No_of_Rel
1	1	1	female	38.0	1	0	71.2833		C	C
2	1	3	female	26.0	0	0	7.9250		S	NaN
3	1	1	female	35.0	1	0	53.1000		S	C
4	0	3	male	35.0	0	0	8.0500		S	NaN

Subtracting with pandas.

In [274...]:

```
1912 - titanic.age
```

Out[274...]:

```
0    1890.0
1    1874.0
2    1886.0
3    1877.0
4    1877.0
...
886   1885.0
887   1893.0
888     NaN
889   1886.0
890   1880.0
Name: age, Length: 891, dtype: float64
```

Create the year of birth using the .sub() method.

In [279...]:

```
titanic["YoB"] = titanic.age.sub(1912, fill_value = None).mul(-1)
```

In [280...]:

```
titanic.head()
```

Out[280...]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck	No_of_Rel	YoB
0	0	3	male	22.0	1	0	7.2500		S	NaN	1 1890.0
1	1	1	female	38.0	1	0	71.2833		C	C	1 1874.0
2	1	3	female	26.0	0	0	7.9250		S	NaN	0 1886.0
3	1	1	female	35.0	1	0	53.1000		S	C	1 1877.0
4	0	3	male	35.0	0	0	8.0500		S	NaN	0 1877.0

Use the fx rate and division to convert currencies.

In [281...]:

```
fx_rate = 1.1 #1.1 dollars per Euro
```

In [282...]:

```
titanic["EUR_fare"] = titanic.fare.div(fx_rate)
```

In [283...]:

```
titanic.head()
```

Out[283...]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck	No_of_Rel	YoB	EUR_fare
--	----------	--------	-----	-----	-------	-------	------	----------	------	-----------	-----	----------

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck	No_of_Rel	YoB	EUR_fare
0	0	3	male	22.0	1	0	7.2500		S	NaN	1	1890.0
1	1	1	female	38.0	1	0	71.2833		C	C	1	1874.0
2	1	3	female	26.0	0	0	7.9250		S	NaN	0	1886.0
3	1	1	female	35.0	1	0	53.1000		S	C	1	1877.0
4	0	3	male	35.0	0	0	8.0500		S	NaN	0	1877.0

In [285...]

```
# Drop some columns to make the data more organized.
titanic.drop(columns = ["sibsp", "parch", "deck", "YoB", "EUR_fare"], inplace = True)
```

In [286...]

```
titanic.head()
```

Out[286...]

	survived	pclass	sex	age	fare	embarked	No_of_Rel
0	0	3	male	22.0	7.2500	S	1
1	1	1	female	38.0	71.2833	C	1
2	1	3	female	26.0	7.9250	S	0
3	1	1	female	35.0	53.1000	S	1
4	0	3	male	35.0	8.0500	S	0

Back to the sales data.

See the screenshots as the code was not able to be perfected here.

In [288...]

```
sales
```

Out[288...]

	Unnamed: 0	Mon	Tue	Wed	Thu	Fri	per_Bonus	perc_Bonus	perc_Bonus	Bonus
0	Steven	34	27	15	NaN	33	0.12	0.12	0.12	13.1088
1	Mike	45	9	74	87.0	12	0.15	0.15	0.15	34.0950
2	Andi	17	33	54	8.0	29	0.10	0.10	0.10	14.1200
3	Paul	87	67	27	45.0	7	0.20	0.20	0.20	46.6800

In [297...]

```
fixed_costs = 5
```

In [298...]

```
# Not working for some reason
sales.iloc[:, :-2].sub(fixed_costs, fill_value = 0)
```

Out[298...]

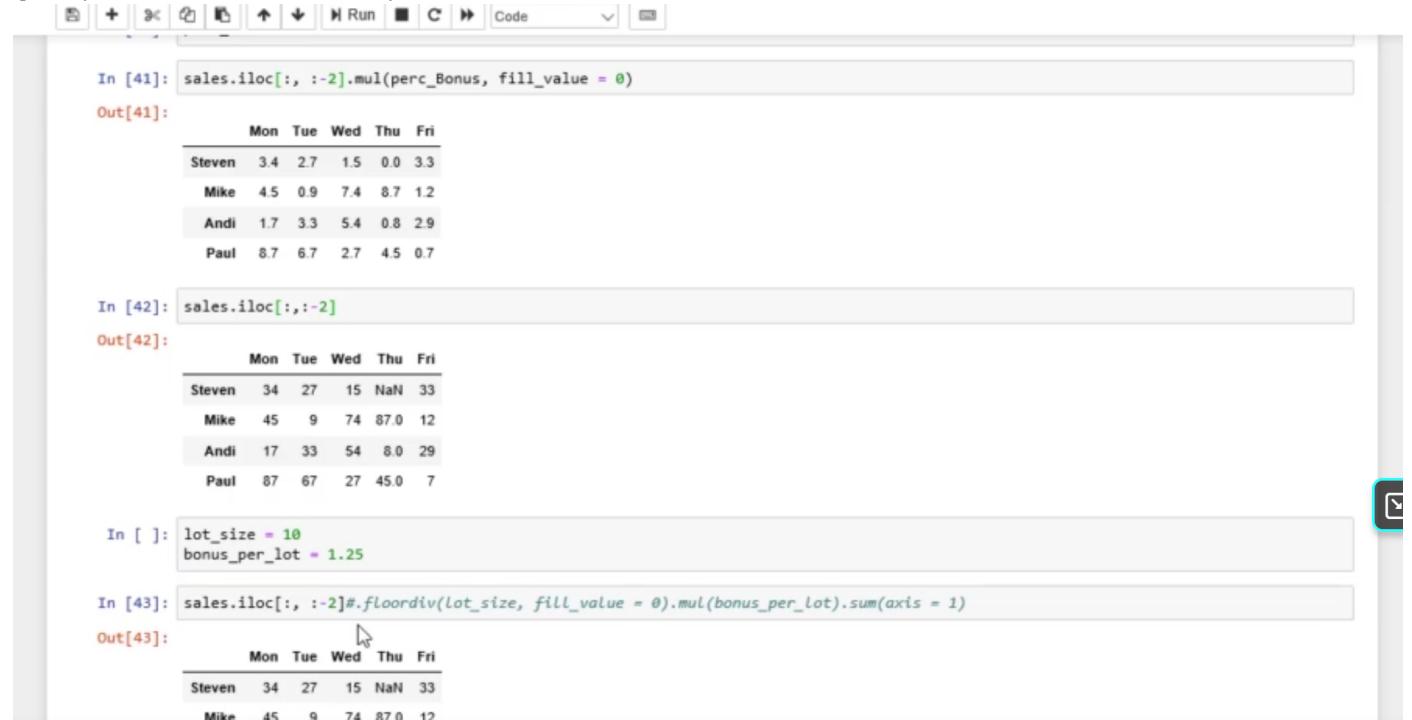
	Unnamed: 0	Mon	Tue	Wed	Thu	Fri	per_Bonus	perc_Bonus
0	Steven	34	27	15	NaN	33	0.12	0.12
1	Mike	45	9	74	87.0	12	0.15	0.15
2	Andi	17	33	54	8.0	29	0.10	0.10
3	Paul	87	67	27	45.0	7	0.20	0.20

```
In [301... type(fixed_costs)
```

```
Out[301... int
```

```
In [304... #type(sales.iloc[:, :-2])  
#sales.iloc[:, :-2]
```

```
sales.iloc[:, :-2].sub(fixed_costs, fill_value = 0)
```



In [41]: sales.iloc[:, :-2].sub(fixed_costs, fill_value = 0)

Out[41]:

	Mon	Tue	Wed	Thu	Fri
Steven	3.4	2.7	1.5	0.0	3.3
Mike	4.5	0.9	7.4	8.7	1.2
Andi	1.7	3.3	5.4	0.8	2.9
Paul	8.7	6.7	2.7	4.5	0.7

In [42]: sales.iloc[:, :-2]

Out[42]:

	Mon	Tue	Wed	Thu	Fri
Steven	34	27	15	NaN	33
Mike	45	9	74	87.0	12
Andi	17	33	54	8.0	29
Paul	87	67	27	45.0	7

In []: lot_size = 10
bonus_per_lot = 1.25

In [43]: sales.iloc[:, :-2].floordiv(lot_size, fill_value = 0).mul(bonus_per_lot).sum(axis = 1)

Out[43]:

	Mon	Tue	Wed	Thu	Fri
Steven	34	27	15	NaN	33
Mike	45	9	74	87.0	12

jupyter Data Preparation Feature Creation Last Checkpoint: vor 25 Minuten (autosaved)



Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3



In [42]: sales.iloc[:, :-2]

Out[42]:

	Mon	Tue	Wed	Thu	Fri
Steven	34	27	15	NaN	33
Mike	45	9	74	87.0	12
Andi	17	33	54	8.0	29
Paul	87	67	27	45.0	7

In [45]: lot_size = 10
bonus_per_lot = 1.25

In [48]: sales.iloc[:, :-2].floordiv(lot_size, fill_value = 0).mul(bonus_per_lot).sum(axis = 1)

Out[48]: Steven 11.25
Mike 25.00
Andi 13.75
Paul 25.00
dtype: float64

Create Dataframes from Scratch with pd.DataFrame()

```
In [306... import pandas as pd
```

```
In [307... player = ["Lionel Messi", "Christiano Ronaldo", "Neymar Junior", "Kylian Mbappe", "Manuel
In [308... nationality = ["Argentina", "Portugal", "Brazil", "France", "Germany"]
In [309... club = ["FC Barcelona", "Juventus FC", "Paris SG", "Paris SG", "FC Bayem"]
In [310... champion = ["False", "False", "False", "True", "True"]
In [311... height = [1.70, 1.87, 1.75, 1.78, 1.93]
In [312... goals = [45, 44, 28, 21, 0]
```

To create a chart you need to create a dictionary. The keys are what the columns will be called on the chart. The value will be the values.

```
In [314... dic = {"Player": player,
             "Nationality": nationality,
             "Club": club,
             "World_Champion": champion,
             "Height": height,
             "Goals": goals,
             }
In [315... # df stands for dataframe use pd.DataFrame(dic) pass in a dictionary.
df = pd.DataFrame(dic)
In [316... df
```

	Player	Nationality	Club	World_Champion	Height	Goals
0	Lionel Messi	Argentina	FC Barcelona	False	1.70	45
1	Christiano Ronaldo	Portugal	Juventus FC	False	1.87	44
2	Neymar Junior	Brazil	Paris SG	False	1.75	28
3	Kylian Mbappe	France	Paris SG	True	1.78	21
4	Manuel Neuer	Germany	FC Bayem	True	1.93	0

Can set a new index with players as the index.

df. Then press tab to see all that can be done with the new DataFrame.

When using df.set_index() allows the name of the key that will be used as the index to be specified.

```
In [319... players = df.set_index("Player")
```

```
In [321... # Players is simply the df with the players defined.  
players
```

Player	Nationality	Club	World_Champion	Height	Goals
Lionel Messi	Argentina	FC Barcelona	False	1.70	45
Christiano Ronaldo	Portugal	Juventus FC	False	1.87	44
Neymar Junior	Brazil	Paris SG	False	1.75	28
Kylian Mbappe	France	Paris SG	True	1.78	21
Manuel Neuer	Germany	FC Bayem	True	1.93	0

The zipping method.

The other method is preferable.

```
In [327... # Puts all of the list of entries stored into lists so it is recognized as the characteristics  
zipped = list(zip(nationality, club, champion, height, goals))
```

```
In [337... # For the names of the index it should be listed with = zipped at the end. Could be placed  
messi, ronaldo, neymar, mbappe, neuer = zipped
```

```
In [338... messi
```

```
Out[338... ('Argentina', 'FC Barcelona', 'False', 1.7, 45)
```

```
In [339... ronaldo
```

```
Out[339... ('Portugal', 'Juventus FC', 'False', 1.87, 44)
```

```
In [340... # data is the zipped names that stores the other attributtes.  
# These characteristics were made using the list(zip) shown above..  
df2 = pd.DataFrame(data = [messi, ronaldo, neymar, mbappe, neuer],  
                     index = ["Lionel Messi", "Christiano Ronaldo", "Neymar Junior", "Kylian Mbappe", "Manuel Neuer"],  
                     columns = ["Nationality", "Club", "World_Champion", "Height", "Goals_2018"])
```

```
In [341... df2
```

Player	Nationality	Club	World_Champion	Height	Goals_2018
Lionel Messi	Argentina	FC Barcelona	False	1.70	45
Christiano Ronaldo	Portugal	Juventus FC	False	1.87	44
Neymar Junior	Brazil	Paris SG	False	1.75	28
Kylian Mbappe	France	Paris SG	True	1.78	21
Manuel Neuer	Germany	FC Bayem	True	1.93	0

Create a pandas Series

```
In [347... pd.Series(index = player, data = nationality, name = "Nationality")
```

```
Out[347... Lionel Messi          Argentina
          Christiano Ronaldo    Portugal
          Neymar Junior         Brazil
          Kylian Mbappe         France
          Manuel Neuer          Germany
          Name: Nationality, dtype: object
```

```
In [354... type(pd.Series(index = player, data = nationality, name = "Nationality"))
```

```
Out[354... pandas.core.series.Series
```

Convert pandas series to a dataframe.

Just use `.to_frame()`

```
In [356... df3 = pd.Series(index = player, data = nationality, name = "Nationality").to_frame()
```

```
In [357... df3
```

```
Out[357...      Nationality
Lionel Messi      Argentina
Christiano Ronaldo    Portugal
Neymar Junior       Brazil
Kylian Mbappe       France
Manuel Neuer          Germany
```

Add new column entries into the dataframe.

```
In [358... df3["Club"] = club
```

```
In [359... df3["Height"] = height
```

```
In [360... df3
```

```
Out[360...      Nationality      Club  Height
Lionel Messi      Argentina  FC Barcelona  1.70
Christiano Ronaldo    Portugal  Juventus FC  1.87
Neymar Junior       Brazil    Paris SG  1.75
Kylian Mbappe       France    Paris SG  1.78
Manuel Neuer          Germany  FC Bayem  1.93
```

```
In [361... ## Adding new Rows (hands-on-approach)
```

```
In [363... type(players)
```

```
Out[363... pandas.core.frame.DataFrame
```

```
In [362... players
```

```
Out[362...      Nationality      Club  World_Champion  Height  Goals
Player
Lionel Messi    Argentina  FC Barcelona      False    1.70    45
Christian Ronaldo    Portugal Juventus FC      False    1.87    44
Neymar Junior    Brazil    Paris SG      False    1.75    28
Kylian Mbappe    France    Paris SG      True     1.78    21
Manuel Neuer    Germany  FC Bayem      True     1.93     0
```

Remember .reset_index() makes the index numerical again. It is very useful.

```
.reset_index(inplace = True)
```

```
In [364... players.reset_index(inplace = True)
```

```
In [365... players
```

```
Out[365...      Player  Nationality      Club  World_Champion  Height  Goals
0    Lionel Messi    Argentina  FC Barcelona      False    1.70    45
1  Christian Ronaldo    Portugal Juventus FC      False    1.87    44
2    Neymar Junior    Brazil    Paris SG      False    1.75    28
3    Kylian Mbappe    France    Paris SG      True     1.78    21
4    Manuel Neuer    Germany  FC Bayem      True     1.93     0
```

```
In [385... players.loc[5, :] = ["Sergios Ramos", "Spain", "Real Madrid", True, 1.84, 5]
```

```
In [367... players
```

```
Out[367...      Player  Nationality      Club  World_Champion  Height  Goals
0    Lionel Messi    Argentina  FC Barcelona      False    1.70    45.0
1  Christian Ronaldo    Portugal Juventus FC      False    1.87    44.0
2    Neymar Junior    Brazil    Paris SG      False    1.75    28.0
3    Kylian Mbappe    France    Paris SG      True     1.78    21.0
```

	Player	Nationality	Club	World_Champion	Height	Goals
4	Manuel Neuer	Germany	FC Bayem	True	1.93	0.0
5	Sergios Ramos	Spain	Real Madrid	True	1.84	5.0

Add Many Rows

```
In [378...]: new = pd.DataFrame(
    data = [{"Mohamed Salah", "Egypt", "FC Liverpool", False, 1.75, 44},
            {"Luis Suarez", "Uruguay", "FC Barcelona", False, 1.82, 31}],
    columns = players.columns
)
# Note the first list does not end until the last line that comes right before columns.
```

```
In [380...]: new
```

```
Out[380...]:
```

	Player	Nationality	Club	World_Champion	Height	Goals
0	Mohamed Salah	Egypt	FC Liverpool	False	1.75	44
1	Luis Suarez	Uruguay	FC Barcelona	False	1.82	31

```
In [381...]: # Deprecated method instead use pd.concat()
players = players.append(new, ignore_index = True)
```

```
C:\Users\alonz\AppData\Local\Temp\ipykernel_27368\2268971855.py:1: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
players = players.append(new, ignore_index = True)
```

Adding New Rows to a Dataframe

```
In [389...]: men2004 = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\men2004.csv")
```

```
In [390...]: men2008 = pd.read_csv(r"C:\Users\alonz\Downloads\Video_Lecture_NBs\Video_Lecture_NBs\men2008.csv")
```

```
In [392...]: men2004.head(10)
```

```
Out[392...]:
```

	Athlete	Medals
0	PHELPS, Michael	8
1	THORPE, Ian	4
2	SCHOEMAN, Roland	3
3	PEIRSOL, Aaron	3
4	CROCKER, Ian	3
5	KITAJIMA, Kosuke	3
6	HANSEN, Brendan	3
7	VAN DEN HOOGENBAND, Pieter	3

Athlete Medals

8	HACKETT, Grant	3
9	MORITA, Tomomi	2

In [393...]

men2008.head(10)

Out[393...]

Athlete Medals

0	PHELPS, Michael	8
1	LOCHTE, Ryan	4
2	BERNARD, Alain	3
3	SULLIVAN, Eamon	3
4	LAUTERSTEIN, Andrew	3
5	GREVERS, Matt	3
6	LEZAK, Jason	3
7	CSEH, Laszlo	3
8	KITAJIMA, Kosuke	3
9	PEIRSOL, Aaron	3

In [394...]

type(men2004)

Out[394...]

pandas.core.frame.DataFrame

In [397...]

Last of the men 2004 data.
men2004.tail(10)

Out[397...]

Athlete Medals

49	GYURTA, Daniel	1
50	JENSEN, Larsen	1
51	KENKHUIS, Johan	1
52	KETCHUM, Dan	1
53	KLIM, Michael	1
54	KRAYZELBURG, Lenny	1
55	KRUPPA, Jens	1
56	BREMBILLA, Emiliano	1
57	MAGNINI, Filippo	1
58	ZWERING, Klaas-Erik	1

**Can combine the two Dataframes with the .append() method.
Even though it is deprecated.**

```
In [406...]: # Set the ignore_index to true to give the data a universal numbered index system.  
men2004.append(men2008, ignore_index = True)
```

```
C:\Users\alonz\AppData\Local\Temp\ipykernel_27368\2243492222.py:2: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.  
men2004.append(men2008, ignore_index = True)
```

Out[406...]

	Athlete	Medals
0	PHELPS, Michael	8
1	THORPE, Ian	4
2	SCHOEMAN, Roland	3
3	PEIRSOL, Aaron	3
4	CROCKER, Ian	3
...
116	LAGUNOV, Evgeniy	1
117	BERENS, Ricky	1
118	LURZ, Thomas	1
119	MALLET, Gregory	1
120	ZHANG, Lin	1

121 rows × 2 columns

The concat() method is how datasets should be combined. Note that the two sets must be put in a list before combining.

Set ignore index to true in order to have one giant index system.

In [422...]

```
# See the uni-index system below.  
pd.concat([men2004, men2008], ignore_index = True, keys = None)
```

Out[422...]

	Athlete	Medals
0	PHELPS, Michael	8
1	THORPE, Ian	4
2	SCHOEMAN, Roland	3
3	PEIRSOL, Aaron	3
4	CROCKER, Ian	3
...
116	LAGUNOV, Evgeniy	1
117	BERENS, Ricky	1
118	LURZ, Thomas	1
119	MALLET, Gregory	1
120	ZHANG, Lin	1

The best way to combine data sets.

See the separation between data sets by setting:

ignore_index = False and by putting in keys = [list]

Doing this would separate the list by the separate concatenations. In this case it would be the year. The indexing system is broken up.

names = [description_of_keyseperator]

This setting will describe what the keys are separating. In this case it is olympic years. Must come in a list format even for only one value. Names can label create a sublabel that is even closer to the value than the column names.

In [431...]

```
# This allows the user to know where the data is coming from across many dataframes.
# These are the superior settings.
men0408 = pd.concat([men2004, men2008], ignore_index =False, keys = [2004, 2008], names =
men0408
```

Out[431...]

		Athlete	Medals
		Year	
2004	0	PHELPS, Michael	8
	1	THORPE, Ian	4
	2	SCHOEMAN, Roland	3
	3	PEIRSOL, Aaron	3
	4	CROCKER, Ian	3
...
2008	57	LAGUNOV, Evgeniy	1
	58	BERENS, Ricky	1
	59	LURZ, Thomas	1
	60	MALLET, Gregory	1
	61	ZHANG, Lin	1

121 rows × 2 columns

Resetting the index.

Reset index makes the year part of the data table.

In [433...]

```
men0408.reset_index()
```

Out[433...]

	Year	level_1	Athlete	Medals
0	2004	0	PHELPS, Michael	8
1	2004	1	THORPE, Ian	4
2	2004	2	SCHOEMAN, Roland	3
3	2004	3	PEIRSOL, Aaron	3
4	2004	4	CROCKER, Ian	3
...
116	2008	57	LAGUNOV, Evgeniy	1
117	2008	58	BERENS, Ricky	1
118	2008	59	LURZ, Thomas	1
119	2008	60	MALLET, Gregory	1
120	2008	61	ZHANG, Lin	1

121 rows × 4 columns

Reseting the index and drops the level_1.

This is even better.

In [434...]

```
men0408.reset_index().drop(columns = "level_1")
```

Out[434...]

	Year	Athlete	Medals
0	2004	PHELPS, Michael	8
1	2004	THORPE, Ian	4
2	2004	SCHOEMAN, Roland	3
3	2004	PEIRSOL, Aaron	3
4	2004	CROCKER, Ian	3
...
116	2008	LAGUNOV, Evgeniy	1
117	2008	BERENS, Ricky	1
118	2008	LURZ, Thomas	1
119	2008	MALLET, Gregory	1
120	2008	ZHANG, Lin	1

121 rows × 3 columns

Ulmite chart in one line. Same as above.

This is the best chart type all in one line.

In [438...]

```
# All together.
men0408_ultimate = pd.concat([men2004, men2008], ignore_index = False, keys = [2004, 2008],
```

Here is another format just in case.

```
men0408_ultimate =
```

```
pd.concat([men2004, men2008], ignore_index =False, keys = [2004, 2008], names =  
["Year"]).reset_index().reset_index().drop(columns = "level_1")
```

In [439...]

```
men0408_ultimate
```

Out[439...]

	index	Year	Athlete	Medals
0	0	2004	PHELPS, Michael	8
1	1	2004	THORPE, Ian	4
2	2	2004	SCHOEMAN, Roland	3
3	3	2004	PEIRSOL, Aaron	3
4	4	2004	CROCKER, Ian	3
...
116	116	2008	LAGUNOV, Evgeniy	1
117	117	2008	BERENS, Ricky	1
118	118	2008	LURZ, Thomas	1
119	119	2008	MALLET, Gregory	1
120	120	2008	ZHANG, Lin	1

121 rows × 4 columns

New Section

Manipulating Elements in a DataFrame

In [470...]

```
titanic = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

In [442...]

```
titanic.head()
```

Out[442...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN

Targeting and changing cells with .loc

Lock onto a specific cell by using the `.iloc[index_No, "character"] = new_value`

In [450...]

```
## Here is how to change passenger at index 1's age from 38 to 40
# Remember to use .loc because it is for an integer.
titanic.loc[1, "age"] = 40
```

Notice how the person at index 1 age was changed. The changes are automatically saved.

In [451...]

```
titanic.head(2)
```

Out[451...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	40.0	1	0	71.2833	C	C

The ages of several passengers can be changed as well.

In [454...]

```
titanic.loc[2:4, "age"] = 42
```

In [455...]

```
titanic.head()
```

Out[455...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	40.0	1	0	71.2833	C	C
2	1	3	female	42.0	0	0	7.9250	S	NaN
3	1	1	female	42.0	1	0	53.1000	S	C
4	0	3	male	42.0	0	0	8.0500	S	NaN

The `iloc()` method can be used to change a value as well.

`.iloc[[index_No, column_No]] = New_value`

In [456...]

```
# The first passenger could be made to survive.
titanic.iloc[0,0] = 1
```

A new survivor was added in assuming he was found out to be alive.

In [457...]

```
titanic.head()
```

Out[457...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	1	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	40.0	1	0	71.2833	C	C
2	1	3	female	42.0	0	0	7.9250	S	NaN

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
3	1	1	female	42.0	1	0	53.1000	S	C
4	0	3	male	42.0	0	0	8.0500	S	NaN

Mathematical operations can be applied when applying new values.

To select the all of the indexes etc. use ":" in the proper section.

In [471...]

```
# Note the data has been reset to the original using the original data.
titanic.loc[:, "age"] = titanic.loc[:, "age"] * 12
```

The age of all the first passenger is given in months.

In [473...]

```
titanic.head()
```

Out[473...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	264.0	1	0	7.2500	S	NaN
1	1	1	female	456.0	1	0	71.2833	C	C
2	1	3	female	312.0	0	0	7.9250	S	NaN
3	1	1	female	420.0	1	0	53.1000	S	C
4	0	3	male	420.0	0	0	8.0500	S	NaN

In [474...]

```
# Convert the ages from months back into years.
titanic.loc[:, "age"] = titanic.loc[:, "age"] / 12
```

In [475...]

```
titanic.head()
```

Out[475...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN

Important

.loc can be used to find datapoints that meet a certain characteristic.

.loc does not need a value for the index and column

.loc can work with a single conditional. It does not need a [index, column]

.loc[dataset.column > value].index

.index can be used to find where the condition is met.

In [483...]

```
# Babies are classified as those who are less than one year old.  
index_babies = titanic.loc[titanic.age < 1].index
```

In [485...]

```
## Thankfully all babies aboard the Titanic were saved.
```

In [486...]

```
titanic.loc[index_babies]
```

Out[486...]

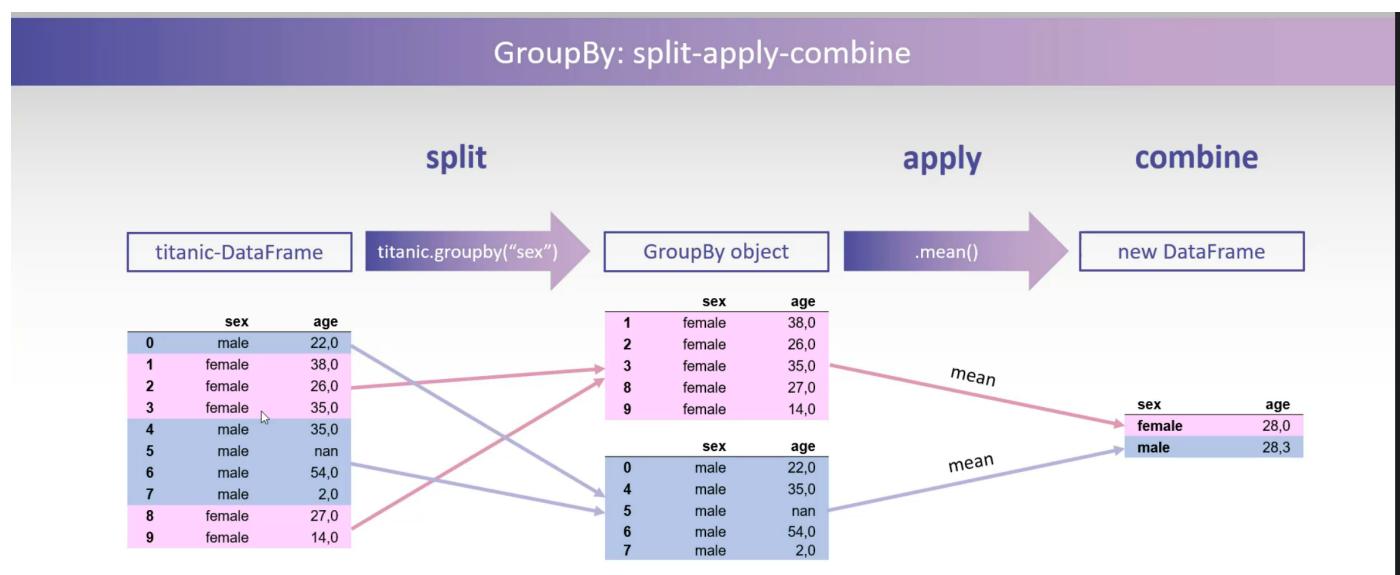
	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
78	1	2	male	0.83	0	2	29.0000	S	NaN
305	1	1	male	0.92	1	2	151.5500	S	C
469	1	3	female	0.75	2	1	19.2583	C	NaN
644	1	3	female	0.75	2	1	19.2583	C	NaN
755	1	2	male	0.67	1	1	14.5000	S	NaN
803	1	3	male	0.42	0	1	8.5167	C	NaN
831	1	2	male	0.83	1	1	18.7500	S	NaN

New Section

GroupBy Operations

GroupBy is Extremely Important

Groupby operations allows for data to be split into different groups. It also allows for the seperated group to be processed individually in a statistical manner. Then the split data can be rejoined into one table again.



Pandas Groupby Operations

Understanding Groupby Objects

In [491...]

```
import pandas as pd
titanic = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

In [492...]

```
titanic.head()
```

Out[492...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN

In [493...]

```
titanic.tail()
```

Out[493...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
886	0	2	male	27.0	0	0	13.00	S	NaN
887	1	1	female	19.0	0	0	30.00	S	B
888	0	3	female	NaN	1	2	23.45	S	NaN
889	1	1	male	26.0	0	0	30.00	C	C
890	0	3	male	32.0	0	0	7.75	Q	NaN

In [494...]

```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   survived  891 non-null   int64  
 1   pclass    891 non-null   int64  
 2   sex       891 non-null   object  
 3   age       714 non-null   float64 
 4   sibsp    891 non-null   int64  
 5   parch    891 non-null   int64  
 6   fare      891 non-null   float64 
 7   embarked  889 non-null   object  
 8   deck      203 non-null   object  
dtypes: float64(2), int64(4), object(3)
memory usage: 62.8+ KB
```

Slice up pieces of data using the `iloc()` feature. The `.loc()` could be used also.

The indexes to be examined can be honed in on. The columns of interest can be isolated as well. These two things can make a slice. Slices is pieces of data that you isolate yourself using the .iloc() method. It is not prebuilt into the software.

In [500...]

```
# This slice contains indexes from zero to 9. Sex and age is what is selected here by column
# If possible get in the habit of using lists for two values as the end range of a :int
titanic_slice = titanic.iloc[:10, [2,3]]
```

In [501...]

```
titanic_slice
```

Out[501...]

	sex	age
0	male	22.0
1	female	38.0
2	female	26.0
3	female	35.0
4	male	35.0
5	male	NaN
6	male	54.0
7	male	2.0
8	female	27.0
9	female	14.0

Use .groupby("column_name") to make a groupby object.

In [508...]

```
# Object created. In the next line it will be put into a variable.
titanic_slice.groupby("sex")
```

Out[508...]

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001CB70E1F970>
```

In [512...]

```
### Creates a groupbyObject gbo.
gbo = titanic_slice.groupby("sex")
```

In [513...]

```
type(gbo)
```

Out[513...]

```
pandas.core.groupby.generic.DataFrameGroupBy
```

Groupby objects has attributes and methods.

To view these type gbo. and then press Tab.

Remember the slice code:

```
titanic_slice = titanic.iloc[:10, [2,3]]
```

It took the slice with .groupby() and it separated it into male and female.

```
In [517... gbo.groups
```

```
Out[517... {'female': [1, 2, 3, 8, 9], 'male': [0, 4, 5, 6, 7]}
```

```
In [518... # gbo can be made into a list
1 = list(gbo)
```

```
In [519... 1
```

```
Out[519... [('female',
             sex    age
             1  female  38.0
             2  female  26.0
             3  female  35.0
             8  female  27.0
             9  female  14.0),
            ('male',
             sex    age
             0  male   22.0
             4  male   35.0
             5  male   NaN
             6  male   54.0
             7  male   2.0)]
```

```
In [520... len(1)
```

```
Out[520... 2
```

```
In [521... 1[0]
```

```
Out[521... ('female',
             sex    age
             1  female  38.0
             2  female  26.0
             3  female  35.0
             8  female  27.0
             9  female  14.0)
```

```
In [528... # Tuple is a list that cannot change. It comes in () instead of square brackets.
type(1[0])
```

```
Out[528... tuple
```

```
In [529... 1[0][0]
```

```
Out[529... 'female'
```

```
In [531... 1[0][1]
```

```
Out[531...      sex  age
             1  female  38.0
             2  female  26.0
             3  female  35.0
```

```
sex  age
```

```
8  female  27.0
```

```
9  female  14.0
```

```
In [532...]
```

```
type(l[0][1])
```

```
Out[532...]
```

```
pandas.core.frame.DataFrame
```

```
In [533...]
```

```
l[1]
```

```
Out[533...]
```

```
('male',
   sex      age
0  male    22.0
4  male    35.0
5  male     NaN
6  male    54.0
7  male     2.0)
```

```
In [535...]
```

```
titanic_slice.loc[titanic_slice.sex == "female"]
```

```
Out[535...]
```

	sex	age
1	female	38.0
2	female	26.0
3	female	35.0
8	female	27.0
9	female	14.0

```
In [536...]
```

```
titanic_slice_f = titanic_slice.loc[titanic_slice.sex == "female"]
titanic_slice_f
```

```
Out[536...]
```

	sex	age
1	female	38.0
2	female	26.0
3	female	35.0
8	female	27.0
9	female	14.0

```
In [537...]
```

```
titanic_slice_m = titanic_slice.loc[titanic_slice.sex == "male"]
titanic_slice_m
```

```
Out[537...]
```

	sex	age
0	male	22.0
4	male	35.0
5	male	NaN

```
sex  age
6  male  54.0
7  male   2.0
```

```
In [538]: titanic_slice_f.equals(l[0][1])
```

```
Out[538]: True
```

```
In [539]: for element in gbo:
    print(element[1])
```

```
      sex  age
1  female  38.0
2  female  26.0
3  female  35.0
8  female  27.0
9  female  14.0
      sex  age
0  male   22.0
4  male   35.0
5  male    NaN
6  male   54.0
7  male   2.0
```

Splitting with many Keys

```
In [5]: import pandas as pd
```

```
In [6]: summer = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\summer")
```

```
In [7]: summer.head()
```

```
Out[7]:
```

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze
3	1896	Athens	Aquatics	Swimming	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold
4	1896	Athens	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver

```
In [8]: summer.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31165 entries, 0 to 31164
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Year              31165 non-null   int64  
 1   City              31165 non-null   object  
 2   Sport             31165 non-null   object  
 3   Discipline        31165 non-null   object  

```

```
4 Athlete      31165 non-null  object
5 Country      31161 non-null  object
6 Gender       31165 non-null  object
7 Event        31165 non-null  object
8 Medal        31165 non-null  object
dtypes: int64(1), object(8)
memory usage: 2.1+ MB
```

```
In [9]: # Remember .nunique() gives a number of unique values there are.
summer.Country.nunique()
```

```
Out[9]: 147
```

Here .groupby("Country") is filtering by the country that participated.

```
In [10]: # The first split is being done by country.
split1 = summer.groupby("Country")
```

```
In [11]: l = list(split1)
```

```
In [20]: #Shortened because list is way too long
l[0:3]
```

```
Out[20]: [('AFG',
            Year      City      Sport Discipline      Athlete Country Gender \
            28965  2008  Beijing  Taekwondo  Taekwondo  NIKPAI, Rohullah  AFG  Men
            30929  2012  London  Taekwondo  Taekwondo  NIKPAI, Rohullah  AFG  Men
            Event  Medal
            28965  - 58 KG  Bronze
            30929  58 - 68 KG  Bronze ),
           ('AHO',
            Year      City      Sport Discipline      Athlete Country Gender \
            19323  1988  Seoul    Sailing    Sailing  BOERSMA, Jan D.  AHO  Men
            Event  Medal
            19323  Board (Division II)  Silver ),
           ('ALG',
            Year      City      Sport Discipline      Athlete Country \
            17060  1984  Los Angeles  Boxing  Boxing  ZAOUI, Mohamed  ALG
            17064  1984  Los Angeles  Boxing  Boxing  MOUSSA, Mustapha  ALG
            19874  1992  Barcelona  Athletics  Athletics  BOULMERKA, Hassiba  ALG
            20200  1992  Barcelona  Boxing  Boxing  SOLTANI, Hocine  ALG
            21610  1996  Atlanta   Athletics  Athletics  MORCELI, Nourredine  ALG
            21946  1996  Atlanta   Boxing  Boxing  SOLTANI, Hocine  ALG
            21960  1996  Atlanta   Boxing  Boxing  BAHARI, Mohamed  ALG
            23536  2000  Sydney    Athletics  Athletics  MERAH-BENIDA, Nouria  ALG
            23624  2000  Sydney    Athletics  Athletics  SAIDI-SIEF, Ali  ALG
            23631  2000  Sydney    Athletics  Athletics  SAID GUERNI, Djabir  ALG
            23655  2000  Sydney    Athletics  Athletics  HAMMAD, Abderrahmane  ALG
            23890  2000  Sydney    Boxing  Boxing  ALLALOU, Mohamed  ALG
            28602  2008  Beijing   Judo  Judo  HADDAD, Soraya  ALG
            28637  2008  Beijing   Judo  Judo  BENIKHLEF, Amar  ALG
            29600  2012  London   Athletics  Athletics  MAKHLOUFI, Taoufik  ALG
            Gender
            17060  Men
            17064  Men
            Event  Medal
            71-75KG  Bronze
            75 - 81KG (Light-Heavyweight)  Bronze
```

```

19874 Women 1500M Gold
20200 Men 54 - 57KG (Featherweight) Bronze
21610 Men 1500M Gold
21946 Men 57 - 60KG (Lightweight) Gold
21960 Men 71-75KG Bronze
23536 Women 1500M Gold
23624 Men 5000M Silver
23631 Men 800M Bronze
23655 Men High Jump Bronze
23890 Men 60 - 63.5KG (Light-Welterweight) Bronze
28602 Women 48 - 52KG (Half-Lightweight) Bronze
28637 Men 81 - 90KG (Middleweight) Silver
29600 Men 1500M Gold )]

```

In [22]: `len(l)`

Out[22]: 147

In [23]: `l[100][1]`

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal
5031	1928	Amsterdam	Aquatics	Swimming	YLDEFONSO, Teofilo	PHI	Men	200M Breaststroke	Bronze
5741	1932	Los Angeles	Aquatics	Swimming	YLDEFONSO, Teofilo	PHI	Men	200M Breaststroke	Bronze
5889	1932	Los Angeles	Athletics	Athletics	TORIBIO, Simeon Galvez	PHI	Men	High Jump	Bronze
5922	1932	Los Angeles	Boxing	Boxing	VILLANUEVA, Jose	PHI	Men	50.8 - 54KG (Bantamweight)	Bronze
6447	1936	Berlin	Athletics	Athletics	WHITE, Miguel S.	PHI	Men	400M Hurdles	Bronze
11005	1964	Tokyo	Boxing	Boxing	VILLANUEVA, Anthony N.	PHI	Men	54 - 57KG (Featherweight)	Silver
18513	1988	Seoul	Boxing	Boxing	SERANTES, Leopoldo	PHI	Men	- 48KG (Light-Flyweight)	Bronze
20184	1992	Barcelona	Boxing	Boxing	VELASCO, Roel	PHI	Men	- 48KG (Light-Flyweight)	Bronze
21927	1996	Atlanta	Boxing	Boxing	VELASCO, Mansueto	PHI	Men	- 48KG (Light-Flyweight)	Silver

In [24]: `# Splitting by two characteristics instead of one.
.groupby() can accept a list.
split2 designates that it takes in two characteristics.
split2 = summer.groupby(by = ["Country", "Gender"])`

In [25]: `# Assign the variable.
l2 = summer.groupby(by = ["Country", "Gender"])`

In [26]: `# Turn the groupby into a list.
l2 = list(split2)
l2[0:2]`

Out[26]: `[((('AFG', 'Men'),`

Year	City	Sport	Discipline	Athlete	Country	Gender
------	------	-------	------------	---------	---------	--------

```

28965 2008 Beijing Taekwondo Taekwondo NIKPAI, Rohullah AFG Men
30929 2012 London Taekwondo Taekwondo NIKPAI, Rohullah AFG Men

Event Medal
28965 - 58 KG Bronze
30929 58 - 68 KG Bronze ),
('AHO', 'Men'),
Year City Sport Discipline Athlete Country Gender \
19323 1988 Seoul Sailing Sailing BOERSMA, Jan D. AHO Men

Event Medal
19323 Board (Division Ii) Silver )

```

In [559...]
The length of the list gives the number of groups.
len(12)

Out[559...]
236

In [561...]
12[104]

Out[561...]
('IRL', 'Women'),
Year City Sport Discipline Athlete Country \
21356 1996 Atlanta Aquatics Swimming SMITH, Michelle Marie IRL
21369 1996 Atlanta Aquatics Swimming SMITH, Michelle Marie IRL
21375 1996 Atlanta Aquatics Swimming SMITH, Michelle Marie IRL
21381 1996 Atlanta Aquatics Swimming SMITH, Michelle Marie IRL
23627 2000 Sydney Athletics Athletics O'SULLIVAN, Sonia IRL
29896 2012 London Boxing Boxing TAYLOR, Katie IRL

Gender Event Medal
21356 Women 200M Butterfly Bronze
21369 Women 200M Individual Medley Gold
21375 Women 400M Freestyle Gold
21381 Women 400M Individual Medley Gold
23627 Women 5000M Silver
29896 Women 60 KG Gold)

In [564...]
12[104][0]

Out[564...]
('IRL', 'Women')

In [566...]
Sorts the data by the gender and the country.
12[104][1]

Out[566...]

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal
21356	1996	Atlanta	Aquatics	Swimming	SMITH, Michelle Marie	IRL	Women	200M Butterfly	Bronze
21369	1996	Atlanta	Aquatics	Swimming	SMITH, Michelle Marie	IRL	Women	200M Individual Medley	Gold
21375	1996	Atlanta	Aquatics	Swimming	SMITH, Michelle Marie	IRL	Women	400M Freestyle	Gold
21381	1996	Atlanta	Aquatics	Swimming	SMITH, Michelle Marie	IRL	Women	400M Individual Medley	Gold
23627	2000	Sydney	Athletics	Athletics	O'SULLIVAN, Sonia	IRL	Women	5000M	Silver
29896	2012	London	Boxing	Boxing	TAYLOR, Katie	IRL	Women	60 KG	Gold

Split-Apply-Combine

In [570...]

```
import pandas as pd
titanic = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

In [571...]

```
titanic_slice = titanic.iloc[:10, [2,3]]
```

In [572...]

```
titanic_slice
```

Out[572...]

	sex	age
0	male	22.0
1	female	38.0
2	female	26.0
3	female	35.0
4	male	35.0
5	male	NaN
6	male	54.0
7	male	2.0
8	female	27.0
9	female	14.0

In [582...]

```
# Always put the .groupby() into a list.
list(titanic_slice.groupby("sex"))[0][1]
```

Out[582...]

	sex	age
1	female	38.0
2	female	26.0
3	female	35.0
8	female	27.0
9	female	14.0

In [578...]

```
type(list(titanic_slice.groupby("sex"))[0][1])
```

Out[578...]

```
pandas.core.frame.DataFrame
```

In [581...]

```
list(titanic_slice.groupby("sex"))[0][1]
```

Out[581...]

	sex	age
1	female	38.0
2	female	26.0
3	female	35.0

```
sex  age
8  female  27.0
9  female  14.0
```

```
In [583...]: list(titanic_slice.groupby("sex"))[1][1]
```

```
Out[583...]:    sex  age
0  male  22.0
4  male  35.0
5  male  NaN
6  male  54.0
7  male  2.0
```

```
In [585...]: titanic.groupby("sex")
```

```
Out[585...]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001CB7129EF10>
```

Focus on this: being able to generate quick tables using .groupby("characteristic")

```
In [589...]: # Finding the mean using groupby().
titanic_slice.groupby("sex").mean()
```

```
Out[589...]:      age
sex
female  28.00
male    28.25
```

```
In [592...]: #This is known as method chaining.
titanic.groupby("sex").sum()
```

```
Out[592...]:      survived  pclass      age  sibsp  parch      fare
sex
female      233      678  7286.00    218     204  13966.6628
male        109     1379  13919.17    248     136  14727.2865
```

Examine sub characteristics.

```
In [594...]: titanic.groupby("sex").survived.sum()
```

```
Out[594...]: sex
female      233
```

```
male      109
Name: survived, dtype: int64
```

Chooses the characteristics that you want to be noted.

```
.groupby("main_characteristic")["List of other characteristics"]
```

```
In [595...]: titanic.groupby("sex") [["fare", "age"]].max()
```

```
Out[595...]: fare  age
```

sex	fare	age
female	512.3292	63.0
male	512.3292	80.0

```
In [597...]: new_df = titanic.groupby("sex").mean()
```

```
In [598...]: new_df
```

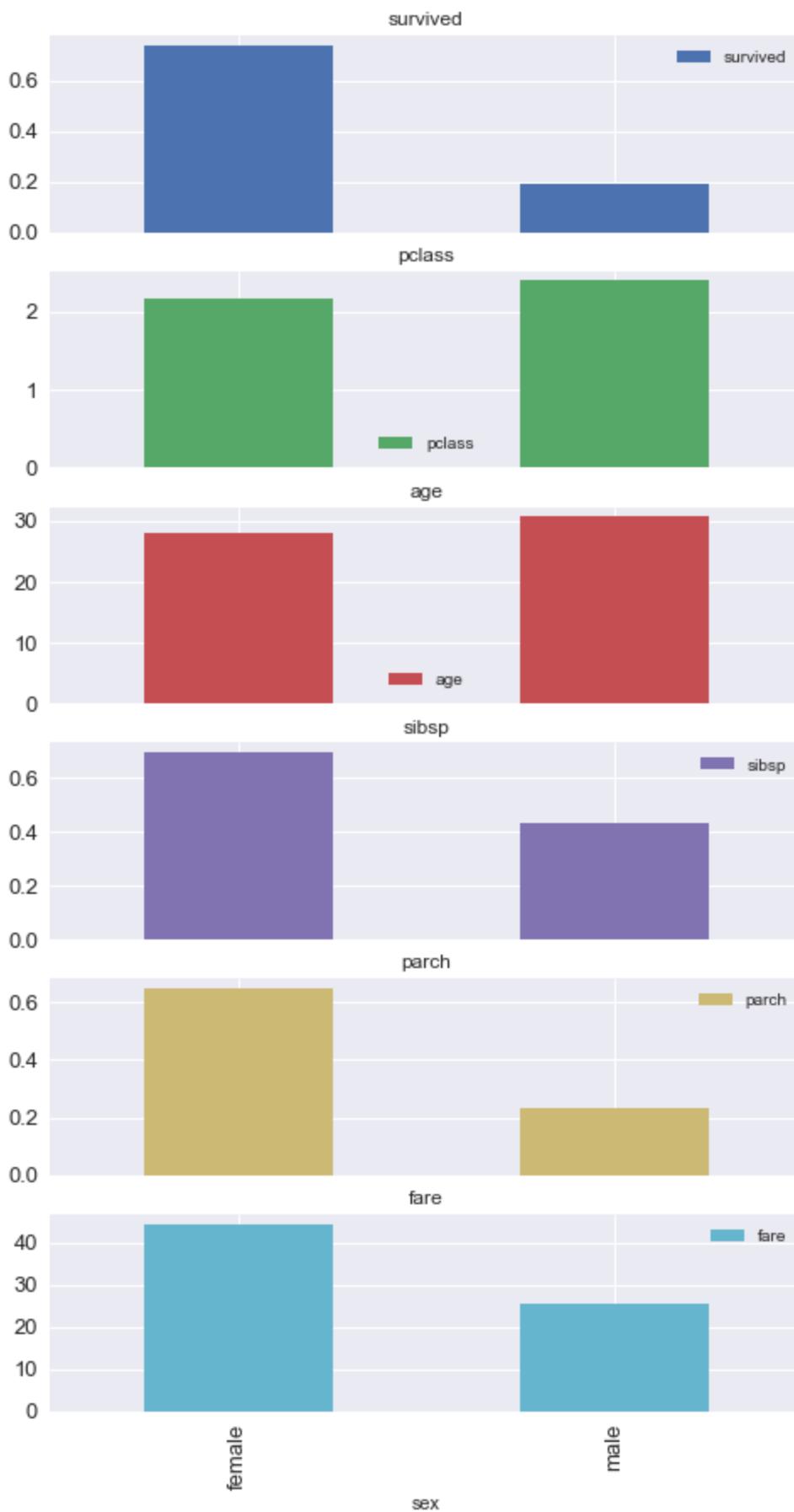
```
Out[598...]: survived  pclass  age  sibsp  parch  fare
```

sex	survived	pclass	age	sibsp	parch	fare
female	0.742038	2.159236	27.915709	0.694268	0.649682	44.479818
male	0.188908	2.389948	30.726645	0.429809	0.235702	25.523893

```
In [599...]: %matplotlib inline
import matplotlib.pyplot as plt
plt.style.use("seaborn")
```

```
In [600...]: new_df.plot(kind = "bar", subplots = True, figsize = (8,15), fontsize = 13)
```

```
Out[600...]: array([<AxesSubplot:title={'center':'survived'}, xlabel='sex',>,
       <AxesSubplot:title={'center':'pclass'}, xlabel='sex',>,
       <AxesSubplot:title={'center':'age'}, xlabel='sex',>,
       <AxesSubplot:title={'center':'sibsp'}, xlabel='sex',>,
       <AxesSubplot:title={'center':'parch'}, xlabel='sex',>,
       <AxesSubplot:title={'center':'fare'}, xlabel='sex',>], dtype=object)
```



Split, Apply, Combine applied

In [602...]

```
import pandas as pd
```

In [603...]

```
summer = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\summer
```

```
In [604...]
```

```
summer.head()
```

```
Out[604...]
```

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze
3	1896	Athens	Aquatics	Swimming	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold
4	1896	Athens	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver

```
In [605...]
```

```
summer.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31165 entries, 0 to 31164
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
---  --  
 0   Year        31165 non-null   int64  
 1   City         31165 non-null   object  
 2   Sport        31165 non-null   object  
 3   Discipline   31165 non-null   object  
 4   Athlete      31165 non-null   object  
 5   Country      31161 non-null   object  
 6   Gender        31165 non-null   object  
 7   Event         31165 non-null   object  
 8   Medal         31165 non-null   object  
dtypes: int64(1), object(8)
memory usage: 2.1+ MB
```

```
In [631...]
```

```
# Grouping the data by the amount of medals they earned. After count them up.
medals_per_country = summer.groupby("Country").Medal.count()
medals_per_country
```

```
Out[631...]
```

```
Country
AFG      2
AHO      1
ALG     15
ANZ     29
ARG    259
...
VIE      2
YUG    435
ZAM      2
ZIM     23
ZZX     48
Name: Medal, Length: 147, dtype: int64
```

```
In [638...]
```

```
# Select the largest counts.
medals_per_country = summer.groupby("Country").Medal.count().nlargest(n = 20)
```

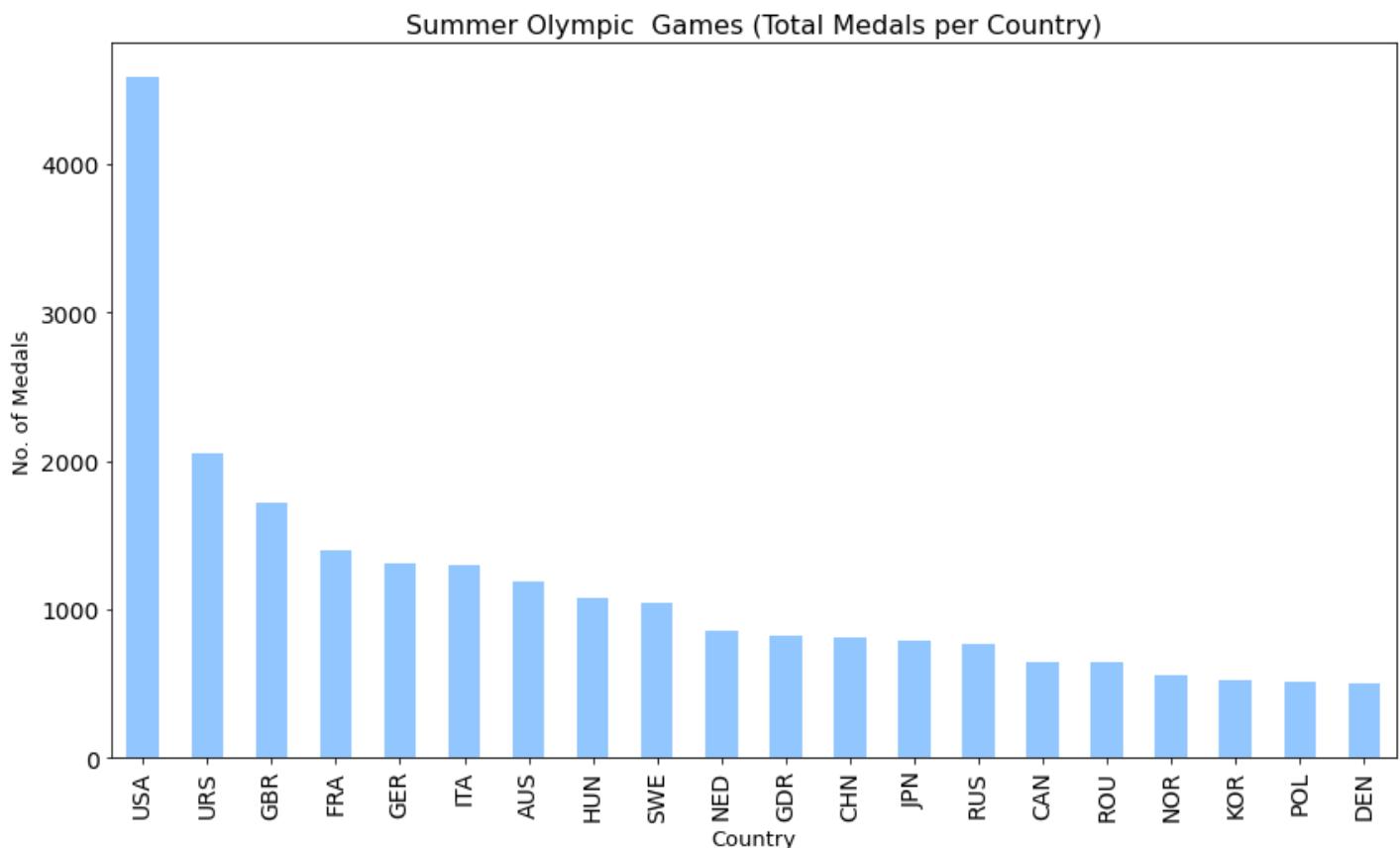
```
In [658...]
```

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
# Need the next two lines for plt.style to work
plt.rcParams.update(plt.rcParamsDefault)
%matplotlib inline
```

```
plt.style.use("seaborn-pastel")
# plt.style.available
```

In [659...]

```
medals_per_country.plot(kind = "bar", figsize = (14, 8), fontsize = 14)
plt.xlabel("Country", fontsize = 13)
plt.ylabel("No. of Medals", fontsize = 13)
plt.title("Summer Olympic Games (Total Medals per Country)", fontsize = 16)
plt.show()
```



Looking at Titanic Data

In [626...]

```
titanic = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NB\Video_Lecture_NB\titanic.csv")
```

In [627...]

```
titanic.head()
```

Out[627...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck
0	0	3	male	22.0	1	0	7.2500	S	NaN
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	S	NaN
3	1	1	female	35.0	1	0	53.1000	S	C
4	0	3	male	35.0	0	0	8.0500	S	NaN

In [628...]

```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
```

```
#   Column    Non-Null Count  Dtype  
---  --  
0   survived    891 non-null   int64  
1   pclass      891 non-null   int64  
2   sex         891 non-null   object  
3   age         714 non-null   float64  
4   sibsp       891 non-null   int64  
5   parch       891 non-null   int64  
6   fare         891 non-null   float64  
7   embarked    889 non-null   object  
8   deck        203 non-null   object  
dtypes: float64(2), int64(4), object(3)  
memory usage: 62.8+ KB
```

In [629...]

```
titanic.describe()
```

Out[629...]

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [660...]

```
titanic.fare.mean()
```

Out[660...]

```
32.2042079685746
```

In [662...]

```
# The fare price goes up based on the class of the ticket.  
titanic.groupby("pclass").fare.mean()
```

Out[662...]

```
pclass
1    84.154687
2    20.662183
3    13.675550
Name: fare, dtype: float64
```

In [663...]

```
titanic.survived.sum()
```

Out[663...]

```
342
```

In [664...]

```
titanic.survived.mean()
```

Out[664...]

```
0.3838383838383838
```

In [665...]

```
titanic.groupby("sex").survived.mean()
```

Out[665...]

```
sex
female    0.742038
```

```
male      0.188908
Name: survived, dtype: float64
```

```
In [666...]: titanic.groupby("pclass").survived.mean()
```

```
Out[666...]: pclass
1      0.629630
2      0.472826
3      0.242363
Name: survived, dtype: float64
```

```
In [667...]: titanic.groupby("pclass").survived.mean()
```

```
Out[667...]: pclass
1      0.629630
2      0.472826
3      0.242363
Name: survived, dtype: float64
```

```
In [672...]: # Makes a new column where everyone is an adult.
titanic["ad_chi"] = "adult"
```

```
In [674...]: # Overrides the new column by calling those who are under 18 children.
titanic.loc[titanic.age < 18, "ad_chi"] = "child"
```

```
In [675...]: titanic.head()
```

```
Out[675...]:
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck	ad_chi	
0	0	3	male	22.0	1	0	7.2500		S	NaN	adult
1	1	1	female	38.0	1	0	71.2833		C	C	adult
2	1	3	female	26.0	0	0	7.9250		S	NaN	adult
3	1	1	female	35.0	1	0	53.1000		S	C	adult
4	0	3	male	35.0	0	0	8.0500		S	NaN	adult

```
In [676...]: titanic.ad_chi.value_counts()
```

```
Out[676...]:
```

adult	778
child	113

```
Name: ad_chi, dtype: int64
```

```
In [677...]: titanic.head(20)
```

```
Out[677...]:
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck	ad_chi	
0	0	3	male	22.0	1	0	7.2500		S	NaN	adult
1	1	1	female	38.0	1	0	71.2833		C	C	adult
2	1	3	female	26.0	0	0	7.9250		S	NaN	adult
3	1	1	female	35.0	1	0	53.1000		S	C	adult
4	0	3	male	35.0	0	0	8.0500		S	NaN	adult
5	0	3	male	NaN	0	0	8.4583		Q	NaN	adult

	survived	pclass	sex	age	sibsp	parch	fare	embarked	deck	ad_chi
6	0	1	male	54.0	0	0	51.8625	S	E	adult
7	0	3	male	2.0	3	1	21.0750	S	NaN	child
8	1	3	female	27.0	0	2	11.1333	S	NaN	adult
9	1	2	female	14.0	1	0	30.0708	C	NaN	child
10	1	3	female	4.0	1	1	16.7000	S	G	child
11	1	1	female	58.0	0	0	26.5500	S	C	adult
12	0	3	male	20.0	0	0	8.0500	S	NaN	adult
13	0	3	male	39.0	1	5	31.2750	S	NaN	adult
14	0	3	female	14.0	0	0	7.8542	S	NaN	child
15	1	2	female	55.0	0	0	16.0000	S	NaN	adult
16	0	3	male	2.0	4	1	29.1250	Q	NaN	child
17	1	2	male	NaN	0	0	13.0000	S	NaN	adult
18	0	3	female	31.0	1	0	18.0000	S	NaN	adult
19	1	3	female	NaN	0	0	7.2250	C	NaN	adult

From .groupby() statistics can be gathered to see how women and children compared on surviving the titanic.

In [678...]: titanic.ad_chi.value_counts()

Out[678...]:

adult	778
child	113
Name: ad_chi, dtype: int64	

In [679...]: titanic.groupby("ad_chi").survived.mean()

Out[679...]:

ad_chi	
adult	0.361183
child	0.539823
Name: survived, dtype: float64	

In [680...]: titanic.groupby(["sex", "ad_chi"]).survived.count()

Out[680...]:

sex	ad_chi	
female	adult	259
	child	55
male	adult	519
	child	58
Name: survived, dtype: int64		

Important

We are able to see how children survival rates differed on the titanic.

Inferences can be made on how gender and sex played a part in salvation fromt the sinking.

```
In [681...]: titanic.groupby(["sex", "ad_chi"]).survived.mean().sort_values(ascending = False)
```

```
Out[681...]:
```

sex	ad_chi	survived
female	adult	0.752896
	child	0.690909
male	child	0.396552
	adult	0.165703

```
Name: survived, dtype: float64
```

```
In [685...]: w_and_c_first = titanic.groupby(["sex", "ad_chi"]).survived.mean().sort_values(ascending = False)
```

```
In [697...]: w_and_c_first.plot(kind = "bar", figsize = (14,8), fontsize = 14, rot = 0)
```

```
plt.xlabel("Groups", fontsize = 13)
```

```
plt.ylabel("Survival Rate", fontsize = 13)
```

```
plt.title("Titanic Survival Rate by Sex/Age-Groups", fontsize = 16)
```

```
plt.show()
```

Groups	Survival Rate
(female, adult)	0.75
(female, child)	0.69
(male, child)	0.40
(male, adult)	0.17

Hierarchical Indexing (Multi-Index) with Groupby

```
In [699...]: import pandas as pd
```

```
usecols = [list_of_characteristics_to_use]
```

```
In [710...]: titanic = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\titanic.csv")
```

```
In [711...]:
```

```
titanic
```

```
Out[711...]
```

	survived	pclass	sex	age	fare
0	0	3	male	22.0	7.2500
1	1	1	female	38.0	71.2833
2	1	3	female	26.0	7.9250
3	1	1	female	35.0	53.1000
4	0	3	male	35.0	8.0500
...
886	0	2	male	27.0	13.0000
887	1	1	female	19.0	30.0000
888	0	3	female	NaN	23.4500
889	1	1	male	26.0	30.0000
890	0	3	male	32.0	7.7500

891 rows × 5 columns

Multiindex is created when you create a .groupby() with a sorter such as .mean() attached to it.

```
In [712...]
```

```
summary = titanic.groupby(["sex", "pclass"]).mean()
```

```
In [713...]
```

```
summary
```

```
Out[713...]
```

	survived	age	fare
sex	pclass		
female	1	0.968085	34.611765
	2	0.921053	28.722973
	3	0.500000	21.750000
male	1	0.368852	41.281386
	2	0.157407	30.740707
	3	0.135447	26.507589

.index tells the user what type of index it is. As shown below it is a multiindex.

```
In [714...]
```

```
summary.index
```

```
Out[714...]
```

```
MultiIndex([( 'female', 1),  
            ( 'female', 2),  
            ( 'female', 3),  
            ( 'male', 1),  
            ( 'male', 2),
```

```
( 'male', 3)],  
names=['sex', 'pclass'])
```

Can sort the rows and columns.

```
In [715... summary.loc[("female", 2), :]
```

```
Out[715... survived      0.921053  
age          28.722973  
fare         21.970121  
Name: (female, 2), dtype: float64
```

```
In [717... summary.loc[("female", 2), "age"]
```

```
Out[717... 28.722972972972972
```

```
In [718... summary.swaplevel().sort_index()
```

		survived	age	fare
pclass	sex			
1	female	0.968085	34.611765	106.125798
	male	0.368852	41.281386	67.226127
2	female	0.921053	28.722973	21.970121
	male	0.157407	30.740707	19.741782
3	female	0.500000	21.750000	16.118810
	male	0.135447	26.507589	12.661633

```
In [719... summary.reset_index()
```

	sex	pclass	survived	age	fare
0	female	1	0.968085	34.611765	106.125798
1	female	2	0.921053	28.722973	21.970121
2	female	3	0.500000	21.750000	16.118810
3	male	1	0.368852	41.281386	67.226127
4	male	2	0.157407	30.740707	19.741782
5	male	3	0.135447	26.507589	12.661633

```
In [720... summary.swaplevel()
```

		survived	age	fare
pclass	sex			
1	female	0.968085	34.611765	106.125798
2	female	0.921053	28.722973	21.970121
3	female	0.500000	21.750000	16.118810

		survived	age	fare
pclass	sex			
1	male	0.368852	41.281386	67.226127
2	male	0.157407	30.740707	19.741782
3	male	0.135447	26.507589	12.661633

```
In [721...]: summary.swaplevel().sort_index()
```

		survived	age	fare
pclass	sex			
1	female	0.968085	34.611765	106.125798
	male	0.368852	41.281386	67.226127
2	female	0.921053	28.722973	21.970121
	male	0.157407	30.740707	19.741782
3	female	0.500000	21.750000	16.118810
	male	0.135447	26.507589	12.661633

Reset index makes the index singular again. It becomes a conventional number.

```
In [727...]: summary.reset_index()
```

	sex	pclass	survived	age	fare
0	female	1	0.968085	34.611765	106.125798
1	female	2	0.921053	28.722973	21.970121
2	female	3	0.500000	21.750000	16.118810
3	male	1	0.368852	41.281386	67.226127
4	male	2	0.157407	30.740707	19.741782
5	male	3	0.135447	26.507589	12.661633

Stack and Unstack

```
In [729...]: summer = pd.read_csv(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\summer.csv")
```

```
In [730...]: summer.head()
```

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	Gold
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Silver

Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal
2	1896	Athens	Aquatics	SWIMMING, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bronze
3	1896	Athens	Aquatics	SWIMMING, Ioannis	GRE	Men	100M Freestyle For Sailors	Gold
4	1896	Athens	Aquatics	SWIMMING, Spiridon	GRE	Men	100M Freestyle For Sailors	Silver

Group by Country and medal gets the count from the group.

```
In [735...]: medals_by_country = summer.groupby(["Country", "Medal"]).Medal.count()
```

```
In [736...]: medals_by_country
```

```
Out[736...]: Country    Medal
AFG        Bronze      2
AHO        Silver      1
ALG        Bronze      8
          Gold       5
          Silver      2
          ..
ZIM        Gold       18
          Silver      4
ZZX        Bronze     10
          Gold      23
          Silver     15
Name: Medal, Length: 347, dtype: int64
```

```
In [750...]: medals_by_country.loc[("USA", "Gold")]
```

```
Out[750...]: 2235
```

```
In [751...]: medals_by_country.shape
```

```
Out[751...]: (347,)
```

Unstack moves from inner index to the column level.

Notice the threee columns shown.

```
In [753...]: medals_by_country.unstack()
```

```
Out[753...]: Medal  Bronze  Gold  Silver
```

Country				
AFG	2.0	NaN	NaN	
AHO	NaN	NaN	1.0	
ALG	8.0	5.0	2.0	
ANZ	5.0	20.0	4.0	
ARG	91.0	69.0	99.0	
...	

```
Medal  Bronze  Gold  Silver
```

```
Country
```

Country	Medal	Bronze	Gold	Silver
VIE	NaN	NaN	2.0	
YUG	118.0	143.0	174.0	
ZAM	1.0	NaN	1.0	
ZIM	1.0	18.0	4.0	
ZZX	10.0	23.0	15.0	

147 rows × 3 columns

```
In [756...]
```

```
## Can unstake the Medal Index just change the level.  
medals_by_country.unstack(level = -2)
```

```
Out[756...]
```

Country	Medal	AFG	AHO	ALG	ANZ	ARG	ARM	AUS	AUT	AZE	BAH	...	URS	URU	USA	UZB	VEN	VIE	YI
Bronze	Bronze	2.0	NaN	8.0	5.0	91.0	8.0	472.0	44.0	15.0	5.0	...	584.0	30.0	1098.0	10.0	8.0	NaN	11
Gold	Gold	NaN	NaN	5.0	20.0	69.0	1.0	312.0	21.0	6.0	13.0	...	838.0	44.0	2235.0	5.0	2.0	NaN	14
Silver	Silver	NaN	1.0	2.0	4.0	99.0	2.0	405.0	81.0	5.0	9.0	...	627.0	2.0	1252.0	5.0	2.0	2.0	17

3 rows × 147 columns

```
In [757...]
```

```
medals_by_country.unstack(level = -1)
```

```
Out[757...]
```

Country	Medal	Bronze	Gold	Silver
AFG	AFG	2.0	NaN	NaN
AHO	AHO	NaN	NaN	1.0
ALG	ALG	8.0	5.0	2.0
ANZ	ANZ	5.0	20.0	4.0
ARG	ARG	91.0	69.0	99.0
...
VIE	VIE	NaN	NaN	2.0
YUG	YUG	118.0	143.0	174.0
ZAM	ZAM	1.0	NaN	1.0
ZIM	ZIM	1.0	18.0	4.0
ZZX	ZZX	10.0	23.0	15.0

147 rows × 3 columns

```
In [758...]
```

```
medals_by_country = medals_by_country.unstack(level = -1, fill_value = 0)
```

```
In [759...]: medals_by_country.head()
```

```
Out[759...]:
```

Country	Medal	
AFG	Bronze	2
AHO	Silver	1
ALG	Bronze	8
	Gold	5
	Silver	2

Name: Medal, dtype: int64

```
In [762...]: medals_by_country.shape
```

```
Out[762...]: (347,)
```

```
In [763...]: medals_by_country = medals_by_country[["Gold", "Silver", "Bronze"]]
```

```
-----  
KeyError                                     Traceback (most recent call last)  
Input In [763], in <cell line: 1>()  
----> 1 medals_by_country = medals_by_country[["Gold", "Silver", "Bronze"]]  
  
File ~\anaconda3\lib\site-packages\pandas\core\series.py:984, in Series.__getitem__(self, key)  
  981     key = np.asarray(key, dtype=bool)  
  982     return self._get_values(key)  
--> 984 return self._get_with(key)  
  
File ~\anaconda3\lib\site-packages\pandas\core\series.py:1024, in Series._get_with(self, key)  
 1021         return self.iloc[key]  
 1023 # handle the dup indexing case GH#4246  
-> 1024 return self.loc[key]  
  
File ~\anaconda3\lib\site-packages\pandas\core\indexing.py:967, in _LocationIndexer.__getitem__(self, key)  
  964 axis = self.axis or 0  
  966 maybe_callable = com.apply if callable(key, self.obj)  
--> 967 return self._getitem_axis(maybe_callable, axis=axis)  
  
File ~\anaconda3\lib\site-packages\pandas\core\indexing.py:1191, in _LocIndexer.__getitem_axis(self, key, axis)  
 1188     if hasattr(key, "ndim") and key.ndim > 1:  
 1189         raise ValueError("Cannot index with multidimensional key")  
-> 1191     return self._getitem_iterable(key, axis=axis)  
 1193 # nested tuple slicing  
 1194 if is_nested_tuple(key, labels):  
  
File ~\anaconda3\lib\site-packages\pandas\core\indexing.py:1132, in _LocIndexer.__getitem_iterable(self, key, axis)  
 1129 self._validate_key(key, axis)  
 1131 # A collection of keys  
-> 1132 keyarr, indexer = self._get_listlike_indexer(key, axis)  
 1133 return self.obj._reindex_with_indexers(  
 1134     {axis: [keyarr, indexer]}, copy=True, allow_dups=True  
 1135 )  
  
File ~\anaconda3\lib\site-packages\pandas\core\indexing.py:1327, in _LocIndexer._get_listlike_indexer(self, key, axis)  
 1324 ax = self.obj._get_axis(axis)  
 1325 axis_name = self.obj.get_axis_name(axis)  
-> 1327 keyarr, indexer = ax._get_indexer_strict(key, axis_name)  
 1329 return keyarr, indexer
```

```

File ~\anaconda3\lib\site-packages\pandas\core\indexes\multi.py:2589, in MultiIndex._get_indexer_strict(self, key, axis_name)
    2586     if len(keyarr) and not isinstance(keyarr[0], tuple):
    2587         indexer = self.get_indexer_level_0(keyarr)
-> 2589         self._raise_if_missing(key, indexer, axis_name)
    2590     return self[indexer], indexer
    2592 return super().___get_indexer_strict(key, axis_name)

File ~\anaconda3\lib\site-packages\pandas\core\indexes\multi.py:2607, in MultiIndex._raise_if_missing(self, key, indexer, axis_name)
    2605     cmask = check == -1
    2606     if cmask.any():
-> 2607         raise KeyError(f"{keyarr[cmask]} not in index")
    2608     # We get here when levels still contain values which are not
    2609     # actually in Index anymore
    2610     raise KeyError(f"{keyarr} not in index")

```

KeyError: "['Gold' 'Silver' 'Bronze'] not in index"

Line 19 the sort values are very important it describes the tie breakers for gold if tied.

Note that the ascending order has to be false in order to override the default settings.



```

ARG 69 99 91

In [16]: medals_by_country.shape
Out[16]: (147, 3)

In [17]: medals_by_country = medals_by_country[["Gold", "Silver", "Bronze"]]

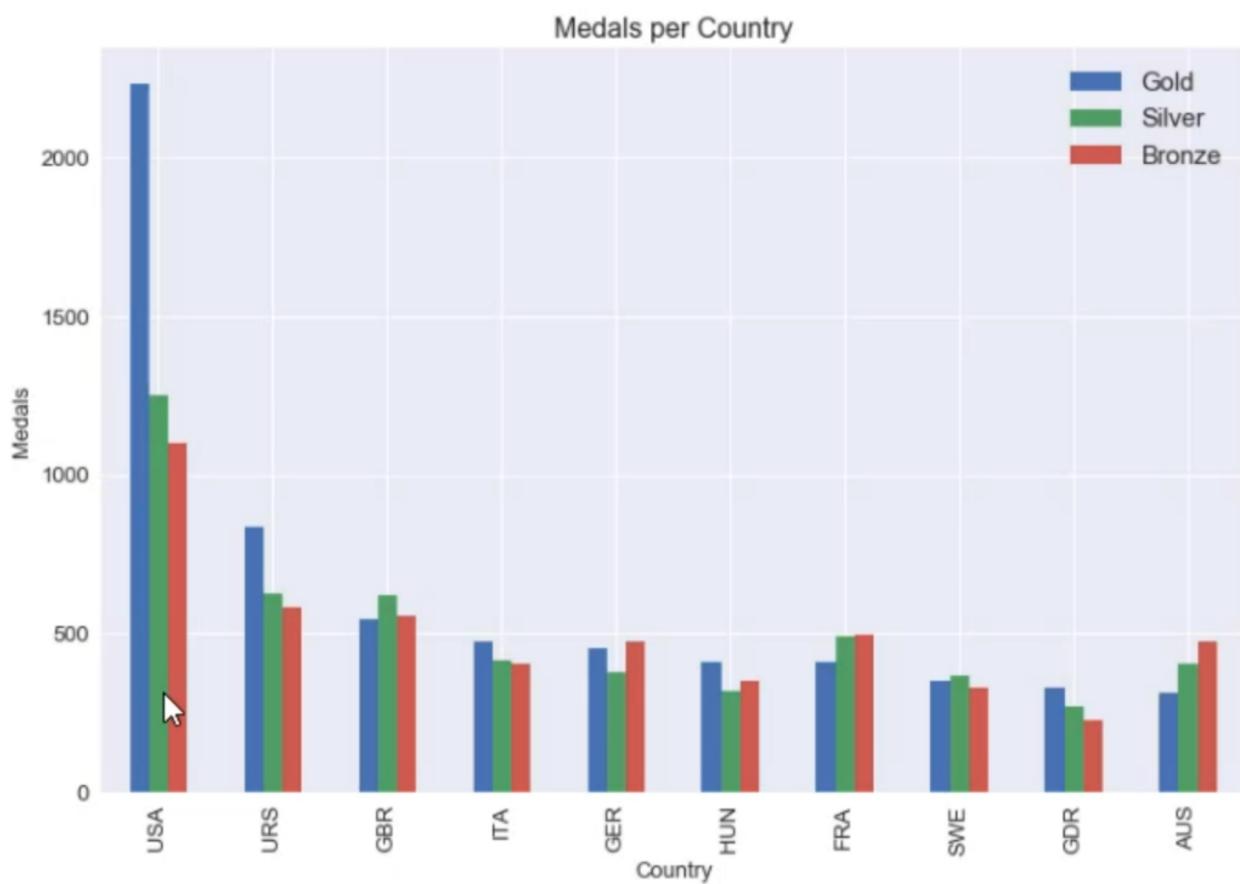
In [19]: medals_by_country.sort_values(by = ["Gold", "Silver", "Bronze"], ascending = [False, False, False], inplace = True)

In [ ]: medals_by_country.head(10)

In [ ]: import matplotlib.pyplot as plt
plt.style.use("seaborn")

In [ ]: medals_by_country.head(10).plot(kind = "bar", figsize = (12,8), fontsize = 13)
plt.xlabel("Country", fontsize = 13)
plt.ylabel("Medals", fontsize = 13)
plt.title("Medals per Country", fontsize = 16)
plt.legend(fontsize = 15)
plt.show()

```



Stack recreates the indexing hierarchy that is undone.

In [24]: medals_by_country.stack().unstack()

Out[24]:

Country	Medal	Count
USA	Gold	2235
	Silver	1252
	Bronze	1098
URS	Gold	838
	Silver	627
	Bronze	584
GBR	Gold	546
	Silver	621
	Bronze	553
ITA	Gold	476
	Silver	416
	Bronze	404
GER	Gold	452
	Silver	378
	Bronze	475
HUN	Gold	412
	Silver	316
	Bronze	351

In []:

In []:

In []:

In []:

Get access to Yaho finance

Update the conda by typing "conda update anaconda" in the Anaconda Prompt.

Next type in: "pip install yfinance" into the Anaconda Prompt

Import yahoo finance

`import yfinance as yf`

In [4]:

```
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
```

In [13]:

```
# A list of ticker symbols for stocks can be made.
tickers = ["AAPL", "BA", "KO", "IBM", "DIS", "MSFT"]
```

Downloading stock information

Stock information can be obtained using:

```
yf.download("ticker_ortickerlist", start = "start_date", end = "end_date")
```

In [10]:

```
# keyword period can be entered to define the range of stocks that should be obtained.
# Shift + tab can be used to show the other settings.
yf.download("AAPL", start = "2010-01-01", end = "2019-02-06").head(5)
```

```
[*****100%*****] 1 of 1 completed
```

Out[10]:

	Open	High	Low	Close	Adj Close	Volume
Date						
2010-01-04	7.622500	7.660714	7.585000	7.643214	6.544688	493729600
2010-01-05	7.664286	7.699643	7.616071	7.656429	6.556005	601904800
2010-01-06	7.656429	7.686786	7.526786	7.534643	6.451722	552160000
2010-01-07	7.562500	7.571429	7.466071	7.520714	6.439795	477131200
2010-01-08	7.510714	7.571429	7.466429	7.570714	6.482609	447610800

Note: The adjusted close price includes the dividend payment.

The volume is measured in the amount of stocks traded.

Using the `yf.download()` method returns the:

"Open", "High", "Low", "Close", "Adj Close", "Volume"

These are given in a tabular format for business day within the range of dates. No weekends nor holidays.

A list of tickers can be added into yf.download()

In [17]:

```
stocks = yf.download(tickers, start = "2010-01-01", end = "2019-02-06")
```

```
[*****100%*****] 6 of 6 completed
```

In [18]:

```
stocks.head()
```

Out[18]:

	Adj Close										Close
	AAPL	BA	DIS	IBM	KO	MSFT	AAPL	BA	DIS	IBM	
Date											
2010-01-04	6.544689	43.777542	27.933924	82.858467	19.496908	23.855656	7.643214	56.180000	32.070000	126.625237	
2010-01-05	6.556003	45.211342	27.864239	81.857529	19.261059	23.863369	7.656429	58.020000	31.990000	125.095604	
2010-01-06	6.451723	46.582794	27.716160	81.325775	19.254217	23.716917	7.534643	59.779999	31.820000	124.282982	
2010-01-07	6.439794	48.468548	27.724876	81.044273	19.206371	23.470268	7.520714	62.200001	31.830000	123.852776	
2010-01-08	6.482609	48.001011	27.768423	81.857529	18.850885	23.632132	7.570714	61.599998	31.879999	125.095604	

5 rows × 36 columns

In [20]:

```
stocks.tail(2)
```

Out[20]:

	Adj Close										
	AAPL	BA	DIS	IBM	KO	MSFT	AAPL	BA	DIS		
Date											
2019-02-04	41.508518	385.763672	110.450172	109.561775	44.407192	102.038948	42.812500	397.000000	111.800003	129	
2019-02-05	42.218719	398.570557	111.299789	109.853516	44.416210	103.467140	43.544998	410.179993	112.660004	129	

2 rows × 36 columns

In [24]:

```
stocks.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2288 entries, 2010-01-04 to 2019-02-05
Data columns (total 36 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   (Adj Close, AAPL)  2288 non-null   float64
 1   (Adj Close, BA)   2288 non-null   float64
 2   (Adj Close, DIS)  2288 non-null   float64
 3   (Adj Close, IBM)  2288 non-null   float64
 4   (Adj Close, KO)   2288 non-null   float64
 5   (Adj Close, MSFT) 2288 non-null   float64
 6   (Close, AAPL)    2288 non-null   float64
```

```

7 (Close, BA)      2288 non-null  float64
8 (Close, DIS)    2288 non-null  float64
9 (Close, IBM)    2288 non-null  float64
10 (Close, KO)    2288 non-null  float64
11 (Close, MSFT)  2288 non-null  float64
12 (High, AAPL)   2288 non-null  float64
13 (High, BA)    2288 non-null  float64
14 (High, DIS)   2288 non-null  float64
15 (High, IBM)   2288 non-null  float64
16 (High, KO)    2288 non-null  float64
17 (High, MSFT)  2288 non-null  float64
18 (Low, AAPL)   2288 non-null  float64
19 (Low, BA)    2288 non-null  float64
20 (Low, DIS)   2288 non-null  float64
21 (Low, IBM)   2288 non-null  float64
22 (Low, KO)    2288 non-null  float64
23 (Low, MSFT)  2288 non-null  float64
24 (Open, AAPL)  2288 non-null  float64
25 (Open, BA)   2288 non-null  float64
26 (Open, DIS)  2288 non-null  float64
27 (Open, IBM)  2288 non-null  float64
28 (Open, KO)   2288 non-null  float64
29 (Open, MSFT) 2288 non-null  float64
30 (Volume, AAPL) 2288 non-null  int64
31 (Volume, BA)  2288 non-null  int64
32 (Volume, DIS) 2288 non-null  int64
33 (Volume, IBM) 2288 non-null  int64
34 (Volume, KO)  2288 non-null  int64
35 (Volume, MSFT) 2288 non-null  int64
dtypes: float64(30), int64(6)
memory usage: 661.4 KB

```

Can convert data from yfinance to a csv

Just save the `df.download()` into a variable. After that is done use the `.to_csv("file_name")` to make a csv document.

```
In [22]: stocks.to_csv("stocks.csv")
```

Next use the `pd.read_csv()` to read the file that was just created.

```
In [26]: # Before.
pd.read_csv("stocks.csv").head(3)
```

```
Out[26]:
```

	Unnamed: 0	Adj Close	Adj Close.1	Adj Close.2	Adj Close.3	Adj Close.4
0	NaN	AAPL	BA	DIS	IBM	KO
1	Date	NaN	NaN	NaN	NaN	NaN
2	2010-01-04	6.544688701629639	43.77754211425781	27.933923721313477	82.85846710205078	19.49690818786621

3 rows × 37 columns

Defining a multiheader in `pd.read(header [line_1, line_2])`

The head is the labels that goes above the columns and the index outside of the actual results of the table.

The first header will be the ticker symbol.

The second header is the date range.

This is added by passing in a list. Note that in the example below the list of separate coordinates.

In this example Ticker Symbols and Dates will be the headers referenced by their horizontal line number.

To sort the data indexing must be done using:

```
index_col = [singular_number_of_index]
```

Exemplar of porting yfinance data from CSV pt. 1 of 2

Must remember to parse_dates with the proper index to make the dates recognizable.

```
parse_dates = ["column_of_dates_that_needs_transform"]
```

In [58]:

```
# Defining a list two headers.  
# The index is the date.  
# IMPORTANT: stocks.columns = stocks.columns_to_flat_index()  
# Used to make columns neater without all of the  
stocks = pd.read_csv("stocks.csv", header = [0,1], index_col = 0, parse_dates = [0])
```

In [59]:

```
stocks.head(4)
```

Out[59]:

	Adj Close										Close	...
	AAPL	BA	DIS	IBM	KO	MSFT	AAPL	BA	DIS	IBM	...	
Date												
2010-01-04	6.544689	43.777542	27.933924	82.858467	19.496908	23.855656	7.643214	56.180000	32.07	126.625237	...	
2010-01-05	6.556003	45.211342	27.864239	81.857529	19.261059	23.863369	7.656429	58.020000	31.99	125.095604	...	
2010-01-06	6.451723	46.582794	27.716160	81.325775	19.254217	23.716917	7.534643	59.779999	31.82	124.282982	...	
2010-01-07	6.439794	48.468548	27.724876	81.044273	19.206371	23.470268	7.520714	62.200001	31.83	123.852776	...	

4 rows × 36 columns

Confirm the index is the date using:

```
dataset.index
```

In [60]:

```
### See the index of stocks.
```

```
stocks.index
```

```
Out[60]: DatetimeIndex(['2010-01-04', '2010-01-05', '2010-01-06', '2010-01-07',  
   '2010-01-08', '2010-01-11', '2010-01-12', '2010-01-13',  
   '2010-01-14', '2010-01-15',  
   ...  
   '2019-01-23', '2019-01-24', '2019-01-25', '2019-01-28',  
   '2019-01-29', '2019-01-30', '2019-01-31', '2019-02-01',  
   '2019-02-04', '2019-02-05'],  
  dtype='datetime64[ns]', name='Date', length=2288, freq=None)
```

Now iloc can be used to locate indexes.

```
In [61]:
```

```
# Here is an example:  
stocks.loc[["2010-01-04"]]
```

```
Out[61]:
```

							Adj Close						Close	...	
	AAPL	BA	DIS	IBM	KO	MSFT	AAPL	BA	DIS	IBM	...	IBM	...	DIS	
Date															
2010-01-04	6.544689	43.777542	27.933924	82.858467	19.496908	23.855656	7.643214	56.18	32.07	126.625237	...	32.5			

1 rows × 36 columns

```
In [62]:
```

```
stocks.head()
```

```
Out[62]:
```

							Adj Close						Close		
	AAPL	BA	DIS	IBM	KO	MSFT	AAPL	BA	DIS	IBM	...	IBM	...	IBM	
Date															
2010-01-04	6.544689	43.777542	27.933924	82.858467	19.496908	23.855656	7.643214	56.180000	32.070000	126.625237	...	32.5	...	32.5	
2010-01-05	6.556003	45.211342	27.864239	81.857529	19.261059	23.863369	7.656429	58.020000	31.990000	125.095604	...	32.5	...	32.5	
2010-01-06	6.451723	46.582794	27.716160	81.325775	19.254217	23.716917	7.534643	59.779999	31.820000	124.282982	...	32.5	...	32.5	
2010-01-07	6.439794	48.468548	27.724876	81.044273	19.206371	23.470268	7.520714	62.200001	31.830000	123.852776	...	32.5	...	32.5	
2010-01-08	6.482609	48.001011	27.768423	81.857529	18.850885	23.632132	7.570714	61.599998	31.879999	125.095604	...	32.5	...	32.5	

5 rows × 36 columns

Exemplar of porting yfinance data from CSV pt. 2 of 2

Make the characteristic title easier to read. Makes the column index flat.

Here is the last step of perfecting the table. Make the chart easier to read by making the columns on one tier, instead of two tiers.

General code setup for flattening the index.

Use `.to_flat_index()`

```
dataset.columns = dataset.columns.to_flat_index()
```

In [65]:

```
# The implementation
stocks.columns = stocks.columns.to_flat_index()
```

In [66]:

```
# A perfected chart.
stocks.head()
```

Out[66]:

	(Adj Close, AAPL)	(Adj Close, BA)	(Adj Close, DIS)	(Adj Close, IBM)	(Adj Close, KO)	(Adj Close, MSFT)	(Close, AAPL)	(Close, BA)	(Close, DIS)	(Close, IBM)
Date										
2010-01-04	6.544689	43.777542	27.933924	82.858467	19.496908	23.855656	7.643214	56.180000	32.070000	126.625237
2010-01-05	6.556003	45.211342	27.864239	81.857529	19.261059	23.863369	7.656429	58.020000	31.990000	125.095604
2010-01-06	6.451723	46.582794	27.716160	81.325775	19.254217	23.716917	7.534643	59.779999	31.820000	124.282982
2010-01-07	6.439794	48.468548	27.724876	81.044273	19.206371	23.470268	7.520714	62.200001	31.830000	123.852776
2010-01-08	6.482609	48.001011	27.768423	81.857529	18.850885	23.632132	7.570714	61.599998	31.879999	125.095604

5 rows × 36 columns

Revert back to Multi-Index

In [68]:

```
## To revert back to the multindex
stocks.columns = pd.MultiIndex.from_tuples(stocks)
```

In [69]:

```
stocks.head()
```

Out[69]:

	Adj Close						Close			
	AAPL	BA	DIS	IBM	KO	MSFT	AAPL	BA	DIS	IBM
Date										
2010-01-04	6.544689	43.777542	27.933924	82.858467	19.496908	23.855656	7.643214	56.180000	32.070000	126.625237
2010-01-05	6.556003	45.211342	27.864239	81.857529	19.261059	23.863369	7.656429	58.020000	31.990000	125.095604

	Adj Close										Close
	AAPL	BA	DIS	IBM	KO	MSFT	AAPL	BA	DIS	IBM	
Date											
2010-01-06	6.451723	46.582794	27.716160	81.325775	19.254217	23.716917	7.534643	59.779999	31.820000	124.282982	
2010-01-07	6.439794	48.468548	27.724876	81.044273	19.206371	23.470268	7.520714	62.200001	31.830000	123.852776	
2010-01-08	6.482609	48.001011	27.768423	81.857529	18.850885	23.632132	7.570714	61.599998	31.879999	125.095604	

5 rows × 36 columns

Changing the axis.

The (axis = 1) allows the user to see each attribute individually. Instead of showing the company information in bulk, it shows the adjusted close in bulk.

Without sort_index()

In [76]:

```
# Shows the companies' information with the adjusted close individually. Instead of grouping stocks.
stocks.swaplevel(axis = 1).head(3).sort_index(axis = 1)
```

Out[76]:

	AAPL	BA	DIS	IBM	KO	MSFT	AAPL	BA	DIS	IBM	...
	Adj Close	Close	Close	Close	Close	...					
Date											
2010-01-04	6.544689	43.777542	27.933924	82.858467	19.496908	23.855656	7.643214	56.180000	32.07	126.625237	...
2010-01-05	6.556003	45.211342	27.864239	81.857529	19.261059	23.863369	7.656429	58.020000	31.99	125.095604	...
2010-01-06	6.451723	46.582794	27.716160	81.325775	19.254217	23.716917	7.534643	59.779999	31.82	124.282982	...

3 rows × 36 columns

With Sort Index

In [75]:

```
stocks.swaplevel(axis = 1).sort_index(axis = 1).head(3)
```

Out[75]:

	AAPL										BA	...
	Adj Close	Close	High	Low	Open	Volume	Adj Close	Close	High	Low	...	
Date												
2010-01-04	6.544689	7.643214	7.660714	7.585000	7.622500	493729600	43.777542	56.180000	56.389999	54.799999	...	

		AAPL									BA	...
	Adj Close	Close	High	Low	Open	Volume	Adj Close	Close	High	Low	...	
Date												
2010-01-05	6.556003	7.656429	7.699643	7.616071	7.664286	601904800	45.211342	58.020000	58.279999	56.000000	...	
2010-01-06	6.451723	7.534643	7.686786	7.526786	7.656429	552160000	46.582794	59.779999	59.990002	57.880001	...	

3 rows × 36 columns

New Section

Initial Inspection, and Visualization

In [78]:

```
import pandas as pd
```

In [79]:

```
# Dates is row 0 Ticker is row 1 becomes the header
# Sort by date so you use index_col = 1
# The date is also parsed through in order to make it a dateobject instead of a string.
stocks = pd.read_csv("stocks.csv", header = [0,1], index_col = 0, parse_dates = [0])
```

In [80]:

```
stocks.head()
```

Out[80]:

							Adj Close					Close
	AAPL	BA	DIS	IBM	KO	MSFT	AAPL	BA	DIS	IBM		
Date												
2010-01-04	6.544689	43.777542	27.933924	82.858467	19.496908	23.855656	7.643214	56.180000	32.070000	126.625237	...	
2010-01-05	6.556003	45.211342	27.864239	81.857529	19.261059	23.863369	7.656429	58.020000	31.990000	125.095604	...	
2010-01-06	6.451723	46.582794	27.716160	81.325775	19.254217	23.716917	7.534643	59.779999	31.820000	124.282982	...	
2010-01-07	6.439794	48.468548	27.724876	81.044273	19.206371	23.470268	7.520714	62.200001	31.830000	123.852776	...	
2010-01-08	6.482609	48.001011	27.768423	81.857529	18.850885	23.632132	7.570714	61.599998	31.879999	125.095604	...	

5 rows × 36 columns

In [81]:

```
stocks.tail()
```

Out[81]:

							Adj Close				
	AAPL	BA	DIS	IBM	KO	MSFT	AAPL	BA	DIS	IBM	
Date											
2010-01-04	6.544689	43.777542	27.933924	82.858467	19.496908	23.855656	7.643214	56.180000	32.070000	126.625237	...
2010-01-05	6.556003	45.211342	27.864239	81.857529	19.261059	23.863369	7.656429	58.020000	31.990000	125.095604	...
2010-01-06	6.451723	46.582794	27.716160	81.325775	19.254217	23.716917	7.534643	59.779999	31.820000	124.282982	...
2010-01-07	6.439794	48.468548	27.724876	81.044273	19.206371	23.470268	7.520714	62.200001	31.830000	123.852776	...
2010-01-08	6.482609	48.001011	27.768423	81.857529	18.850885	23.632132	7.570714	61.599998	31.879999	125.095604	...

	AAPL	BA	DIS	IBM	KO	MSFT	AAPL	BA	DIS	Adj Close
Date										
2019-01-30	40.054214	376.746307	108.800331	108.905334	43.153873	102.656548	41.312500	387.720001	110.129997	128
2019-01-31	40.342648	374.705688	110.173546	108.937752	43.397327	100.774788	41.610001	385.619995	111.519997	128
2019-02-01	40.362049	376.464508	109.956207	108.678413	43.911274	99.182564	41.630001	387.429993	111.300003	128
2019-02-04	41.508518	385.763672	110.450172	109.561775	44.407192	102.038948	42.812500	397.000000	111.800003	129
2019-02-05	42.218719	398.570557	111.299789	109.853516	44.416210	103.467140	43.544998	410.179993	112.660004	129

5 rows × 36 columns

Use info to gauge the what the column is measuring and the type it takes.

dataset.info gives all of the types of the characteristics/columns and it of course lists them.

In [87]:

`stocks.info()`

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2288 entries, 2010-01-04 to 2019-02-05
Data columns (total 36 columns):
 #  Column           Non-Null Count  Dtype  
--- 
 0  (Adj Close, AAPL)  2288 non-null   float64
 1  (Adj Close, BA)   2288 non-null   float64
 2  (Adj Close, DIS)  2288 non-null   float64
 3  (Adj Close, IBM)  2288 non-null   float64
 4  (Adj Close, KO)   2288 non-null   float64
 5  (Adj Close, MSFT) 2288 non-null   float64
 6  (Close, AAPL)    2288 non-null   float64
 7  (Close, BA)      2288 non-null   float64
 8  (Close, DIS)    2288 non-null   float64
 9  (Close, IBM)    2288 non-null   float64
 10 (Close, KO)     2288 non-null   float64
 11 (Close, MSFT)   2288 non-null   float64
 12 (High, AAPL)   2288 non-null   float64
 13 (High, BA)     2288 non-null   float64
 14 (High, DIS)   2288 non-null   float64
 15 (High, IBM)   2288 non-null   float64
 16 (High, KO)     2288 non-null   float64
 17 (High, MSFT)   2288 non-null   float64
 18 (Low, AAPL)   2288 non-null   float64
 19 (Low, BA)     2288 non-null   float64
 20 (Low, DIS)   2288 non-null   float64
 21 (Low, IBM)   2288 non-null   float64
 22 (Low, KO)     2288 non-null   float64
 23 (Low, MSFT)   2288 non-null   float64
 24 (Open, AAPL)  2288 non-null   float64
 25 (Open, BA)    2288 non-null   float64
 26 (Open, DIS)  2288 non-null   float64
 27 (Open, IBM)  2288 non-null   float64
 28 (Open, KO)    2288 non-null   float64
 29 (Open, MSFT) 2288 non-null   float64
```

```

30 (Volume, AAPL)      2288 non-null  int64
31 (Volume, BA)        2288 non-null  int64
32 (Volume, DIS)       2288 non-null  int64
33 (Volume, IBM)       2288 non-null  int64
34 (Volume, KO)        2288 non-null  int64
35 (Volume, MSFT)      2288 non-null  int64
dtypes: float64(30), int64(6)
memory usage: 661.4 KB

```

stock.describe() Describes all of the columns.

Consider that .describe() summarizes statistics of all of the characteristics that is measured.

In [88]: `stocks.describe()`

Out[88]:

	Adj Close									
	AAPL	BA	DIS	IBM	KO	MSFT	AAPL	BA	DIS	IBM
count	2288.000000	2288.000000	2288.000000	2288.000000	2288.000000	2288.000000	2288.000000	2288.000000	2288.000000	2288.000000
mean	22.940784	130.186519	71.644731	114.176664	31.195599	43.601609	25.010675	144.103820	7.7	2288.000000
std	11.686973	88.687438	29.810688	15.262236	6.580685	24.889400	11.723928	88.320854	2.2	2288.000000
min	5.873125	43.777542	25.536152	76.565094	17.258005	17.898998	6.858929	56.180000	2.2	2288.000000
25%	13.469302	60.522992	38.754951	106.466381	26.723810	23.379728	15.426250	73.755001	4.4	2288.000000
50%	21.384418	111.242420	79.027561	116.460384	31.761055	36.456238	23.747499	127.639999	8.8	2288.000000
75%	28.833540	135.417145	99.533241	125.895844	36.195435	54.180529	31.697500	152.267502	10.10	2288.000000
max	56.054825	398.570557	116.560318	143.238144	45.444115	111.083595	58.017502	410.179993	12.12	2288.000000

8 rows × 36 columns

The .loc and .copy() explained

.loc[] is locating all of the dates available and it is looking at the close column. That is determined by the information that is put in the brackets of course.

.copy() is very important.

It allows the actually columns of the data points that is being tapped into to not overwrite the main datasource.

In [93]: `close = stocks.loc[:, "Close"].copy()`

In [94]: `close.head()`

Out[94]:

	AAPL	BA	DIS	IBM	KO	MSFT
Date						
2010-01-04	7.643214	56.180000	32.070000	126.625237	28.520000	30.950001
2010-01-05	7.656429	58.020000	31.990000	125.095604	28.174999	30.959999
2010-01-06	7.534643	59.779999	31.820000	124.282982	28.165001	30.770000

Date

2010-01-07	7.520714	62.200001	31.830000	123.852776	28.094999	30.450001
2010-01-08	7.570714	61.599998	31.879999	125.095604	27.575001	30.660000

In [95]:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use("seaborn")
```

Why are only six companies being shown on the graph?

Remember the stocks were imported from a list that was used to download via `yf?` That was used to select and download the information concerning the six companies. That information was then taken and made into datachart then downloaded into a CSV then reimported.

In [102...]

```
close.plot(figsize = (15, 8), fontsize = 13)
plt.title("Closing Price over Time - Various Companies", fontsize = 20)
plt.ylabel("Price in Dollars", fontsize = 12)
plt.legend(fontsize = 13)
plt.show()
```



Super Important

Normalizing Time Series to a Base Value of 100 - Equalizing Gains

Normalizing to a Base of 100 essentially equalizes each of the stock to the base price. Making only gains tracked. The stocks will start at the same base and diverge to show equalized gains in an equalized manner.

In [105...]

```
close.head()
```

Out[105...]

	AAPL	BA	DIS	IBM	KO	MSFT
--	-------------	-----------	------------	------------	-----------	-------------

Date

2010-01-04	7.643214	56.180000	32.070000	126.625237	28.520000	30.950001
2010-01-05	7.656429	58.020000	31.990000	125.095604	28.174999	30.959999
2010-01-06	7.534643	59.779999	31.820000	124.282982	28.165001	30.770000
2010-01-07	7.520714	62.200001	31.830000	123.852776	28.094999	30.450001
2010-01-08	7.570714	61.599998	31.879999	125.095604	27.575001	30.660000

Example of normalizing a stock.

AAPL or Apple stock will be used as an example. To normalize the first data point must be set as the new zero for that stock, in this case Apple.

First find the first value.

In [107...]

```
# Using .iloc to find the first value.  
# The first value coordinate [0, 0] first closing price that was isolated.  
close.iloc[0, 0]
```

Out[107...]

7.643214225769043

Step 2

Use the first value must be used to divide every other value.

Use the .div() method to divide all by the original that was found using iloc.

In [117...]

```
close.AAPL.div(close.iloc[0, 0])#.mul(100)
```

Out[117...]

```
Date  
2010-01-04    1.000000  
2010-01-05    1.001729  
2010-01-06    0.985795  
2010-01-07    0.983973  
2010-01-08    0.990514  
...  
2019-01-30    5.405121  
2019-01-31    5.444045  
2019-02-01    5.446662  
2019-02-04    5.601374  
2019-02-05    5.697210  
Name: AAPL, Length: 2288, dtype: float64
```

Step 3

$$(\text{all_values}/\text{original_value}) * 100 = \text{equalized_return}$$

Multiply the divided value (all_values/original_value) * 100 in order to get the return from the original value.

Interpretation

A value > 100 means a positive return

A value = 100 means breaking even

A value < 100 means losing money.

Apple returns a few days from 2010.

```
In [120...]: close.AAPL.div(close.iloc[0,0]).mul(100).head()
```

```
Out[120...]: Date
2010-01-04    100.000000
2010-01-05    100.172893
2010-01-06    98.579511
2010-01-07    98.397266
2010-01-08    99.051443
Name: AAPL, dtype: float64
```

Apple 9 years from 2010.

Apple did a 5x return in less than a decade.

```
In [121...]: close.AAPL.div(close.iloc[0,0]).mul(100).tail()
```

```
Out[121...]: Date
2019-01-30    540.512130
2019-01-31    544.404479
2019-02-01    544.666155
2019-02-04    560.137381
2019-02-05    569.721022
Name: AAPL, dtype: float64
```

Applying the change for all of the stocks.

```
In [126...]: close.head(2)
```

	AAPL	BA	DIS	IBM	KO	MSFT
Date						
2010-01-04	7.643214	56.18	32.07	126.625237	28.520000	30.950001
2010-01-05	7.656429	58.02	31.99	125.095604	28.174999	30.959999

How to apply to all stocks?

Look at the components seperately and then combine.

Select the row to act as the base of the normal.

```
In [129...]: # This code will get the first value of the close of each stock on the first day.
close.iloc[0]
```

AAPL 7.643214

```
Out[129... BA      56.180000
DIS      32.070000
IBM     126.625237
KO      28.520000
MSFT    30.950001
Name: 2010-01-04 00:00:00, dtype: float64
```

```
In [135... # This code will select closing price for every piece of data in the table.
close.iloc[:,].head(2)
```

```
Out[135...          AAPL      BA      DIS      IBM      KO      MSFT
                  Date
2010-01-04  7.643214  56.18  32.07  126.625237  28.520000  30.950001
2010-01-05  7.656429  58.02  31.99  125.095604  28.174999  30.959999
```

The code that will make the code equalized. Base 1.

```
close.iloc[:,].div(close.iloc[0])
```

The code is shown again here. `close.iloc[:,].div(close.iloc[0])`

```
In [141... # Makes all of the code base one.
close.iloc[:,].div(close.iloc[0]).head(3)
```

```
Out[141...          AAPL      BA      DIS      IBM      KO      MSFT
                  Date
2010-01-04  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000
2010-01-05  1.001729  1.032752  0.997505  0.987920  0.987903  1.000323
2010-01-06  0.985795  1.064080  0.992205  0.981502  0.987553  0.994184
```

Multiply to make base 100

Just add `.mul(100)` at the end of the code.

```
close.iloc[:,].div(close.iloc[0]).mul(100)
```

```
In [143... close.iloc[:,].div(close.iloc[0]).mul(100).head(3)
```

```
Out[143...          AAPL      BA      DIS      IBM      KO      MSFT
                  Date
2010-01-04  100.000000 100.000000 100.000000 100.000000 100.000000 100.000000
2010-01-05  100.172893 103.275187 99.750546  98.792000  98.790318 100.032305
2010-01-06  98.579511 106.407972 99.220455  98.150247  98.755261  99.418416
```

Plot an equalized Base 100 graph.

Save the equalized data as a different variable, then plot it.

```
In [145... close_equalized = close.iloc[:,].div(close.iloc[0]).mul(100)
```

```
In [146...]
```

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use("seaborn")
```

```
In [163...]
```

```
# Note that the size of the dimensions for the chart was freestyled.
# It is best to stick with the more conventional values shown in other charts, do NOT emulate
close_equalized.plot(figsize = (15, 12))
plt.title("Stock Returns Over Time- Base 100 From 2015 - 2019", fontsize = 20)
plt.xlabel("Date", fontsize = 14)
plt.ylabel("Returns", fontsize = 14)
plt.legend(fontsize = 20)
plt.show()
```

Stock Returns Over Time- Base 100 From 2015 - 2019



New Section

Utilizing the shift() Method

```
In [164...]
```

```
close.head()
```

```
Out[164...]
```

	AAPL	BA	DIS	IBM	KO	MSFT
Date						
2010-01-04	7.643214	56.180000	32.070000	126.625237	28.520000	30.950001
2010-01-05	7.656429	58.020000	31.990000	125.095604	28.174999	30.959999

AAPL **BA** **DIS** **IBM** **KO** **MSFT**

Date

	AAPL	BA	DIS	IBM	KO	MSFT
2010-01-06	7.534643	59.779999	31.820000	124.282982	28.165001	30.770000
2010-01-07	7.520714	62.200001	31.830000	123.852776	28.094999	30.450001
2010-01-08	7.570714	61.599998	31.879999	125.095604	27.575001	30.660000

In [165...]

```
# Get the closing prices of AAPL and turn it into a DataFrame.  
aapl = close.AAPL.copy().to_frame()
```

In [179...]

```
type(aapl)
```

Out[179...]

```
pandas.core.frame.DataFrame
```

In [166...]

```
aapl.head()
```

Out[166...]

AAPL

Date

	AAPL
2010-01-04	7.643214
2010-01-05	7.656429
2010-01-06	7.534643
2010-01-07	7.520714
2010-01-08	7.570714

In [181...]

```
aapl.shift(periods = 1).head()
```

Out[181...]

AAPL **lag1** **Diff** **pct_change**

Date

	AAPL	lag1	Diff	pct_change
2010-01-04	NaN	NaN	NaN	NaN
2010-01-05	7.643214	NaN	NaN	NaN
2010-01-06	7.656429	7.643214	0.013215	0.172893
2010-01-07	7.534643	7.656429	-0.121786	-1.590632
2010-01-08	7.520714	7.534643	-0.013929	-0.184871

.shift(periods = n) moves an entry down a row within a column.

See how the first value within the table becomes nullified? It was due to the period of the shift being one. See another example below.

In [183...]

```
# There will be two empty rows due to the shift periods being 2.  
aapl.shift(periods = 2).head()
```

Out[183...]

AAPL lag1 Diff pct_change

Date

Date	AAPL	lag1	Diff	pct_change
2010-01-04	NaN	NaN	NaN	NaN
2010-01-05	NaN	NaN	NaN	NaN
2010-01-06	7.643214	NaN	NaN	NaN
2010-01-07	7.656429	7.643214	0.013215	0.172893
2010-01-08	7.534643	7.656429	-0.121786	-1.590632

In [168...]

```
# The data lag1 describes a .shift(periods = 1)
aapl["lag1"] = aapl.shift(periods = 1)
```

In [169...]

```
aapl.head()
```

Out[169...]

AAPL lag1

Date

Date	AAPL	lag1
2010-01-04	7.643214	NaN
2010-01-05	7.656429	7.643214
2010-01-06	7.534643	7.656429
2010-01-07	7.520714	7.534643
2010-01-08	7.570714	7.520714

The shift() method can be combined with the subtract .sub() method.

This would be used to find the change in value between rows. In this case the difference between the closing price each day.

The first day would not produce a valid output of course.

In [184...]

```
aapl.AAPL.sub(aapl.lag1).head(10)
```

Out[184...]

```
Date
2010-01-04      NaN
2010-01-05      0.013215
2010-01-06     -0.121786
2010-01-07     -0.013929
2010-01-08      0.050000
2010-01-11     -0.066785
2010-01-12     -0.085358
2010-01-13      0.104643
2010-01-14     -0.043571
2010-01-15     -0.125000
dtype: float64
```

In [185...]

```
# Creates a New column that subtracts from the cell that came before it. Shows the difference
aapl["Diff"] = aapl.AAPL.sub(aapl.lag1)
```

```
In [186... aapl.head()
```

```
Out[186... AAPL lag1 Diff pct_change
```

Date

Date	AAPL	lag1	Diff	pct_change
2010-01-04	7.643214	NaN	NaN	NaN
2010-01-05	7.656429	7.643214	0.013215	0.172893
2010-01-06	7.534643	7.656429	-0.121786	-1.590632
2010-01-07	7.520714	7.534643	-0.013929	-0.184871
2010-01-08	7.570714	7.520714	0.050000	0.664833

```
In [187... aapl.AAPL.div(aapl.lag1).sub(1).mul(100)
```

```
Out[187... Date
2010-01-04      NaN
2010-01-05    1.001729
2010-01-06    0.984094
2010-01-07    0.998151
2010-01-08    1.006648
...
2019-01-30    1.068335
2019-01-31    1.007201
2019-02-01    1.000481
2019-02-04    1.028405
2019-02-05    1.017109
Length: 2288, dtype: float64
```

Finding the percent difference using .shift(period = n)

The last value, yesterday's stock price is considered the new standard, therefore the new price is divide by the old price.

What is left is a fraction or a ratio of the new_value/ old_value with that said to see if there is a gain or loss 1 has to be subtracted from it to see what is left after the initial is taken out. (This is because the original value which becomes the denominator divided by itself is one.

Then multiply the difference by one to get the percent change.

```
In [188... aapl.AAPL.div(aapl.lag1).sub(1).mul(100)
```

```
Out[188... Date
2010-01-04      NaN
2010-01-05    0.172893
2010-01-06   -1.590632
2010-01-07   -0.184871
2010-01-08    0.664833
...
2019-01-30    6.833468
2019-01-31    0.720123
2019-02-01    0.048066
2019-02-04    2.840497
2019-02-05    1.710945
Length: 2288, dtype: float64
```

A column for percentage can easily be added.

See below. Now the reader of the graph can better gauge how the stock fluctuated overise both numerically and percentage wise. This later equalizes the metrics in a way that favors comparison.

```
In [177... aapl["pct_change"] = aapl.AAPL.div(aapl.lag1).sub(1).mul(100)
```

```
In [178... aapl.head()
```

```
Out[178...          AAPL      lag1      Diff  pct_change
               Date
2010-01-04  7.643214      NaN      NaN      NaN
2010-01-05  7.656429  7.643214  0.013215  0.172893
2010-01-06  7.534643  7.656429 -0.121786 -1.590632
2010-01-07  7.520714  7.534643 -0.013929 -0.184871
2010-01-08  7.570714  7.520714  0.050000  0.664833
```

The method diff() and pct_change()

```
In [192... aapl.head()
```

```
Out[192...          AAPL      lag1      Diff  pct_change
               Date
2010-01-04  7.643214      NaN      NaN      NaN
2010-01-05  7.656429  7.643214  0.013215  0.172893
2010-01-06  7.534643  7.656429 -0.121786 -1.590632
2010-01-07  7.520714  7.534643 -0.013929 -0.184871
2010-01-08  7.570714  7.520714  0.050000  0.664833
```

```
In [193... aapl.AAPL.diff(periods = 1)
```

```
Out[193...          Date
2010-01-04      NaN
2010-01-05  0.013215
2010-01-06 -0.121786
2010-01-07 -0.013929
2010-01-08  0.050000
...
2019-01-30  2.642502
2019-01-31  0.297501
2019-02-01  0.020000
2019-02-04  1.182499
2019-02-05  0.732498
Name: AAPL, Length: 2288, dtype: float64
```

.diff(periods = n) is a way to calculate a difference with the shift built in

Doing the multiple step process is no longer necessary.

```
In [194... aapl["Diff2"] = aapl.AAPL.diff(periods = 1)
```

```
In [195... aapl.head(10)
```

```
Out[195...          AAPL      lag1      Diff  pct_change      Diff2
Date
2010-01-04  7.643214      NaN      NaN      NaN      NaN
2010-01-05  7.656429  7.643214  0.013215  0.172893  0.013215
2010-01-06  7.534643  7.656429 -0.121786 -1.590632 -0.121786
2010-01-07  7.520714  7.534643 -0.013929 -0.184871 -0.013929
2010-01-08  7.570714  7.520714  0.050000  0.664833  0.050000
2010-01-11  7.503929  7.570714 -0.066785 -0.882147 -0.066785
2010-01-12  7.418571  7.503929 -0.085358 -1.137513 -0.085358
2010-01-13  7.523214  7.418571  0.104643  1.410553  0.104643
2010-01-14  7.479643  7.523214 -0.043571 -0.579154 -0.043571
2010-01-15  7.354643  7.479643 -0.125000 -1.671203 -0.125000
```

```
In [196... # Both methods are proven to be the same.
aapl.Diff.equals(aapl.Diff2)
```

```
Out[196... True
```

Very important.

This is a another way to calculate percent.

.pct_change(periods = n)

```
In [197... aapl["pct_change2"] = aapl.AAPL.pct_change(periods = 1).mul(100)
```

```
In [198... aapl.head()
```

```
Out[198...          AAPL      lag1      Diff  pct_change      Diff2  pct_change2
Date
2010-01-04  7.643214      NaN      NaN      NaN      NaN      NaN
2010-01-05  7.656429  7.643214  0.013215  0.172893  0.013215  0.172893
2010-01-06  7.534643  7.656429 -0.121786 -1.590632 -0.121786 -1.590632
2010-01-07  7.520714  7.534643 -0.013929 -0.184871 -0.013929 -0.184871
2010-01-08  7.570714  7.520714  0.050000  0.664833  0.050000  0.664833
```

Important

Resampling to calculate other intervals.

In this case the resample method can be used to find the changes percent wise over a longer time period.

Add a B for business.

```
In [203...]: aapl.AAPL.resample("BM").last()
```



```
Out[203...]: Date
2010-01-29    6.859286
2010-02-26    7.307857
2010-03-31    8.392857
2010-04-30    9.324643
2010-05-31    9.174286
Freq: BM, Name: AAPL, dtype: float64
```

Very important resampling and calculating percent Returns

```
In [204...]: # The calculations are diff than the instructor but the method is sound.
aapl.AAPL.resample("M").last().pct_change(periods = 1).mul(100)
```



```
Out[204...]: Date
2010-01-31      NaN
2010-02-28    6.539620
2010-03-31   14.847028
2010-04-30   11.102138
2010-05-31   -1.612472
...
2018-10-31   -3.047756
2018-11-30  -18.404459
2018-12-31  -11.669838
2019-01-31   5.515403
2019-02-28   4.650319
Freq: M, Name: AAPL, Length: 110, dtype: float64
```

New Section

Measuring Stock Performance with Mean Return and STD of Returns

```
In [205...]: import numpy as np
```

```
In [206...]: aapl = close.AAPL.copy().to_frame()
```

```
In [207...]: aapl.head()
```

```
Out[207...]: AAPL
```

Date

AAPL

Date

Date	
2010-01-04	7.643214
2010-01-05	7.656429
2010-01-06	7.534643
2010-01-07	7.520714
2010-01-08	7.570714

In [216...]

```
# ret is the daily returns due to the formula.
ret = aapl.pct_change().dropna()
ret.head()
```

Out[216...]

AAPL

Date

Date	
2010-01-05	0.001729
2010-01-06	-0.015906
2010-01-07	-0.001849
2010-01-08	0.006648
2010-01-11	-0.008821

In [210...]

```
ret.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2287 entries, 2010-01-05 to 2019-02-05
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  --   --   --   --   --   --   -- 
 0   AAPL    2287 non-null   float64 
dtypes: float64(1)
memory usage: 35.7 KB
```

In [211...]

```
ret.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2287 entries, 2010-01-05 to 2019-02-05
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  --   --   --   --   --   --   -- 
 0   AAPL    2287 non-null   float64 
dtypes: float64(1)
memory usage: 35.7 KB
```

Use Histograms to chart returns

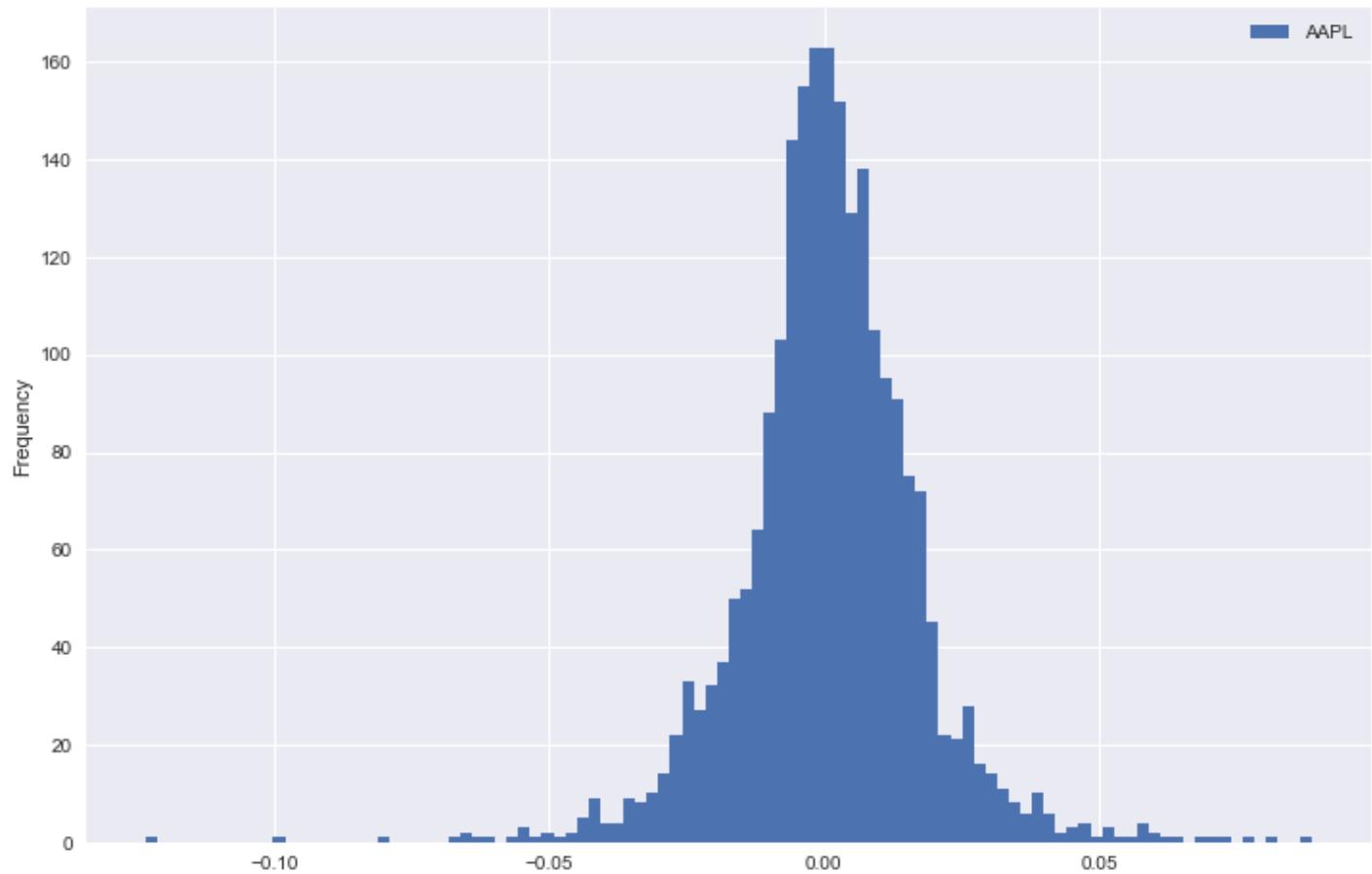
The mean return is how much the investment returns a day. Volatility is the range of the returns. Investors want high returns with low risk but it is hard for them to receive both in an investment. High returns tend to be highly volatile. Low volatility tend to give low returns. Example would be looking at bank savings, vs. bonds, vs. stocks, vs. crypto.

```
.plot(kind = "hist")
```

Histograms can be used to measure how many times a value was returned. In this instance this histogram shows how many time a return of a certain value was recorded.

In [213...]

```
ret.plot(kind = "hist", figsize = (12, 8), bins = 100)  
plt.show()
```



Attach `.mean()` to `.pct_return()` for powerful results.

Can use statistics on the daily returns.

Including mean, variance, and standard deviation.

In [215...]

```
# Finding the mean.  
daily_mean_Return = ret.mean()  
daily_mean_Return
```

Out[215...]

```
AAPL      0.000896  
dtype: float64
```

In [218...]

```
# Finding the variance.  
var_daily_Returns = ret.var()  
var_daily_Returns
```

Out[218...]

```
AAPL      0.000269  
dtype: float64
```

Square root of variance to find the variance

```
In [219...]
```

```
# Finding the variance.  
std_daily_Returns = np.sqrt(var_daily_Returns)  
std_daily_Returns
```

```
Out[219...]
```

AAPL	0.016394
	dtype: float64

Finding Standard Deviation via .std()

```
In [221...]
```

```
ret.std()
```

```
Out[221...]
```

AAPL	0.016394
	dtype: float64

Find annual return by using multiplying daily statistic by 252 business days.

The daily statistic was found by using .pct_change()

The mean of the percent wil give the APR

This would be given after multiplied by 100.

```
In [223...]
```

```
ann_mean_Return = ret.mean() * 252  
ann_mean_Return
```

```
Out[223...]
```

AAPL	0.225708
	dtype: float64

The variance will measure how much the APR will vary.

For the stock.

```
In [232...]
```

```
ann_var_Return = ret.var()  
ann_var_Return
```

```
Out[232...]
```

AAPL	0.000269
	dtype: float64

```
In [233...]
```

```
np.sqrt(ann_var_Return)
```

```
Out[233...]
```

AAPL	0.016394
	dtype: float64

Standard deviation tells you how much the data varies.

```
In [230...]
```

```
ann_std_Returns = ret.std()  
ann_std_Returns
```

```
Out[230...]
```

AAPL	0.016394
	dtype: float64

New Section

Financial Time Series - Return and Risk

In [234...]

```
import numpy as np
```

Charts does not show the risk that comes with the return.

In [239...]

```
close_equalized.plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```



In [241...]

```
close.head()
```

Out[241...]

	AAPL	BA	DIS	IBM	KO	MSFT
--	------	----	-----	-----	----	------

Date

2010-01-04	7.643214	56.180000	32.070000	126.625237	28.520000	30.950001
2010-01-05	7.656429	58.020000	31.990000	125.095604	28.174999	30.959999
2010-01-06	7.534643	59.779999	31.820000	124.282982	28.165001	30.770000
2010-01-07	7.520714	62.200001	31.830000	123.852776	28.094999	30.450001
2010-01-08	7.570714	61.599998	31.879999	125.095604	27.575001	30.660000

The code isolated that gets the percentage change with no missing values. close.pct_change().dropna()

In [245...]

```
# Drops all additional rows that does not show the percent change.
close.pct_change().dropna().head()
```

Out[245...]

	AAPL	BA	DIS	IBM	KO	MSFT
--	------	----	-----	-----	----	------

Date

2010-01-05	0.001729	0.032752	-0.002495	-0.012080	-0.012097	0.000323
------------	----------	----------	-----------	-----------	-----------	----------

AAPL **BA** **DIS** **IBM** **KO** **MSFT**

Date

	AAPL	BA	DIS	IBM	KO	MSFT
2010-01-06	-0.015906	0.030334	-0.005314	-0.006496	-0.000355	-0.006137
2010-01-07	-0.001849	0.040482	0.000314	-0.003462	-0.002485	-0.010400
2010-01-08	0.006648	-0.009646	0.001571	0.010035	-0.018509	0.006897
2010-01-11	-0.008821	-0.011851	-0.016311	-0.010470	0.020308	-0.012720

In [246...]

```
ret = close.pct_change().dropna()
```

In [247...]

```
ret.head()
```

Out[247...]

AAPL **BA** **DIS** **IBM** **KO** **MSFT**

Date

	AAPL	BA	DIS	IBM	KO	MSFT
2010-01-05	0.001729	0.032752	-0.002495	-0.012080	-0.012097	0.000323
2010-01-06	-0.015906	0.030334	-0.005314	-0.006496	-0.000355	-0.006137
2010-01-07	-0.001849	0.040482	0.000314	-0.003462	-0.002485	-0.010400
2010-01-08	0.006648	-0.009646	0.001571	0.010035	-0.018509	0.006897
2010-01-11	-0.008821	-0.011851	-0.016311	-0.010470	0.020308	-0.012720

In [252...]

```
ret.describe()
```

Out[252...]

AAPL **BA** **DIS** **IBM** **KO** **MSFT**

count	2287.000000	2287.000000	2287.000000	2287.000000	2287.000000	2287.000000
mean	0.000896	0.000990	0.000636	0.000088	0.000281	0.000650
std	0.016394	0.015488	0.013139	0.012429	0.009207	0.014579
min	-0.123558	-0.089290	-0.091708	-0.082790	-0.060291	-0.113995
25%	-0.007070	-0.007180	-0.005795	-0.005999	-0.004522	-0.006901
50%	0.000759	0.001100	0.000755	0.000213	0.000441	0.000354
75%	0.009667	0.009380	0.007530	0.006392	0.005465	0.008078
max	0.088741	0.098795	0.076302	0.088645	0.056872	0.104522

.T

.T is transpose. By default it switches the columns of the rows and the characteristics. See how the characteristics for the charts switched for the two .describe() examples above and below this cell?

Transpose mean to take place.

In [253...]

```
ret.describe().T
```

Out[253...]

	count	mean	std	min	25%	50%	75%	max
AAPL	2287.0	0.000896	0.016394	-0.123558	-0.007070	0.000759	0.009667	0.088741
BA	2287.0	0.000990	0.015488	-0.089290	-0.007180	0.001100	0.009380	0.098795
DIS	2287.0	0.000636	0.013139	-0.091708	-0.005795	0.000755	0.007530	0.076302
IBM	2287.0	0.000088	0.012429	-0.082790	-0.005999	0.000213	0.006392	0.088645
KO	2287.0	0.000281	0.009207	-0.060291	-0.004522	0.000441	0.005465	0.056872
MSFT	2287.0	0.000650	0.014579	-0.113995	-0.006901	0.000354	0.008078	0.104522

In [248...]

```
# The [:] selects the whole index that is the tickers.
# The list selects the characteristics that needs to be seen.
ret.describe().T.loc[:, ["mean", "std"]]
```

Out[248...]

	mean	std
AAPL	0.000896	0.016394
BA	0.000990	0.015488
DIS	0.000636	0.013139
IBM	0.000088	0.012429
KO	0.000281	0.009207
MSFT	0.000650	0.014579

In [254...]

```
summary = ret.describe().T.loc[:, ["mean", "std"]]
```

Annualize the mean and standard deviation.

Notice that these values are getting assigned their own column also.

In [257...]

```
summary["mean"] = summary["mean"] * 252
summary["std"] = summary["std"] * np.sqrt(252)
```

In [256...]

```
summary
```

Out[256...]

	mean	std
AAPL	0.225708	0.260254
BA	0.249376	0.245871
DIS	0.160287	0.208570
IBM	0.022081	0.197310
KO	0.070923	0.146161
MSFT	0.163698	0.231440

Create a scatterplot of the mean vs. the Standard deviation to measure the risk vs. reward.

The Y will be the mean. The X will be the risk.

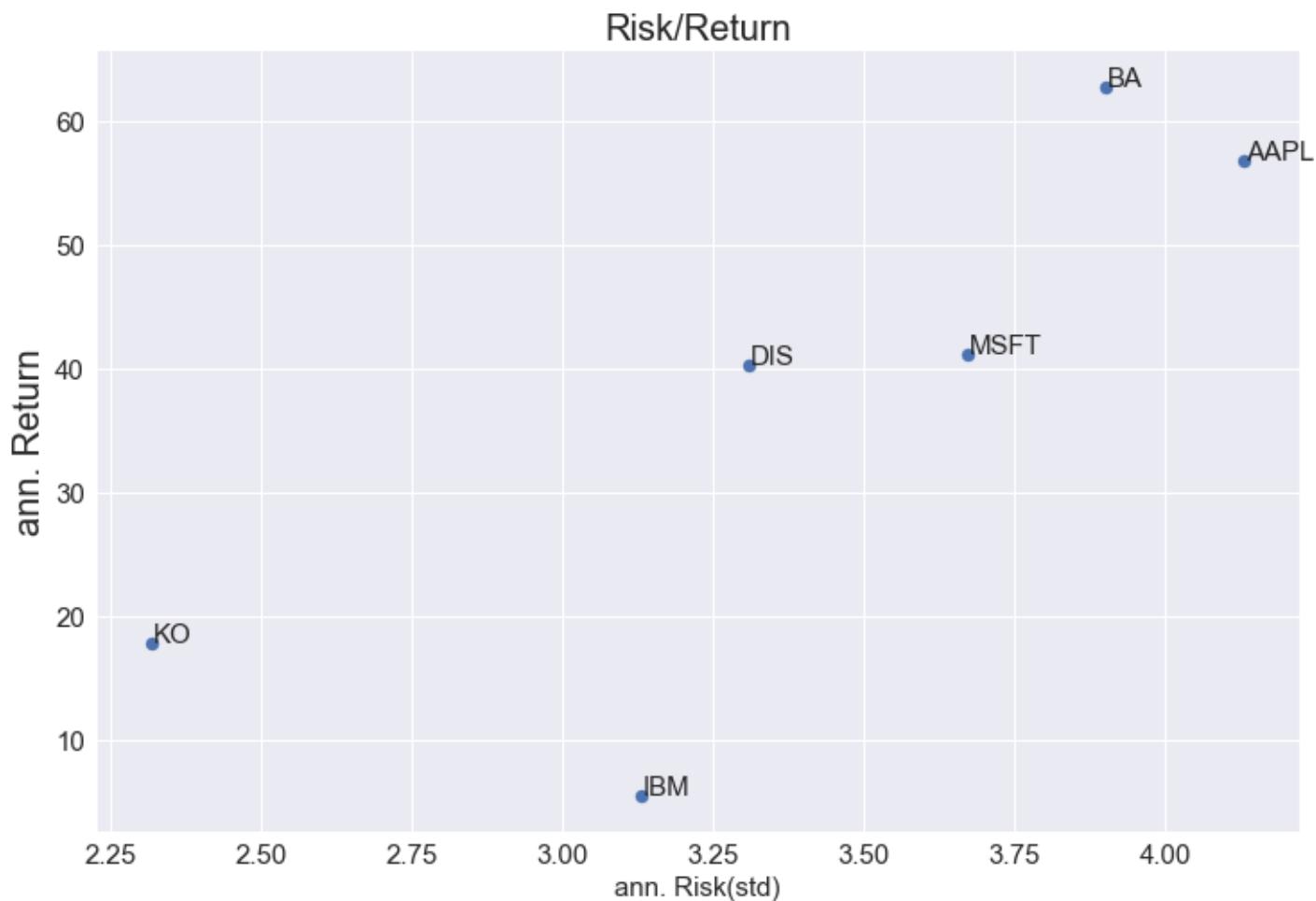
plt.annotate()

That will make the plot point on the graph be labeled.

```
for i in summary.index: plt.annotate(i, xy = (summary.loc[i, "std"]+0.002, summary.loc[i, "mean"]+0.002), size = 15)
```

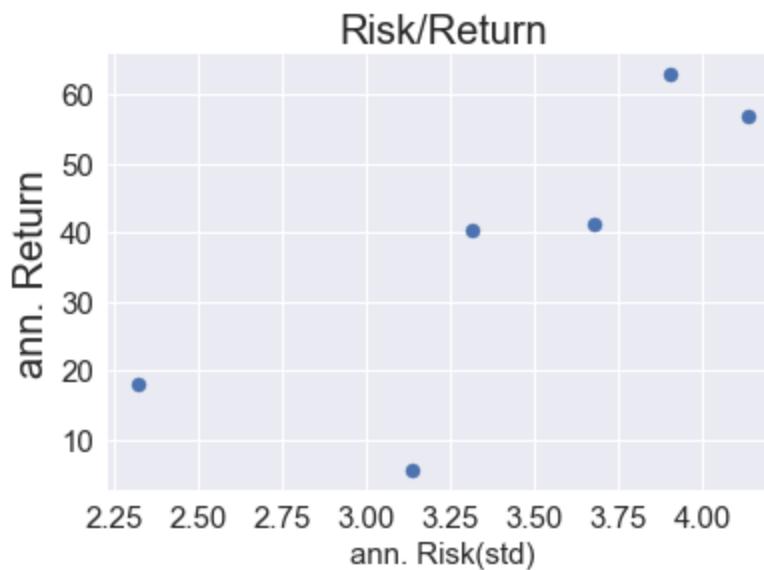
In [267...]

```
summary.plot.scatter(x = "std", y = "mean", figsize = (12, 8), s = 50, fontsize = 15)
plt.title("Risk/Return", fontsize = 20)
plt.xlabel("ann. Risk(std)", fontsize = 15)
plt.ylabel("ann. Return", fontsize = 20)
for i in summary.index:
    plt.annotate(i, xy = (summary.loc[i, "std"]+0.002, summary.loc[i, "mean"]+0.002), size = 15)
plt.show()
```



In [268...]

```
# Not annotated.
summary.plot.scatter(x = "std", y = "mean", figsize = (6, 4), s = 50, fontsize = 15)
plt.title("Risk/Return", fontsize = 20)
plt.xlabel("ann. Risk(std)", fontsize = 15)
plt.ylabel("ann. Return", fontsize = 20)
#for i in summary.index:
#    plt.annotate(i, xy = (summary.loc[i, "std"]+0.002, summary.loc[i, "mean"]+0.002), size = 15)
plt.show()
```



Why use mean over standard deviation?

It is a common practice.

Read more here: <https://www.investopedia.com/ask/answers/021915/how-standard-deviation-used-determine-risk.asp#:~:text=Standard%20deviation%20helps%20determine%20market,investments%20come%20with%20low%20risk.>

Mean is the reward because it tells you how much you stand to earn by buying a certain stock on average.

Standard deviation is how much the data tends to spread from the mean. The greater the spread the greater the volatility. People need the possibility of high returns to offset high volatility. If a stock has high returns but a high standard deviation this is normal.

If a stock has a high return and a low standard deviation, that is rare. That investment would be highly desired by investors.

Financial Time Series - Covariance and Correlation

In [271...]

```
ret.head()
```

Out[271...]

	AAPL	BA	DIS	IBM	KO	MSFT
Date						
2010-01-05	0.001729	0.032752	-0.002495	-0.012080	-0.012097	0.000323
2010-01-06	-0.015906	0.030334	-0.005314	-0.006496	-0.000355	-0.006137
2010-01-07	-0.001849	0.040482	0.000314	-0.003462	-0.002485	-0.010400
2010-01-08	0.006648	-0.009646	0.001571	0.010035	-0.018509	0.006897
2010-01-11	-0.008821	-0.011851	-0.016311	-0.010470	0.020308	-0.012720

In [272...]

```
ret.cov()
```

Out[272...]

AAPL	BA	DIS	IBM	KO	MSFT	
AAPL	0.000269	0.000104	0.000080	0.000076	0.000040	0.000107
BA	0.000104	0.000240	0.000105	0.000088	0.000057	0.000101

	AAPL	BA	DIS	IBM	KO	MSFT
DIS	0.000080	0.000105	0.000173	0.000073	0.000052	0.000086
IBM	0.000076	0.000088	0.000073	0.000154	0.000047	0.000088
KO	0.000040	0.000057	0.000052	0.000047	0.000085	0.000051
MSFT	0.000107	0.000101	0.000086	0.000088	0.000051	0.000213

In [273...]

```
ret.corr()
```

Out[273...]

	AAPL	BA	DIS	IBM	KO	MSFT
AAPL	1.000000	0.409917	0.373137	0.374747	0.266943	0.447157
BA	0.409917	1.000000	0.518342	0.459294	0.401417	0.446867
DIS	0.373137	0.518342	1.000000	0.448732	0.432673	0.450880
IBM	0.374747	0.459294	0.448732	1.000000	0.407401	0.485883
KO	0.266943	0.401417	0.432673	0.407401	1.000000	0.378181
MSFT	0.447157	0.446867	0.450880	0.485883	0.378181	1.000000

In [275...]

```
import seaborn as sns
```

.cov()

Is used to find covariance.

Positive covariance mean they move in the same direction. Negative means opposite direction. Zero means no covariance. The scale is from -1 to 1. Having some values near zero helps make the portfolio safer.

Covariance is affected by location and scale.

<https://www.investopedia.com/ask/answers/021915/how-standard-deviation-used-determine-risk.asp#:~:text=Standard%20deviation%20helps%20determine%20market,investments%20come%20with%20low%20risk.>

Heatmaps can be used to show the covariance.

Remember heatmaps is used by Seaborn.

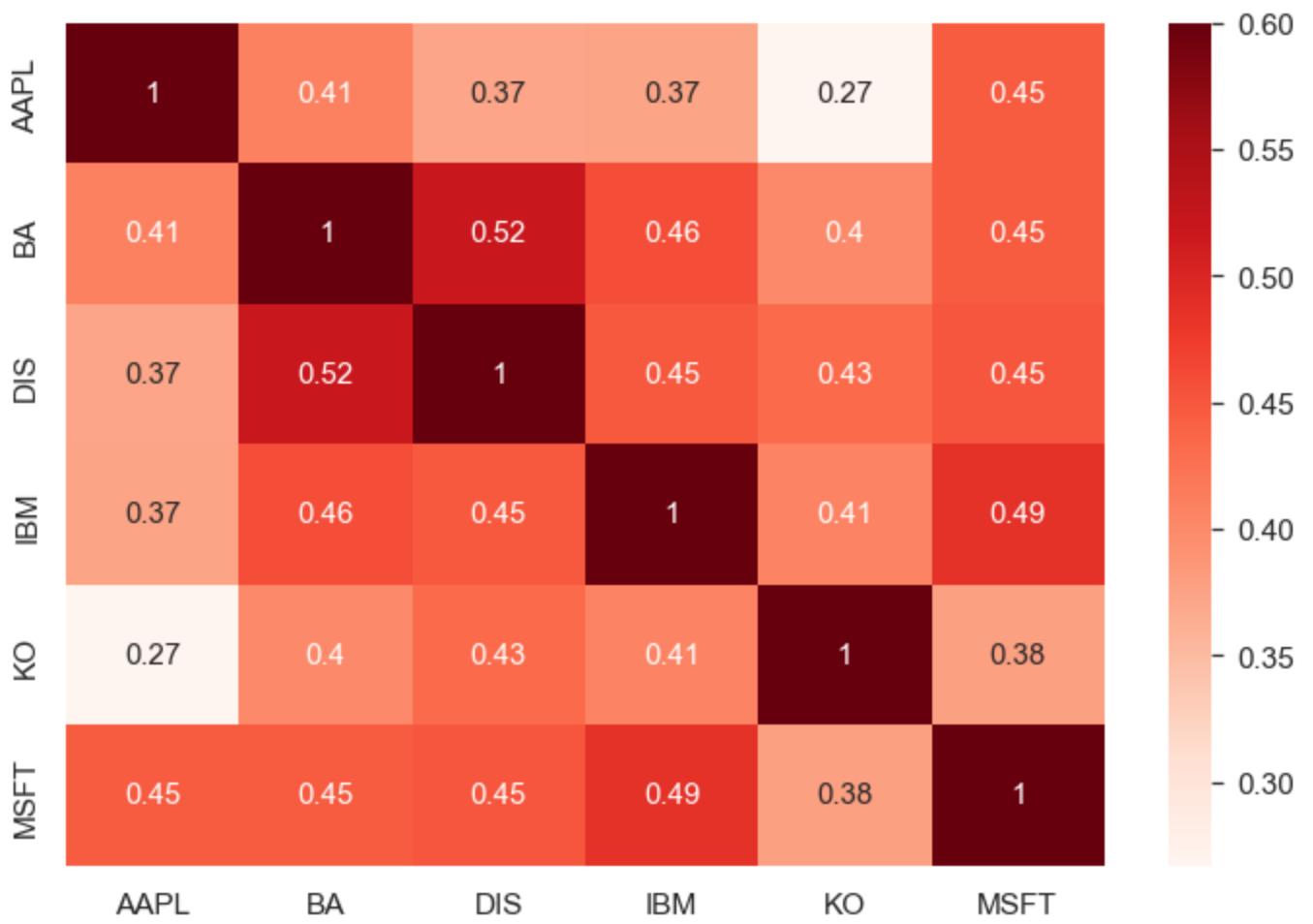
Covariance is used for portfolio risk. The whole portfolio should not move together as that would put its soudnes in jeopardy if the industry tanks, or if they all go in the same direction for whatever reason.

In [277...]

```
plt.figure(figsize = (12, 8))
sns.set(font_scale = 1.4)
sns.heatmap(ret.corr(), cmap = "Reds", annot = True, annot_kws = {"size":15}, vmax = 0.6)
```

Out[277...]

<AxesSubplot:>



In []:

Financial Data - Advanced Analysis Techniques

Importing Financial Data From Excel

In [3]:

```
import pandas as pd
```

Use pd.read_excel() to import .xls documents.

Notice how the excel date format worked for the pandas style format automatically.

This is why it is good to check a documents info after it is imported.

In [5]:

```
pd.read_excel(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\SP500.xls")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12107 entries, 0 to 12106
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
---  --  
 0   Date        12107 non-null   datetime64[ns]
 1   Open         12107 non-null   float64 
 2   High         12107 non-null   float64 
 3   Low          12107 non-null   float64 
 4   Close        12107 non-null   float64 
 5   Adj Close    12107 non-null   float64 
 6   Volume       12107 non-null   int64  
dtypes: datetime64[ns] (1), float64(5), int64(1)
memory usage: 662.2 KB
```

To recognize dates use:

parse_dates = ["Date"] pass in the index using index_col = "Date"

Of course the date entry can vary depending on the nature of the document.

In [8]:

```
pd.read_excel(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\SP500.xls", r
```

Out[8]:

	Open	High	Low	Close	Adj Close	Volume
Date						
1970-12-31	92.269997	92.790001	91.360001	92.150002	92.150002	13390000
1971-01-04	92.150002	92.190002	90.639999	91.150002	91.150002	10010000
1971-01-05	91.150002	92.279999	90.690002	91.800003	91.800003	12600000
1971-01-06	91.800003	93.000000	91.500000	92.349998	92.349998	16960000
1971-01-07	92.349998	93.260002	91.750000	92.379997	92.379997	16460000
1971-01-08	92.379997	93.019997	91.599998	92.190002	92.190002	14100000
1971-01-11	92.190002	92.669998	90.989998	91.980003	91.980003	14720000
1971-01-12	91.980003	93.279999	91.629997	92.720001	92.720001	17820000

	Open	High	Low	Close	Adj Close	Volume
Date						
1971-01-13	92.720001	93.660004	91.879997	92.559998	92.559998	19070000
1971-01-14	92.559998	93.360001	91.669998	92.800003	92.800003	17600000

Use: usecols = "first_letter: last_letter" for select columns.

usecols = "A: Z" This will import only certain columns from the excel document. The letters of course corresponds to the column letters that appear on the top of an excel documents.

Of course use_cols is short for use columns.

See below how only certain columns were imported due to the usecols feature.

In [10]:

```
pd.read_excel(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\SP500.xls", usecols="A:Z")
```

Out[10]:

	Open	High	Low	Close	Adj Close	Volume
Date						
1970-12-31	92.269997	92.790001	91.360001	92.150002	92.150002	13390000
1971-01-04	92.150002	92.190002	90.639999	91.150002	91.150002	10010000
1971-01-05	91.150002	92.279999	90.690002	91.800003	91.800003	12600000
1971-01-06	91.800003	93.000000	91.500000	92.349998	92.349998	16960000
1971-01-07	92.349998	93.260002	91.750000	92.379997	92.379997	16460000
1971-01-08	92.379997	93.019997	91.599998	92.190002	92.190002	14100000
1971-01-11	92.190002	92.669998	90.989998	91.980003	91.980003	14720000
1971-01-12	91.980003	93.279999	91.629997	92.720001	92.720001	17820000
1971-01-13	92.720001	93.660004	91.879997	92.559998	92.559998	19070000
1971-01-14	92.559998	93.360001	91.669998	92.800003	92.800003	17600000

use_cols = "A,B,C" is also an accepted format.

Just make sure that the values are located inside of a string.

Can also use: use_cols = "A, C:E"

Note that a list will NOT work.

In [34]:

```
pd.read_excel(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\SP500.xls", usecols="A,B,C")
```

Out[34]:

	Open	High
Date		
1970-12-31	92.269997	92.790001
1971-01-04	92.150002	92.190002

Use: sheet_name = "title_of_sheet" to import a certain page of an excel document.

```
sheet_name = "SP5000"
```

This will pull up the page that analysis must be done for.

In [36]:

```
pd.read_excel(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\SP500.xls", s
```

Out[36]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1970-12-31	92.269997	92.790001	91.360001	92.150002	92.150002	13390000
1	1971-01-04	92.150002	92.190002	90.639999	91.150002	91.150002	10010000
2	1971-01-05	91.150002	92.279999	90.690002	91.800003	91.800003	12600000
3	1971-01-06	91.800003	93.000000	91.500000	92.349998	92.349998	16960000
4	1971-01-07	92.349998	93.260002	91.750000	92.379997	92.379997	16460000

sheet_name can accept intergers as well as the name of the sheet.

The name of the sheet has to be a string. The interger would be passed in as an interger.

In [38]:

```
# Selecting a sheet using the name of the sheet.  
pd.read_excel(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\SP500.xls", s
```

Out[38]:

	Unnamed: 0	City	Sales
0	Mike	New York	25
1	Jim	Boston	43
2	Steven	London	76
3	Joe	Madrid	12
4	Tom	Paris	89

In [37]:

```
# Integer method of selecting the second page, that is 1 because python counts from zero.  
pd.read_excel(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\SP500.xls", s
```

Out[37]:

	Unnamed: 0	City	Sales
0	Mike	New York	25
1	Jim	Boston	43
2	Steven	London	76
3	Joe	Madrid	12
4	Tom	Paris	89

Defining the document that is actually being used.

In [16]:

```
# Note sp500 will be lowercased, instead of uppercased to facilitate faster typing.  
sp500 = pd.read_excel(r"C:\Users\alonz\Documents\Video_Lecture_NBs\Video_Lecture_NBs\SP500
```

In [17]:

```
sp500.head()
```

Out[17]:

	Open	High	Low	Close
Date				
1970-12-31	92.269997	92.790001	91.360001	92.150002
1971-01-04	92.150002	92.190002	90.639999	91.150002
1971-01-05	91.150002	92.279999	90.690002	91.800003
1971-01-06	91.800003	93.000000	91.500000	92.349998
1971-01-07	92.349998	93.260002	91.750000	92.379997

In [18]:

```
sp500.tail()
```

Out[18]:

	Open	High	Low	Close
Date				
2018-12-21	2465.379883	2504.409912	2408.550049	2416.620117
2018-12-24	2400.560059	2410.340088	2351.100098	2351.100098
2018-12-26	2363.120117	2467.760010	2346.580078	2467.699951
2018-12-27	2442.500000	2489.100098	2397.939941	2488.830078
2018-12-28	2498.770020	2520.270020	2472.889893	2485.739990

In [19]:

```
sp500.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 12107 entries, 1970-12-31 to 2018-12-28
Data columns (total 4 columns):
 #   Column   Non-Null Count   Dtype  
---  --  
 0   Open      12107 non-null   float64 
 1   High      12107 non-null   float64 
 2   Low       12107 non-null   float64 
 3   Close     12107 non-null   float64 
dtypes: float64(4)
memory usage: 472.9 KB
```

Saving from Pandas to Excel.

Saving to a CSV: use `.to_csv("chosen_name.csv")`

In [40]:

```
# Example of saving a CSV file.
sp500.to_csv("SP500.csv")
```

Saving to an Xls file is depreciating.

If the method below does not work openpyxl has to be installed. Below is the documentation:

<https://openpyxl.readthedocs.io/en/stable/>

Alternatively one can use excel to convert .csv to .xls see below:

<https://www.howtogeek.com/770734/how-to-convert-a-csv-file-to-microsoft-excel/>

In [41]:

```
# Example of saving to an excel file.
sp500.to_excel("sp500_red.xls")
```

```
C:\Users\alonz\AppData\Local\Temp\ipykernel_20752\2754551972.py:2: FutureWarning: As the xlwt package is no longer maintained, the xlwt engine will be removed in a future version of pandas. This is the only engine in pandas that supports writing in the xls format. Install openpyxl and write to an xlsx file instead. You can set the option io.excel.xls.writer to 'xlwt' to silence this warning. While this option is deprecated and will also raise a warning, it can be globally set and the warning suppressed.
sp500.to_excel("sp500_red.xls")
```

New Section

Simple Moving Averages (SMA) with rolling()

Use the rolling() method to calculate rolling averages.

What is a rolling average?

Rolling average falls under the umbrella of rolling statistics. Rolling statistics takes note of the data that precedes the current point by a set amount. For example a rolling average can have a window of 50 days for a stock. That means that the rolling average would be comprised of a stock's current price and the preceding 49 days.

Again sp500 will be a lower case variable for ease of use.

Importing All of the Necessary Components

In [50]:

```
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use("seaborn")
```

In [55]:

```
# Importing the CSV file that was saved from the last section.
# Note that from importing from a CSV the date format will not be recognized like could be
sp500 = pd.read_csv("SP500.csv", parse_dates = ["Date"], index_col = "Date")
```

In [47]:

```
sp500.head()
```

Out[47]:

	Open	High	Low	Close
Date				
1970-12-31	92.269997	92.790001	91.360001	92.150002
1971-01-04	92.150002	92.190002	90.639999	91.150002
1971-01-05	91.150002	92.279999	90.690002	91.800003
1971-01-06	91.800003	93.000000	91.500000	92.349998
1971-01-07	92.349998	93.260002	91.750000	92.379997

```
In [48]: sp500.tail()
```

```
Out[48]:
```

	Open	High	Low	Close
Date				
2018-12-21	2465.379883	2504.409912	2408.550049	2416.620117
2018-12-24	2400.560059	2410.340088	2351.100098	2351.100098
2018-12-26	2363.120117	2467.760010	2346.580078	2467.699951
2018-12-27	2442.500000	2489.100098	2397.939941	2488.830078
2018-12-28	2498.770020	2520.270020	2472.889893	2485.739990

```
In [49]: sp500.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 12107 entries, 1970-12-31 to 2018-12-28
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  --  
 0   Open    12107 non-null   float64
 1   High    12107 non-null   float64
 2   Low     12107 non-null   float64
 3   Close   12107 non-null   float64
dtypes: float64(4)
memory usage: 472.9 KB
```

Conventional pandas series is default after using pandas to import.

```
In [58]: sp500.Close.head(2)
```

```
Out[58]:
```

Date	Close
1970-12-31	92.150002
1971-01-04	91.150002

Name: Close, dtype: float64

```
In [59]: type(sp500.Close)
```

```
Out[59]: pandas.core.series.Series
```

selected_column.to_frame() for column to DataFrame

Converts panda series to a Dataframe that is isolated. It will be just the index and the column selected.

```
In [74]: sp500 = sp500.Close.to_frame()
```

```
In [63]: sp500.head()
```

```
Out[63]:
```

	Close
Date	
1970-12-31	92.150002

Close

Date

1971-01-04	91.150002
1971-01-05	91.800003
1971-01-06	92.349998
1971-01-07	92.379997

Plotting the closings prices of S&P 500.

Note that the sp500 variable name has been replaced by the closing price that has been assigned to that name.

The figure below shows all of the closings.

Note this is the conventional close and not the adjusted close. This means that dividends are NOT included.

In [64]:

```
sp500.plot(figsize = (12, 8), fontsize = 15)
plt.legend(loc = "upper left", fontsize = 15)
plt.show()
```



Creating rolling indexes

Can be done in 3 steps.

Step 1 - The scope can be limited by a range of indexes.

Below the data set spans 10 years.

```
In [112]: # Use .copy() to not taint the original data source.  
sp500 = SP500.loc["2008-12-31": "2018-12-31"].copy()
```

```
In [113]: sp500 = sp500.Close.to_frame()
```

Step 2 - Use: data_source.rolling(window = integer)

```
In [114]: sp500.rolling(window = 10)
```

```
Out[114]: Rolling [window=10,center=False,axis=0,method=single]
```

```
In [115]: sp500.head(15)
```

```
Out[115]:
```

	Close
	Date
2008-12-31	903.250000
2009-01-02	931.799988
2009-01-05	927.450012
2009-01-06	934.700012
2009-01-07	906.650024
2009-01-08	909.729980
2009-01-09	890.349976
2009-01-12	870.260010
2009-01-13	871.789978
2009-01-14	842.619995
2009-01-15	843.739990
2009-01-16	850.119995
2009-01-20	805.219971
2009-01-21	840.239990
2009-01-22	827.500000

Using `rolling(window = 10)` makes a new Rolling data type.

Windows determines how many timestamps are taken into consideration.

```
In [86]: type(sp500.rolling(window = 10))
```

```
Out[86]: pandas.core.window.rolling.Rolling
```

```
In [87]: sp500.head(15)
```

```
Out[87]:
```

	Close
	Date
2008-12-31	903.250000
2009-01-02	931.799988
2009-01-05	927.450012
2009-01-06	934.700012
2009-01-07	906.650024
2009-01-08	909.729980
2009-01-09	890.349976
2009-01-12	870.260010
2009-01-13	871.789978
2009-01-14	842.619995
2009-01-15	843.739990
2009-01-16	850.119995
2009-01-20	805.219971
2009-01-21	840.239990
2009-01-22	827.500000

Date	Close
Date	
2008-12-31	903.250000
2009-01-02	931.799988
2009-01-05	927.450012
2009-01-06	934.700012
2009-01-07	906.650024
2009-01-08	909.729980
2009-01-09	890.349976
2009-01-12	870.260010
2009-01-13	871.789978
2009-01-14	842.619995
2009-01-15	843.739990
2009-01-16	850.119995
2009-01-20	805.219971
2009-01-21	840.239990
2009-01-22	827.500000

Step 3 chain methods after .rolling(windows = int) to make rolling statistics.

rolling() only produces a data frame with a statistic at the end of it.

In [95]:

```
sp500.rolling(window = 10)
```

Out[95]:

```
# Only the last ten values are taken into account.
sp500.rolling(window = 10).mean()
```

Out[96]:

Date	Close
2008-12-31	NaN
2009-01-02	NaN
2009-01-05	NaN
2009-01-06	NaN
2009-01-07	NaN
...	...
2018-12-21	2565.915991
2018-12-24	2537.254004
2018-12-26	2520.345996

Close

Date

2018-12-27 2504.121997

2018-12-28 2487.641992

2516 rows × 1 columns

In [94]:

```
sp500.rolling(window = 10).median()
```

Close

Date

2008-12-31 NaN

2009-01-02 NaN

2009-01-05 NaN

2009-01-06 NaN

2009-01-07 NaN

...

...

2018-12-21 2573.054931

2018-12-24 2546.049926

2018-12-26 2526.449951

2018-12-27 2497.895019

2018-12-28 2487.285034

2516 rows × 1 columns

In [98]:

```
# max() can be used to calculate the maximum.  
sp500.rolling(window = 10).max()
```

Close

Date

2008-12-31 NaN

2009-01-02 NaN

2009-01-05 NaN

2009-01-06 NaN

2009-01-07 NaN

...

...

2018-12-21 2651.070068

2018-12-24 2651.070068

2018-12-26 2651.070068

2018-12-27 2650.540039

Close

Date

2018-12-28 2599.949951

2516 rows × 1 columns

In [100...

```
# min() can be used to calculate the minimum.  
sp500.rolling(window = 10).min()
```

Out[100...

Close

Date

2008-12-31 NaN

2009-01-02 NaN

2009-01-05 NaN

2009-01-06 NaN

2009-01-07 NaN

... ...

2018-12-21 2416.620117

2018-12-24 2351.100098

2018-12-26 2351.100098

2018-12-27 2351.100098

2018-12-28 2351.100098

2516 rows × 1 columns

.rolling(window = int, min_periods = int)

min_periods = int

The min period shows the number of indexes that are required to have a value. The rest could be shown to be NaN.

Window defines how many of the index rows are included in the statistical calculation.

MINIMUM PERIODS SHOWS HOW MANY DATA POINTS ARE NEEDED TO MAKE AN AVERAGE. By default the windows and the minimum period are the same.

In [118...

```
sp500.rolling(window = 10, min_periods = 5).mean()
```

Out[118...

Close

Date

2008-12-31 NaN

2009-01-02 NaN

Close

Date

2009-01-05	NaN
2009-01-06	NaN
2009-01-07	920.770007
...	...
2018-12-21	2565.915991
2018-12-24	2537.254004
2018-12-26	2520.345996
2018-12-27	2504.121997
2018-12-28	2487.641992

2516 rows × 1 columns

In [108...

```
sp500.rolling(window = 10, min_periods = 5).mean()
```

Out[108...

Close

Date	
2008-12-31	NaN
2009-01-02	NaN
2009-01-05	NaN
2009-01-06	NaN
2009-01-07	920.770007
...	...
2018-12-21	2565.915991
2018-12-24	2537.254004
2018-12-26	2520.345996
2018-12-27	2504.121997
2018-12-28	2487.641992

2516 rows × 1 columns

Momentum Trading Strategies with SMAs

This is used for technical analysis and not fundamental analysis.

In [121...

```
sp500.head()
```

Out[121...

Close

Date

Close

Date

Date	Close
2008-12-31	903.250000
2009-01-02	931.799988
2009-01-05	927.450012
2009-01-06	934.700012
2009-01-07	906.650024

In [122...]

```
sp500.tail()
```

Out[122...]

Close

Date	Close
2018-12-21	2416.620117
2018-12-24	2351.100098
2018-12-26	2467.699951
2018-12-27	2488.830078
2018-12-28	2485.739990

Adding new columns to the data that calculates rolling averages.

In [124...]

```
# Rolling stats subsets of data will have capitals.  
sp500["SMA50"] = sp500.rolling(window = 50, min_periods = 50).mean()
```

In [125...]

```
sp500
```

Out[125...]

Close SMA50

Date	Close	SMA50
2008-12-31	903.250000	NaN
2009-01-02	931.799988	NaN
2009-01-05	927.450012	NaN
2009-01-06	934.700012	NaN
2009-01-07	906.650024	NaN
...
2018-12-21	2416.620117	2692.420195
2018-12-24	2351.100098	2684.874795
2018-12-26	2467.699951	2678.886196
2018-12-27	2488.830078	2673.646997
2018-12-28	2485.739990	2667.163398

2516 rows × 2 columns

Below is a chart of prices charted with the moving average.

In [128...]

```
sp500.plot(figsize = (12, 8), fontsize = 15)
plt.legend(loc = "upper left", fontsize = 15)
plt.title("S&P 500 Closing", fontsize = 13)
plt.show()
```



In [130...]

```
# The 200 Day Moving Average
sp500["SMA200"] = sp500.Close.rolling(window = 200).mean()
```

In [131...]

```
# Shows no value because the minimum window is not reached.
sp500.head()
```

Out[131...]

	Close	SMA50	SMA200
Date			
2008-12-31	903.250000	NaN	NaN
2009-01-02	931.799988	NaN	NaN
2009-01-05	927.450012	NaN	NaN
2009-01-06	934.700012	NaN	NaN
2009-01-07	906.650024	NaN	NaN

Date

2008-12-31	903.250000	NaN	NaN
2009-01-02	931.799988	NaN	NaN
2009-01-05	927.450012	NaN	NaN
2009-01-06	934.700012	NaN	NaN
2009-01-07	906.650024	NaN	NaN

In [132...]

```
# Tails will show values for all of the columns.
```

```
sp500.tail()
```

Out[132...]

	Close	SMA50	SMA200
--	--------------	--------------	---------------

Date

2018-12-21	2416.620117	2692.420195	2753.65980
2018-12-24	2351.100098	2684.874795	2751.48245
2018-12-26	2467.699951	2678.886196	2749.90585
2018-12-27	2488.830078	2673.646997	2748.52345
2018-12-28	2485.739990	2667.163398	2747.20475

In [133...]

```
sp500.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2516 entries, 2008-12-31 to 2018-12-28
Data columns (total 3 columns):
 #   Column   Non-Null Count   Dtype  
---  --  
 0   Close    2516 non-null    float64 
 1   SMA50   2467 non-null    float64 
 2   SMA200  2317 non-null    float64 
dtypes: float64(3)
memory usage: 78.6 KB
```

Closing price and the moving averages, both 50 day and 200.

In [135...]

```
sp500.plot(figsize = (15, 10), fontsize = 15)
plt.title("S&P Closing Price")
plt.ylabel("Price in Dollars")
plt.xlabel("Year")
plt.legend(fontsize = 15)
plt.show()
```



Moving Averages Graph

Momentum vs Contrarian Investing

A common strategy is to long a stock when the shorter SMA is above the longterm SMA. When the long term SMA is above the short term SMA sell or short.

The most recent trend is more indicative of the near future. So invest when the lines cross and the short term line is above the long term line. That is a momentum based strategy.

A contrarian strategy would try to buy a stock when the long term SMA is below the short term SMA, and vice versa.

They believe that a reversion to the mean would impede the prevailing trend.

In [140]:

```
# Notice the range for the iloc column has -2: don't forget the : to get the rest of the data
sp500.iloc[:, -2:].plot(figsize = (15, 10), fontsize = 15)
plt.title("S&P Moving Averages")
plt.ylabel("Price in Dollars")
plt.xlabel("Year")
plt.legend(fontsize = 15)
plt.show()
```



Performance Reporting with rolling()

Risk and Return are the Most Important.

Risk/Reward are both calculated by Monthly data for a 3 year period.

Also known as a 36 month period.

Return is the annualized average monthly return over 36 months.

Risk is the annualized standard deviation of monthly return.

In [141...]

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use("seaborn")
```

In [142...]

```
sp500 = pd.read_csv("SP500.csv", parse_dates = ["Date"], index_col = "Date", usecols = [
```

In [143...]

```
sp500.head()
```

Out[143...]

Close

Date

Close

Date

Date	Close
1970-12-31	92.150002
1971-01-04	91.150002
1971-01-05	91.800003
1971-01-06	92.349998
1971-01-07	92.379997

Utilizing resample()

Resample the data to get the monthly data.

Resample then rolling. That means get the data adjusted for time then calculate the rolling statistics.

```
In [144...]: sp500.resample("M", kind = "period").last()
```

```
Out[144...]:
```

Close

Date	Close
1970-12	92.150002
1971-01	95.879997
1971-02	96.750000
1971-03	100.309998
1971-04	103.949997
...	...
2018-08	2901.520020
2018-09	2913.979980
2018-10	2711.739990
2018-11	2760.169922
2018-12	2485.739990

577 rows × 1 columns

Pass kind = "period" to define the months properly.

Monthly returns are calculated here percentagewise. last() gets the end of the month resample not the start. .dropna() gets rid of empty values.

This gets the percent change of the data for each month.

```
In [145...]: month_ret = sp500.resample("M", kind = "period").last().pct_change().dropna()
```

```
In [146...]:
```

month_ret

```
Out[146...
```

Close

Date	Close
1971-01	0.040477
1971-02	0.009074
1971-03	0.036796
1971-04	0.036287
1971-05	-0.041558
...	...
2018-08	0.030263
2018-09	0.004294
2018-10	-0.069403
2018-11	0.017859
2018-12	-0.099425

576 rows × 1 columns

Get's the annualized data of the returns for 36 months utilizing what was created before. Annualized by multiplying the monthly data by 12.

```
In [147...
```

```
month_ret.rolling(36).mean()*12
```

```
Out[147...
```

Close

Date	Close
1971-01	NaN
1971-02	NaN
1971-03	NaN
1971-04	NaN
1971-05	NaN
...	...
2018-08	0.133653
2018-09	0.143899
2018-10	0.093103
2018-11	0.098888
2018-12	0.071590

576 rows × 1 columns

Creates a new column on the chart.

```
In [148...
```

```
month_ret["Return"] = month_ret.rolling(36).mean()*12
```

```
In [149...]: month_ret.Close.rolling(36).std()*np.sqrt(12)
```

```
Out[149...]: Date
1971-01      NaN
1971-02      NaN
1971-03      NaN
1971-04      NaN
1971-05      NaN
...
2018-08    0.094305
2018-09    0.091740
2018-10    0.093480
2018-11    0.093556
2018-12    0.111514
Freq: M, Name: Close, Length: 576, dtype: float64
```

Risk is the standard deviation of monthly returns.

```
In [153...]: # Creates the std of the past 36 months in an annualized fashion.
month_ret.Close.rolling(36).std()*np.sqrt(12)
```

```
Out[153...]: Date
1971-01      NaN
1971-02      NaN
1971-03      NaN
1971-04      NaN
1971-05      NaN
...
2018-08    0.094305
2018-09    0.091740
2018-10    0.093480
2018-11    0.093556
2018-12    0.111514
Freq: M, Name: Close, Length: 576, dtype: float64
```

```
In [158...]: month_ret.loc[:"2017-01"].Close.rolling(36).std()*np.sqrt(12)
```

```
Out[158...]: Date
1971-01      NaN
1971-02      NaN
1971-03      NaN
1971-04      NaN
1971-05      NaN
...
2016-09    0.107810
2016-10    0.106598
2016-11    0.107094
2016-12    0.106836
2017-01    0.104162
Freq: M, Name: Close, Length: 553, dtype: float64
```

```
In [160...]: month_ret["Risk"] = month_ret.Close.rolling(36).std()*np.sqrt(12)
```

```
In [164...]: # Drop invalid responses here and in the original chart.
month_ret.dropna(inplace = True)
```

The chart below shows the annualized monthly gain for the past 3 years.

In [165...]

```
month_ret.head()
```

Out[165...]

	Close	Return	Risk
Date			
1973-12	0.016569	0.026493	0.123219
1974-01	-0.010046	0.009652	0.121276
1974-02	-0.003624	0.005419	0.121201
1974-03	-0.023280	-0.014606	0.119981
1974-04	-0.039051	-0.039719	0.119791

In [166...]

```
month_ret.tail()
```

Out[166...]

	Close	Return	Risk
Date			
2018-08	0.030263	0.133653	0.094305
2018-09	0.004294	0.143899	0.091740
2018-10	-0.069403	0.093103	0.093480
2018-11	0.017859	0.098888	0.093556
2018-12	-0.099425	0.071590	0.111514

S&P 500 can be quite volatile and risky. It depends on the market environment.

Whenever the annualized standard deviation over the last 3 years is greater than the return the investment is considered risky. Perhaps the returns would look better if the adjusted close was looked at which includes dividends.

Risk and return can be negatively correlated as shown below. High risk years have low returns. Low risk years have high returns.

In [171...]

```
month_ret.iloc[:, -2: ].plot(figsize = (15, 10), fontsize = 15)
plt.title("Risk and Reward over Time for the S&P 500", fontsize = 16)
plt.legend(fontsize = 15)
plt.show()
```



Calculate the correlation between risk and return. Use: `.corr()`

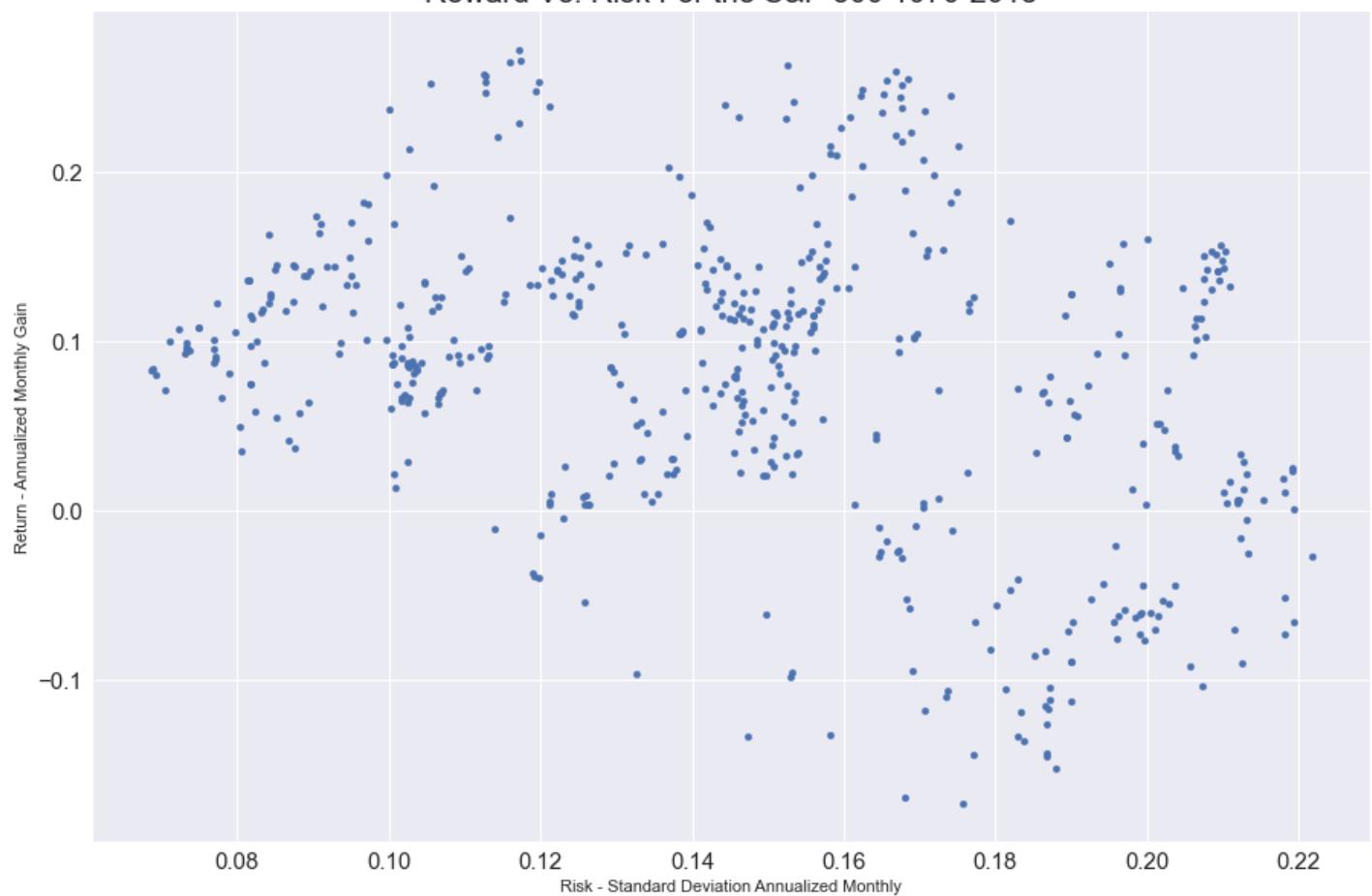
Below it can be seen that the negative correlation is quite substantial.

```
In [172...]: month_ret.iloc[:, -2:].corr()
```

```
Out[172...]:
```

	Return	Risk
Return	1.000000	-0.327747
Risk	-0.327747	1.000000

```
In [177...]: month_ret.iloc[:, -2:].plot(kind = "scatter", x = "Risk", y = "Return", figsize = (15, 10))
plt.title("Reward Vs. Risk For the S&P 500 1970-2018", fontsize = 20)
plt.ylabel("Return - Annualized Monthly Gain")
plt.xlabel("Risk - Standard Deviation Annualized Monthly")
plt.show()
```



Performance and Investment Periods / Time Diversification

Will calculate average monthly returns from a period of 12 months to 240 months.

In [208...]

```
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use("seaborn")
```

In [209...]

```
sp500 = pd.read_csv("SP500.csv", parse_dates = ["Date"], index_col = "Date", usecols = [
```

In [210...]

```
sp500.head()
```

Out[210...]

Close

Date	
1970-12-31	92.150002
1971-01-04	91.150002
1971-01-05	91.800003
1971-01-06	92.349998
1971-01-07	92.379997

In [211...]

```
sp500.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 12107 entries, 1970-12-31 to 2018-12-28
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  --  
 0   Close    12107 non-null   float64 
dtypes: float64(1)
memory usage: 189.2 KB
```

In [212...]

```
sp500.plot(figsize = (15,10), fontsize = 15)
plt.legend(fontsize = 15)
plt.show()
```



In [226...]

```
month_ret = sp500.resample("M", kind = "period").last().pct_change().dropna()
```

In [227...]

```
month_ret.tail()
```

Out[227...]

Close

	Date
2018-08	0.030263
2018-09	0.004294
2018-10	-0.069403
2018-11	0.017859
2018-12	-0.099425

In [228...]

```
month_ret.columns = ["m_returns"]
```

```
In [229]: month_ret.tail()
```

```
Out[229]: m_returns
```

Date	m_returns
2018-08	0.030263
2018-09	0.004294
2018-10	-0.069403
2018-11	0.017859
2018-12	-0.099425

```
In [230]: month_ret.rolling(3 * 12).mean() * 12
```

```
Out[230]: m_returns
```

Date	m_returns
1971-01	NaN
1971-02	NaN
1971-03	NaN
1971-04	NaN
1971-05	NaN
...	...
2018-08	0.133653
2018-09	0.143899
2018-10	0.093103
2018-11	0.098888
2018-12	0.071590

576 rows × 1 columns

```
In [231]: month_ret.tail()
```

```
Out[231]: m_returns
```

Date	m_returns
2018-08	0.030263
2018-09	0.004294
2018-10	-0.069403
2018-11	0.017859
2018-12	-0.099425

```
In [232... type(month_ret)
```

```
Out[232... pandas.core.frame.DataFrame
```

Convert to Dataframe

How to calculate rolling statistics for each period.

The actual for loop takes all of the data into consideration for the for loop. It creates a rolling average over a longer time period.

Formatting: Use a for loop. The "{}" symbol puts the item in the for loop as the title.

```
In [235...
```

```
for years in [1, 3, 5, 10, 20]:  
    month_ret["{}Y".format(years)] = month_ret.m_returns.rolling(years * 12).mean() * 12
```

```
In [236...
```

```
month_ret.tail()
```

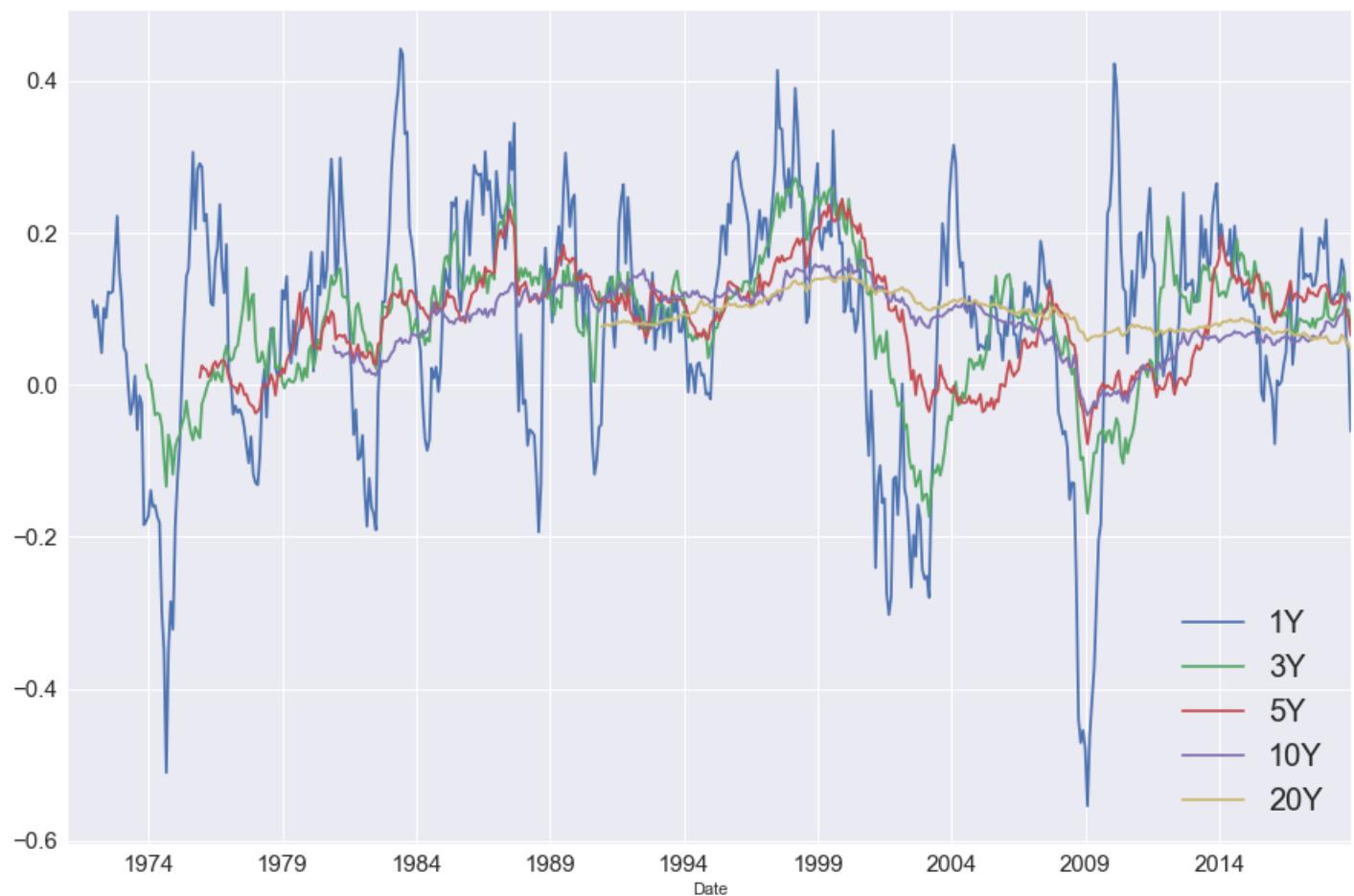
```
Out[236...
```

	m_returns	1Y	3Y	5Y	10Y	20Y
--	-----------	----	----	----	-----	-----

Date	m_returns	1Y	3Y	5Y	10Y	20Y
2018-08	0.030263	0.165207	0.133653	0.119980	0.092904	0.066120
2018-09	0.004294	0.150198	0.143899	0.114889	0.102412	0.063215
2018-10	-0.069403	0.058606	0.093103	0.092089	0.112415	0.055730
2018-11	0.017859	0.048383	0.098888	0.090051	0.121685	0.053667
2018-12	-0.099425	-0.060874	0.071590	0.065453	0.110961	0.045877

```
In [239...
```

```
month_ret.iloc[:, -5: ].plot(figsize = (15, 10), subplots = False, fontsize = 15, sharey = True)  
plt.legend(fontsize = 20)  
plt.show()
```



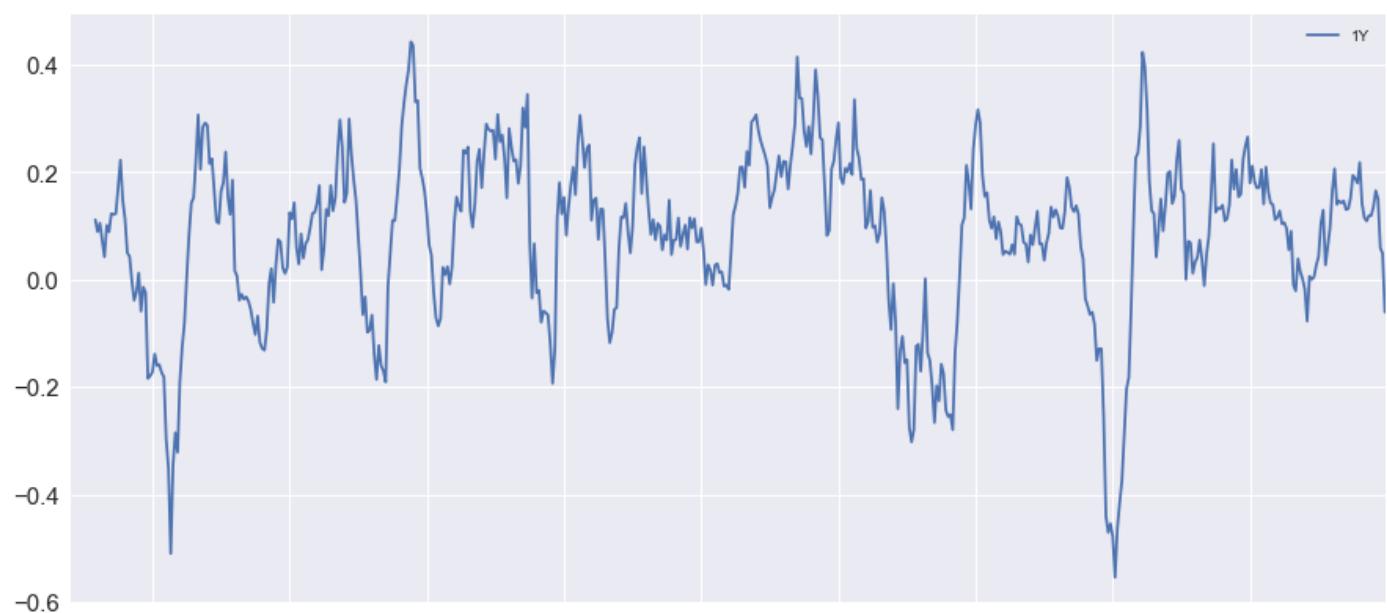
Subplots.

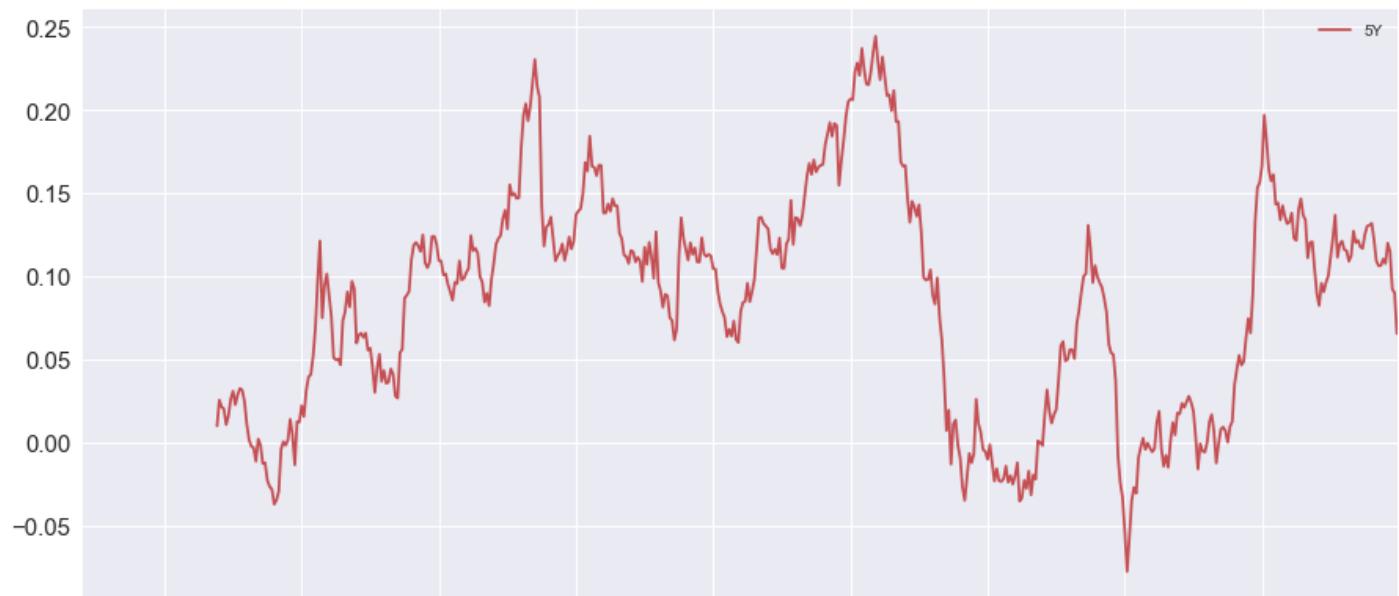
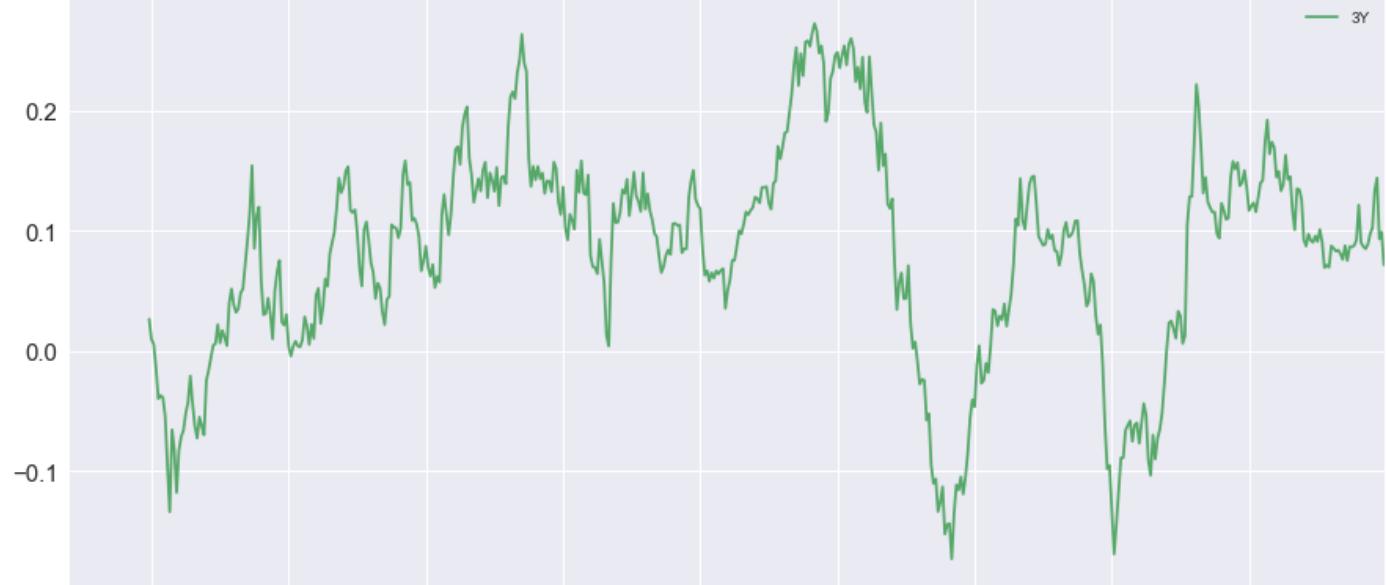
subplots = True

Over longer periods risk goes down. Loses are eliminated over 20 years.

In [241...]

```
month_ret.iloc[:, -5: ].plot(figsize = (15, 40), subplots = True, fontsize = 15, sharey = False)
plt.legend(fontsize = 20)
plt.show()
```







Next Section

Simple Return vs. Log Returns

Simple returns are the arithmetic of simple returns that is the conventional standard format that involves basic numbers.

Log returns are log based and are more reliable.

In [243...]

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

In [244...]

```
df = pd.DataFrame(index = [2016, 2017, 2018], data = [100, 50, 95], columns = ["Price"])
```

In [245...]

```
df
```

Out[245...]

	Price
2016	100
2017	50
2018	95

2016	100
2017	50
2018	95

Simple returns can be seen in the basic percentage change year over year shown here. The original value was 100. In the first year 50% of the value was lost. In the second year a 90% of the remaining value of 50 was gained bringing it up to 95.

In [247...]

```
simple_returns = df.pct_change().dropna()
simple_returns
```

Out[247...]

	Price
2017	-0.5
2018	0.9

2017	-0.5
2018	0.9

Simple percentages can be misleading.

See here how the average of the two percentage changes is 0.2. This seems good but it is not. 50% of the value was lost in the first year. A hundred percent gain is needed to break even but the gain was only 90%. $-50 + 90 = 40$ this makes the average return .20 or 20%.

The average percentage gain is positive even though value was lost. Logs do not have this problem.

```
In [248... simple_returns.mean()
```

```
Out[248... Price    0.2
          dtype: float64
```

```
In [249... 100 * 1.2 * 1.2
```

```
Out[249... 144.0
```

```
In [250... df
```

```
Out[250...      Price
2016    100
2017     50
2018    95
```

Use logs by doing this: `log(current_price/ previous_price)`

Find the log of the current price divided by the previous price.

`np.log()` takes the log

Use the shift to help get the previous values.

A negative log shows that value was lost. A positive log shows value were gained even if the value is less than 1.

```
In [251... np.log(df / df.shift(1))
```

```
Out[251...      Price
2016      NaN
2017    -0.693147
2018     0.641854
```

```
In [252... log_returns = np.log(df / df.shift(1)).dropna()
```

```
In [253... log_returns
```

Out[253...]

Price

2017 -0.693147

2018 0.641854

A positive average log means that money was gained. A negative value means value was lost.

In [254...]

log_returns.mean()

Out[254...]

Price -0.025647
dtype: float64

Find the final using average in log form.

Note that exp is e on the calculator.

Original_value (exp)^(number of periods Average log returns) = Current_value

np.exp() is used for exponents.

Total gain or loss is found by subtracting Original value - Current Value

In [258...]

100 * np.exp(2 * log_returns.mean())

Out[258...]

Price 95.0
dtype: float64

Over a long period of time with many different values the differences between log returns and simple returns are low.

This is an extreme example. Also the fluctuation between values tend not to be this extreme. Simple returns tend to be the standard in financial industries.

Next Section

S&P500 Return Triangle

Note the triangle is log based.

To find the proper square of the triangle,

You must start with the investment term on the X-axis. Then select the FINAL year of the investment to find the average return.

In [272...]

```
import pandas as pd
import numpy as np
import seaborn as sns

import matplotlib.pyplot as plt
plt.style.use("seaborn")
```

In [273...]

```
sp500 = pd.read_csv("SP500.csv", parse_dates = ["Date"], index_col = "Date", usecols = [
```

In [274...]

```
sp500.head()
```

Out[274...]

Close

	Date
1970-12-31	92.150002
1971-01-04	91.150002
1971-01-05	91.800003
1971-01-06	92.349998
1971-01-07	92.379997

In [275...]

```
sp500 = sp500.loc["1988-12-30" : "2018-12-31"].copy()
```

In [276...]

```
sp500.head()
```

Out[276...]

Close

	Date
1988-12-30	277.720001
1989-01-03	275.309998
1989-01-04	279.429993
1989-01-05	280.010010
1989-01-06	280.670013

In [277...]

```
annual = sp500.resample("A", kind = "period").last()  
annual
```

Out[277...]

Close

	Date
1988	277.720001
1989	353.399994
1990	330.220001
1991	417.089996
1992	435.709991
1993	466.450012
1994	459.269989
1995	615.929993
1996	740.739990
1997	970.429993

Close

Date

1998 1229.229980

1999 1469.250000

2000 1320.280029

2001 1148.079956

2002 879.820007

2003 1111.920044

2004 1211.920044

2005 1248.290039

2006 1418.300049

2007 1468.359985

2008 903.250000

2009 1115.099976

2010 1257.640015

2011 1257.599976

2012 1426.189941

2013 1848.359985

2014 2058.899902

2015 2043.939941

2016 2238.830078

2017 2673.610107

2018 2485.739990

Creates a log of the returns.

In [278...]

```
annual["Return"] = np.log(annual.Close / annual.Close.shift())
```

In [279...]

```
annual.dropna(inplace = True)
```

In [280...]

```
annual
```

Out[280...]

Close **Return**

Date

1989 353.399994 0.240987

1990 330.220001 -0.067841

1991 417.089996 0.233543

1992 435.709991 0.043675

Close Return

Date

1993	466.450012	0.068174
1994	459.269989	-0.015513
1995	615.929993	0.293495
1996	740.739990	0.184516
1997	970.429993	0.270090
1998	1229.229980	0.236404
1999	1469.250000	0.178364
2000	1320.280029	-0.106908
2001	1148.079956	-0.139753
2002	879.820007	-0.266129
2003	1111.920044	0.234126
2004	1211.920044	0.086118
2005	1248.290039	0.029569
2006	1418.300049	0.127684
2007	1468.359985	0.034687
2008	903.250000	-0.485902
2009	1115.099976	0.210700
2010	1257.640015	0.120293
2011	1257.599976	-0.000032
2012	1426.189941	0.125801
2013	1848.359985	0.259292
2014	2058.899902	0.107873
2015	2043.939941	-0.007293
2016	2238.830078	0.091074
2017	2673.610107	0.177476
2018	2485.739990	-0.072859

In [281...]

```
years = annual.index.size
years
```

Out[281...]

30

In [282...]

```
windows = [year for year in range(30, 0, -1)]
windows
```

Out[282...]

[30,
29,
28,
27,
26,

25,
24,
23,
22,
21,
20,
19,
18,
17,
16,
15,
14,
13,
12,
11,
10,
9,
8,
7,
6,
5,
4,
3,
2,
1]

In [283...]

```
for year in windows:  
    annual["{}Y".format(year)] = annual.Return.rolling(year).mean()
```

In [284...]

annual

Out[284...]

Close **Return** **30Y** **29Y** **28Y** **27Y** **26Y** **25Y** **24Y** **23Y** ...

	Close	Return	30Y	29Y	28Y	27Y	26Y	25Y	24Y	23Y	...
Date											
2005	1248.290039	0.029569	NaN	0.0							
2006	1418.300049	0.127684	NaN	0.0							
2007	1468.359985	0.034687	NaN	0.0							
2008	903.250000	-0.485902	NaN	-0.0							
2009	1115.099976	0.210700	NaN	-0.0							
2010	1257.640015	0.120293	NaN	-0.0							
2011	1257.599976	-0.000032	NaN	0.065667	0.0						
2012	1426.189941	0.125801	NaN	NaN	NaN	NaN	NaN	NaN	0.068173	0.060659	0.0
2013	1848.359985	0.259292	NaN	NaN	NaN	NaN	NaN	0.075818	0.068936	0.074882	0.0
2014	2058.899902	0.107873	NaN	NaN	NaN	NaN	0.077051	0.070493	0.076257	0.069418	0.0
2015	2043.939941	-0.007293	NaN	NaN	NaN	0.073927	0.067501	0.072915	0.066222	0.067203	0.0
2016	2238.830078	0.091074	NaN	NaN	0.074539	0.068374	0.073613	0.067216	0.068197	0.068198	0.0
2017	2673.610107	0.177476	NaN	0.078089	0.072271	0.077460	0.071457	0.072568	0.072751	0.076589	0.0
2018	2485.739990	-0.072859	0.073057	0.067266	0.072092	0.066112	0.066975	0.066927	0.070362	0.060661	0.0

30 rows × 32 columns

In [285...]

```
triangle = annual.drop(columns = ["Close", "Return"])
```

In [286...]

```
triangle
```

Out[286...]

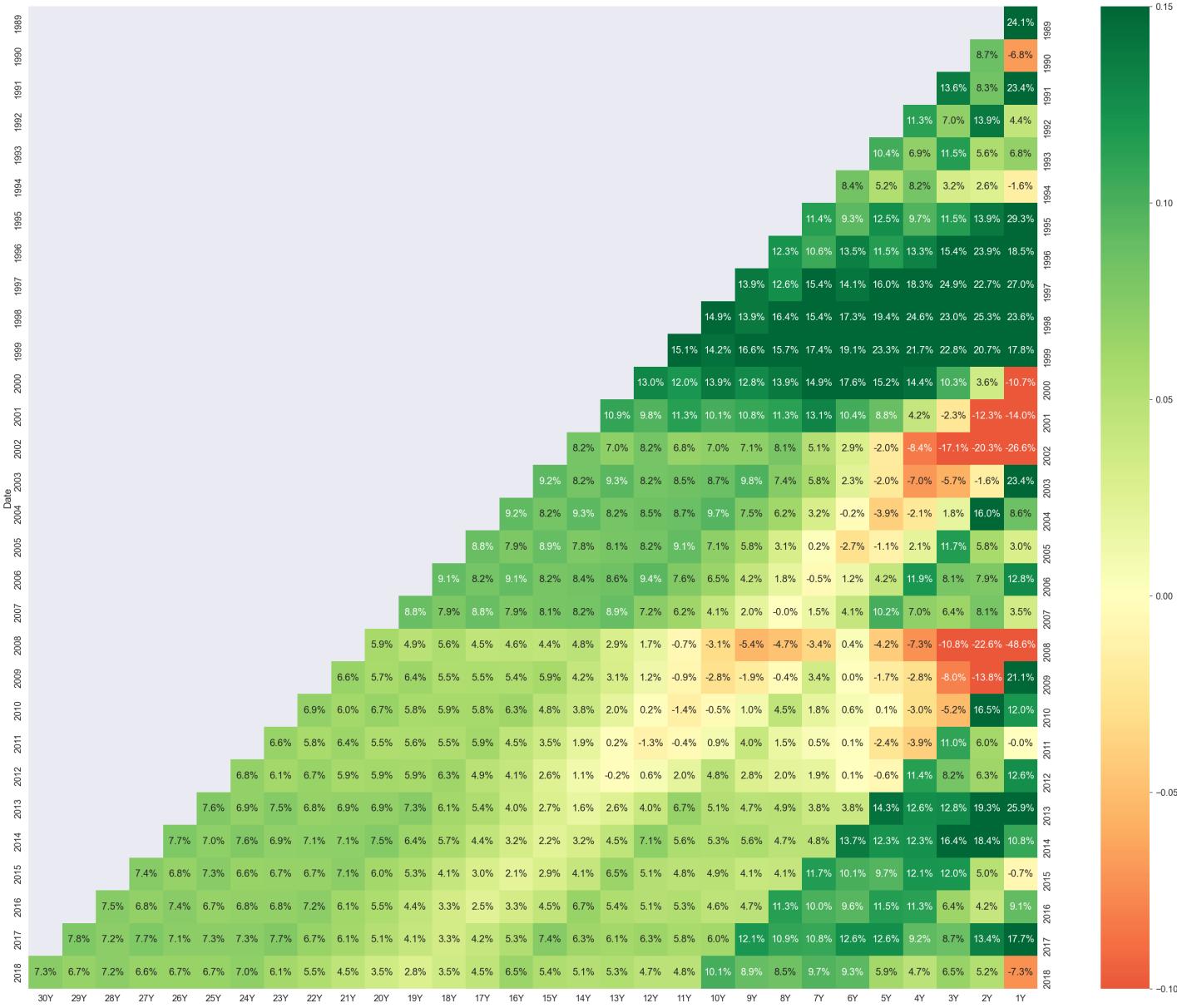
	30Y	29Y	28Y	27Y	26Y	25Y	24Y	23Y	22Y	21Y	...	10
Date												
1989	NaN	...	NaN									
1990	NaN	...	NaN									
1991	NaN	...	NaN									
1992	NaN	...	NaN									
1993	NaN	...	NaN									
1994	NaN	...	NaN									
1995	NaN	...	NaN									
1996	NaN	...	NaN									
1997	NaN	...	NaN									
1998	NaN	...	0.1487!									
1999	NaN	...	0.1424!									
2000	NaN	...	0.1385!									
2001	NaN	...	0.1012!									

Date	30Y	29Y	28Y	27Y	26Y	25Y	24Y	23Y	22Y	21Y	...	10
2002	NaN	...	0.0702									
2003	NaN	...	0.0868									
2004	NaN	...	0.0970									
2005	NaN	...	0.0706									
2006	NaN	...	0.0649									
2007	NaN	...	0.0414									
2008	NaN	...	-0.0308									
2009	NaN	0.066195	...	-0.0275								
2010	NaN	0.068654	0.060447	...	-0.0048							
2011	NaN	0.065667	0.057698	0.063676	...	0.0091						
2012	NaN	NaN	NaN	NaN	NaN	NaN	0.068173	0.060659	0.066500	0.058546	...	0.0483
2013	NaN	NaN	NaN	NaN	NaN	0.075818	0.068936	0.074882	0.067671	0.068813	...	0.0508
2014	NaN	NaN	NaN	NaN	0.077051	0.070493	0.076257	0.069418	0.070589	0.070704	...	0.0529
2015	NaN	NaN	NaN	0.073927	0.067501	0.072915	0.066222	0.067203	0.067158	0.071095	...	0.0493
2016	NaN	NaN	0.074539	0.068374	0.073613	0.067216	0.068197	0.068198	0.072003	0.061456	...	0.0456
2017	NaN	0.078089	0.072271	0.077460	0.071457	0.072568	0.072751	0.076589	0.066730	0.061121	...	0.0599
2018	0.073057	0.067266	0.072092	0.066112	0.066975	0.066927	0.070362	0.060661	0.055031	0.044790	...	0.1012

30 rows × 30 columns

In [301]:

```
# This is made using seaborn sns.heatmap(triangle)
# annot = True writes the value
# fmt = ".1%" Gives the amount of decimals and that there should be a percent at the end
plt.figure(figsize = (50, 40))
sns.set(font_scale = 1.8)
sns.heatmap(triangle, annot = True, fmt = ".1%", cmap = "RdYlGn", vmin = -0.10, vmax = 0.1
plt.tick_params(axis = "y", labelright = True)
plt.show()
```



New Section

The S&P 500 Dollar Triangles

In [272...]

```
import pandas as pd
import numpy as np
import seaborn as sns

import matplotlib.pyplot as plt
plt.style.use("seaborn")
```

In [273...]

```
sp500 = pd.read_csv("SP500.csv", parse_dates = ["Date"], index_col = "Date", usecols = [
```

In [274...]

```
sp500.head()
```

Out[274...]

Close

Date

1970-12-31 92.150002

Close

Date

Date	
1971-01-04	91.150002
1971-01-05	91.800003
1971-01-06	92.349998
1971-01-07	92.379997

In [275...]

```
sp500 = sp500.loc["1988-12-30" : "2018-12-31"].copy()
```

In [276...]

```
sp500.head()
```

Out[276...]

Close

Date	
1988-12-30	277.720001
1989-01-03	275.309998
1989-01-04	279.429993
1989-01-05	280.010010
1989-01-06	280.670013

In [277...]

```
annual = sp500.resample("A", kind = "period").last()  
annual
```

Out[277...]

Close

Date	
1988	277.720001
1989	353.399994
1990	330.220001
1991	417.089996
1992	435.709991
1993	466.450012
1994	459.269989
1995	615.929993
1996	740.739990
1997	970.429993
1998	1229.229980
1999	1469.250000
2000	1320.280029
2001	1148.079956
2002	879.820007

Close

Date

2003	1111.920044
2004	1211.920044
2005	1248.290039
2006	1418.300049
2007	1468.359985
2008	903.250000
2009	1115.099976
2010	1257.640015
2011	1257.599976
2012	1426.189941
2013	1848.359985
2014	2058.899902
2015	2043.939941
2016	2238.830078
2017	2673.610107
2018	2485.739990

Creates a log of the returns.

In [278...]

```
annual["Return"] = np.log(annual.Close / annual.Close.shift())
```

In [279...]

```
annual.dropna(inplace = True)
```

In [280...]

```
annual
```

Out[280...]

Close Return

Date		
1989	353.399994	0.240987
1990	330.220001	-0.067841
1991	417.089996	0.233543
1992	435.709991	0.043675
1993	466.450012	0.068174
1994	459.269989	-0.015513
1995	615.929993	0.293495
1996	740.739990	0.184516
1997	970.429993	0.270090

Close Return

Date

1998	1229.229980	0.236404
1999	1469.250000	0.178364
2000	1320.280029	-0.106908
2001	1148.079956	-0.139753
2002	879.820007	-0.266129
2003	1111.920044	0.234126
2004	1211.920044	0.086118
2005	1248.290039	0.029569
2006	1418.300049	0.127684
2007	1468.359985	0.034687
2008	903.250000	-0.485902
2009	1115.099976	0.210700
2010	1257.640015	0.120293
2011	1257.599976	-0.000032
2012	1426.189941	0.125801
2013	1848.359985	0.259292
2014	2058.899902	0.107873
2015	2043.939941	-0.007293
2016	2238.830078	0.091074
2017	2673.610107	0.177476
2018	2485.739990	-0.072859

In [281...]

```
years = annual.index.size
years
```

30

Out[281...]

In [282...]

```
windows = [year for year in range(30, 0, -1)]
windows
```

Out[282...]

```
[30,
 29,
 28,
 27,
 26,
 25,
 24,
 23,
 22,
 21,
 20,
 19,
 18,
```

```
17,  
16,  
15,  
14,  
13,  
12,  
11,  
10,  
9,  
8,  
7,  
6,  
5,  
4,  
3,  
2,  
1]
```

Differing code between the two triangles.

In [302...]

```
for year in windows:  
    annual["{}Y".format(year)] = np.exp(year * annual.Return.rolling(year).mean()) * 100
```

In [303...]

```
annual
```

Out[303...]

	Close	Return	30Y	29Y	28Y	27Y	26Y	25Y	24Y
Date									
1989	353.399994	0.240987	NaN						
1990	330.220001	-0.067841	NaN						
1991	417.089996	0.233543	NaN						
1992	435.709991	0.043675	NaN						
1993	466.450012	0.068174	NaN						
1994	459.269989	-0.015513	NaN						
1995	615.929993	0.293495	NaN						
1996	740.739990	0.184516	NaN						
1997	970.429993	0.270090	NaN						
1998	1229.229980	0.236404	NaN						
1999	1469.250000	0.178364	NaN						
2000	1320.280029	-0.106908	NaN						
2001	1148.079956	-0.139753	NaN						
2002	879.820007	-0.266129	NaN						
2003	1111.920044	0.234126	NaN						
2004	1211.920044	0.086118	NaN						
2005	1248.290039	0.029569	NaN						
2006	1418.300049	0.127684	NaN						
2007	1468.359985	0.034687	NaN						

	Close	Return	30Y	29Y	28Y	27Y	26Y	25Y	24Y
Date									
2008	903.250000	-0.485902		NaN	NaN	NaN	NaN	NaN	NaN
2009	1115.099976	0.210700		NaN	NaN	NaN	NaN	NaN	NaN
2010	1257.640015	0.120293		NaN	NaN	NaN	NaN	NaN	NaN
2011	1257.599976	-0.000032		NaN	NaN	NaN	NaN	NaN	NaN
2012	1426.189941	0.125801		NaN	NaN	NaN	NaN	NaN	41
2013	1848.359985	0.259292		NaN	NaN	NaN	NaN	665.548026	523.022076
2014	2058.899902	0.107873		NaN	NaN	NaN	741.358165	582.597605	623.493397
2015	2043.939941	-0.007293		NaN	NaN	NaN	735.971458	578.364453	618.963096
2016	2238.830078	0.091074		NaN	NaN	806.146504	633.511634	677.981367	536.773862
2017	2673.610107	0.177476		NaN	962.699877	756.539375	809.645115	641.015160	613.621483
2018	2485.739990	-0.072859	895.052564	703.378617	752.752705	595.972096	570.503326	532.905976	541.237192

30 rows × 32 columns

In [304]:

triangle

Out[304]:

	30Y	29Y	28Y	27Y	26Y	25Y	24Y	23Y	22Y	21Y	...	10
Date												
1989	NaN	...	NaN									
1990	NaN	...	NaN									
1991	NaN	...	NaN									
1992	NaN	...	NaN									
1993	NaN	...	NaN									
1994	NaN	...	NaN									
1995	NaN	...	NaN									
1996	NaN	...	NaN									
1997	NaN	...	NaN									
1998	NaN	...	0.1487!									
1999	NaN	...	0.1424!									
2000	NaN	...	0.1385!									
2001	NaN	...	0.1012!									
2002	NaN	...	0.0702!									
2003	NaN	...	0.0868!									
2004	NaN	...	0.0970!									
2005	NaN	...	0.0706!									
2006	NaN	...	0.0649!									

	30Y	29Y	28Y	27Y	26Y	25Y	24Y	23Y	22Y	21Y	...	10
Date												
2007	NaN	...	0.0414									
2008	NaN	...	-0.0308									
2009	NaN	0.066195	...	-0.02758								
2010	NaN	0.068654	0.060447	...	-0.00486							
2011	NaN	0.065667	0.057698	0.063676	...	0.00911						
2012	NaN	NaN	NaN	NaN	NaN	NaN	0.068173	0.060659	0.066500	0.058546	...	0.04830
2013	NaN	NaN	NaN	NaN	NaN	0.075818	0.068936	0.074882	0.067671	0.068813	...	0.05082
2014	NaN	NaN	NaN	NaN	0.077051	0.070493	0.076257	0.069418	0.070589	0.070704	...	0.05299
2015	NaN	NaN	NaN	0.073927	0.067501	0.072915	0.066222	0.067203	0.067158	0.071095	...	0.04931
2016	NaN	NaN	0.074539	0.068374	0.073613	0.067216	0.068197	0.068198	0.072003	0.061456	...	0.04564
2017	NaN	0.078089	0.072271	0.077460	0.071457	0.072568	0.072751	0.076589	0.066730	0.061121	...	0.05991
2018	0.073057	0.067266	0.072092	0.066112	0.066975	0.066927	0.070362	0.060661	0.055031	0.044790	...	0.10123

30 rows \times 30 columns

In [305...

```
triangle = annual.drop(columns = ["Close", "Return"])
```

In [306...

triangle

Out[306...]

30Y **29Y** **28Y** **27Y** **26Y** **25Y** **24Y** **23Y** **22Y**

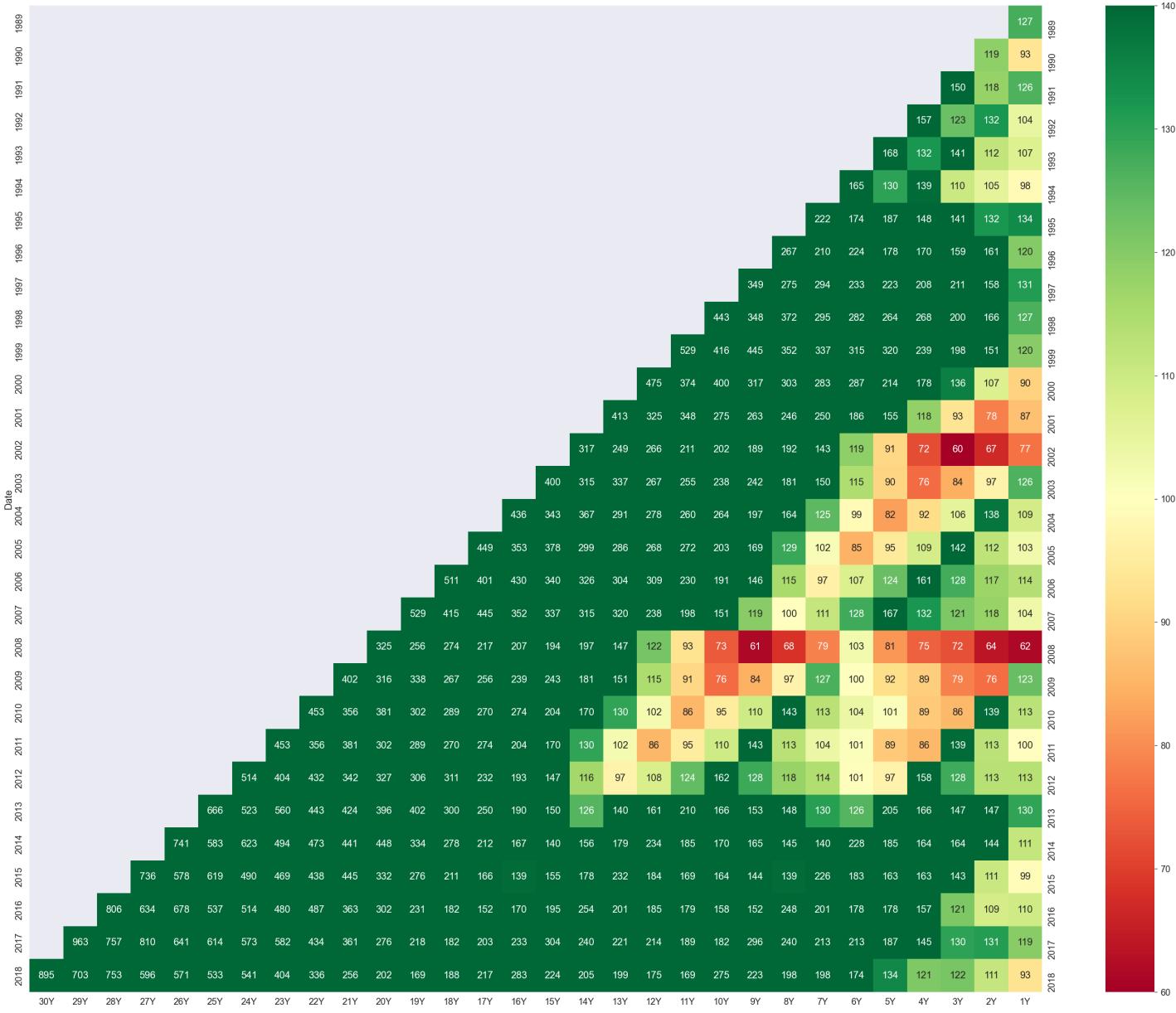
Date	30Y	29Y	28Y	27Y	26Y	25Y	24Y	23Y	22Y
2004	NaN								
2005	NaN								
2006	NaN								
2007	NaN								
2008	NaN								
2009	NaN	4							
2010	NaN	452.844595	3						
2011	NaN	452.830178	355.857385						
2012	NaN	NaN	NaN	NaN	NaN	NaN	513.535192	403.562526	431.890841
2013	NaN	NaN	NaN	NaN	NaN	665.548026	523.022076	559.735927	443.156154
2014	NaN	NaN	NaN	741.358165	582.597605	623.493397	493.634449	472.539061	4
2015	NaN	NaN	735.971458	578.364453	618.963096	490.047702	469.105594	438.190565	4
2016	NaN	806.146504	633.511634	677.981367	536.773862	513.834919	479.972134	487.475805	3
2017	962.699877	756.539375	809.645115	641.015160	613.621483	573.182557	582.143439	434.076947	3
2018	895.052564	703.378617	752.752705	595.972096	570.503326	532.905976	541.237192	403.575085	335.575239

30 rows × 30 columns

If you put 100 dollars into the S&P500

In [310]:

```
# NEW Return on Dollars Triangle
# This is made using seaborn sns.heatmap(triangle)
# annot = True writes the value
# fmt = ".1%" Gives the amount of decimals and that there should be a percent at the end
plt.figure(figsize = (50, 40))
sns.set(font_scale = 1.8)
sns.heatmap(triangle, annot = True, fmt = ".0f", cmap = "RdYlGn", vmin = 60, vmax = 140, cbar_kws = {"label": "Value", "shrink": 0.5}, square = True, xticklabels = 1, yticklabels = 1)
plt.tick_params(axis = "y", labelright = True)
plt.show()
```



Exponentially weighted moving averages.

These are moving averages that give more weight to the more recent values.

In [358...]

```
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use("seaborn")
```

In [359...]

```
sp500 = pd.read_csv("SP500.csv", parse_dates = ["Date"], index_col = "Date", usecols = [
```

In [360...]

```
sp500.head()
```

Out[360...]

Close

	Date
1970-12-31	92.150002
1971-01-04	91.150002
1971-01-05	91.800003

Close**Date**

```
1971-01-06 92.349998
```

```
1971-01-07 92.379997
```

```
In [361...]
```

```
sp500 = sp500.loc["2008-12-31": "2018-12-31"].copy()
```

```
In [362...]
```

```
sp500.Close.rolling(window = 10).mean()
```

```
Out[362...]
```

```
Date
2008-12-31      NaN
2009-01-02      NaN
2009-01-05      NaN
2009-01-06      NaN
2009-01-07      NaN
...
2018-12-21  2565.915991
2018-12-24  2537.254004
2018-12-26  2520.345996
2018-12-27  2504.121997
2018-12-28  2487.641992
Name: Close, Length: 2516, dtype: float64
```

```
In [363...]
```

```
# Span helps make this calculation possible. There is also a half-life option.
sp500.Close.ewm(span = 10, min_periods = 10).mean()
```

```
Out[363...]
```

```
Date
2008-12-31      NaN
2009-01-02      NaN
2009-01-05      NaN
2009-01-06      NaN
2009-01-07      NaN
...
2018-12-21  2547.124619
2018-12-24  2511.483797
2018-12-26  2503.523098
2018-12-27  2500.851640
2018-12-28  2498.104067
Name: Close, Length: 2516, dtype: float64
```

```
In [364...]
```

```
sp500["SWA"] = sp500.Close.rolling(window = 10).mean()
sp500["EWA"] = sp500.Close.ewm(span = 100, min_periods = 100).mean()
```

```
In [365...]
```

```
sp500
```

```
Out[365...]
```

	Close	SWA	EWA
--	--------------	------------	------------

Date

2008-12-31	903.250000	NaN	NaN
2009-01-02	931.799988	NaN	NaN
2009-01-05	927.450012	NaN	NaN
2009-01-06	934.700012	NaN	NaN
2009-01-07	906.650024	NaN	NaN

Close SWA EWA

Date

2018-12-21	2416.620117	2565.915991	2732.088580	
2018-12-24	2351.100098	2537.254004	2724.544253	
2018-12-26	2467.699951	2520.345996	2719.458228	
2018-12-27	2488.830078	2504.121997	2714.891334	
2018-12-28	2485.739990	2487.641992	2710.353683	

2516 rows × 3 columns

In [367...]

```
# Simple Weighted Averages Compared With Exponentially Weighted Averages
sp500.iloc[:, -2:].plot(figsize = (15, 10), fontsize = 15)
plt.legend(fontsize = 15)
plt.show()
```



Expanding Windows

Let's include more and more data points as they become available to become part of the statistic.

In [369...]

```
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use("seaborn")
```

In [370...]

```
sp500 = pd.read_csv("SP500.csv", parse_dates = ["Date"], index_col = "Date", usecols = [
```

```
In [371... sp500 = sp500.loc["2008-12-31": "2018-12-31"].copy()
```

```
In [372... sp500.head()
```

```
Out[372...          Close
```

Date	Close
2008-12-31	903.250000
2009-01-02	931.799988
2009-01-05	927.450012
2009-01-06	934.700012
2009-01-07	906.650024

```
In [373... sp500.Close.rolling(10).mean()
```

```
Out[373...          Date
2008-12-31      NaN
2009-01-02      NaN
2009-01-05      NaN
2009-01-06      NaN
2009-01-07      NaN
...
2018-12-21  2565.915991
2018-12-24  2537.254004
2018-12-26  2520.345996
2018-12-27  2504.121997
2018-12-28  2487.641992
Name: Close, Length: 2516, dtype: float64
```

```
In [374... sp500.Close.expanding().mean()
```

```
Out[374...          Date
2008-12-31  903.250000
2009-01-02  917.524994
2009-01-05  920.833333
2009-01-06  924.300003
2009-01-07  920.770007
...
2018-12-21  1764.040123
2018-12-24  1764.273732
2018-12-26  1764.553536
2018-12-27  1764.841518
2018-12-28  1765.128044
Name: Close, Length: 2516, dtype: float64
```

There is a .expanding() method.

```
In [375... sp500["SMA50"] = sp500.Close.rolling(10).mean()
sp500["EXP"] = sp500.Close.expanding().mean()
```

```
In [376... sp500.head()
```

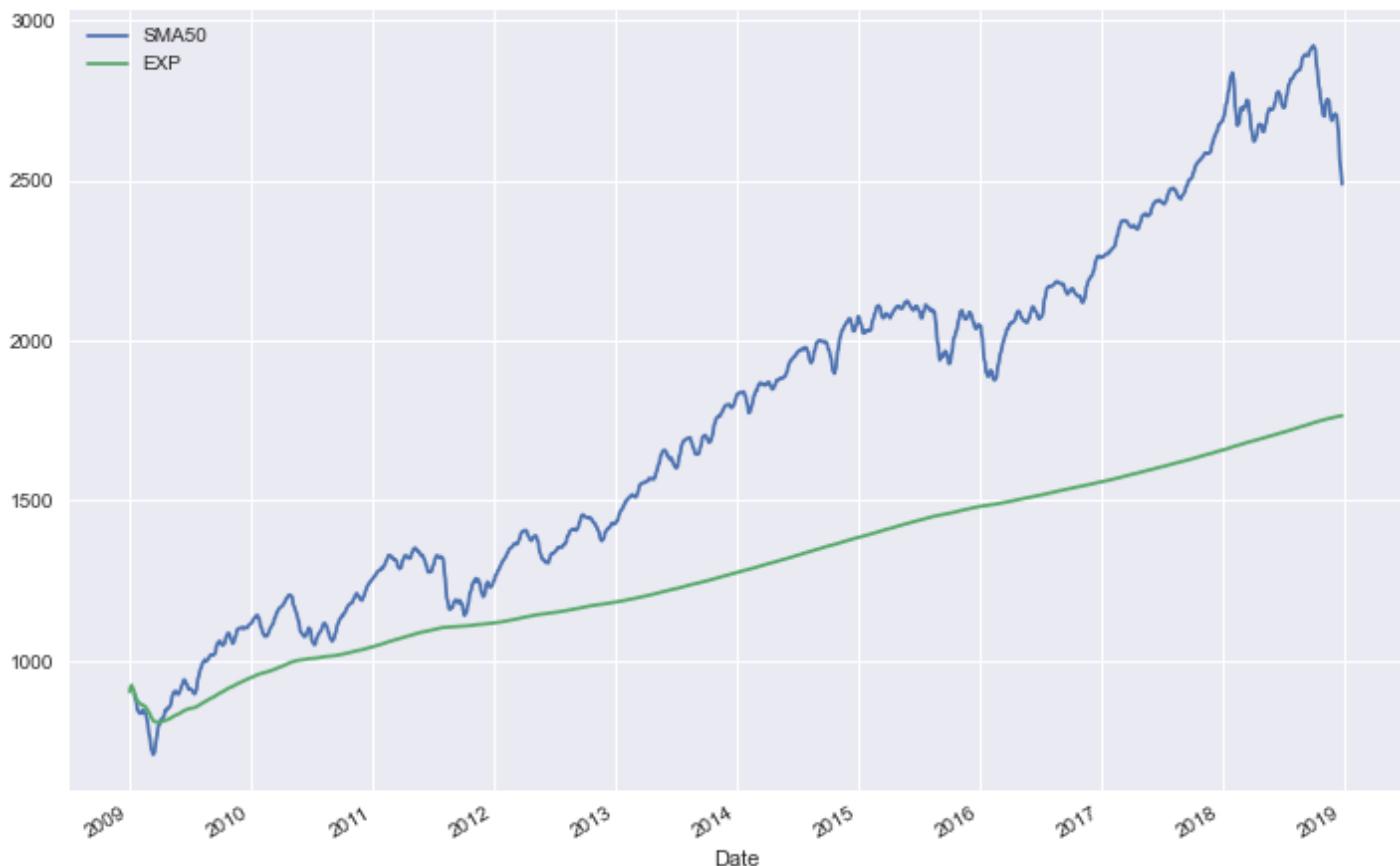
```
Out[376...          Close  SMA50      EXP
```

Date	Close	SMA50	EXP
Date			
2008-12-31	903.250000	NaN	903.250000
2009-01-02	931.799988	NaN	917.524994
2009-01-05	927.450012	NaN	920.833333
2009-01-06	934.700012	NaN	924.300003
2009-01-07	906.650024	NaN	920.770007

```
In [377]: sp500.Close.expanding(min_periods = 5).mean()
```

```
Out[377]: Date
2008-12-31      NaN
2009-01-02      NaN
2009-01-05      NaN
2009-01-06      NaN
2009-01-07    920.770007
...
2018-12-21  1764.040123
2018-12-24  1764.273732
2018-12-26  1764.553536
2018-12-27  1764.841518
2018-12-28  1765.128044
Name: Close, Length: 2516, dtype: float64
```

```
In [378]: sp500.iloc[:, -2:].plot(figsize = (12, 8))
plt.show()
```



Expanding Minimum

In [379...]

```
sp500["SMA50"] = sp500.Close.rolling(10).mean()  
sp500["EXP"] = sp500.Close.expanding().min()
```

In [380...]

```
sp500.iloc[:, -2:].plot(figsize = (12, 8))  
plt.show()
```



Expanding Max

In [381...]

```
sp500["SMA50"] = sp500.Close.rolling(10).mean()  
sp500["EXP"] = sp500.Close.expanding().max()
```

In [382...]

```
sp500.iloc[:, -2:].plot(figsize = (12, 8))  
plt.show()
```



There is also Rolling Correlation

Can see how the relationship between two stocks can vary over time.

In []:

Create and Analyze Financial Indexes

What is a Financial Index

A financial index is an assortment of stocks that are an aggregate of different securities. Examples include the S&P 500, NASDAQ, and the Dow Jones. There are also bond indexes and commodity indexes. Indexes in other countries included Euro stoxx50, Dax 30, Nikkei 225 etc.

Why to create an index?

They are used as a benchmark. It allows smaller users to have exposure in different stocks via mutual funds etc. It allows different investment strategies to be categorized.

How to create an index?

1. Define the Target Market/ Segment (What is the overarching theme of the stock).
2. Identify securities in the segment. Pick which stocks to include in the segment.
3. Determine how the stocks will be weighted. Will it be weighted by Price, will it be an equal weighted index, or a value Weighted index (market cap).
4. Is the return a price index or a total return? Total return is adjusted to take included dividends in earnings.
Most of the time dividends are not included.

Data Import, Visualization & Normalization

In [4]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import yfinance as yf

plt.style.use("seaborn")
```

In [5]:

```
stocks = yf.download(["AMZN", "BA", "DIS", "IBM", "KO", "MSFT"], start = "2014-01-01", end = "2014-01-02")
[*****100%*****] 6 of 6 completed
```

In [7]:

```
stocks.head()
```

Out[7]:

Date	AMZN	BA	DIS	IBM	KO	MSFT	AMZN	BA	DIS
2014-01-02	397.970001	116.807945	70.192520	125.096390	31.200077	31.798315	397.970001	136.669998	76.269997
2014-01-03	396.440002	117.619896	70.045273	125.844833	31.046608	31.584404	396.440002	137.619995	76.110001
2014-01-06	393.630005	118.295044	69.778374	125.413277	30.900801	30.916939	393.630005	138.410004	75.820000
2014-01-07	398.029999	120.089859	70.256950	127.914818	30.992886	31.156536	398.029999	140.509995	76.339996

	AMZN	BA	DIS	IBM	KO	MSFT	AMZN	BA	DIS
--	------	----	-----	-----	----	------	------	----	-----

Date

2014-01-08	401.920013	120.354843	69.226196	126.741592	30.647587	30.600323	401.920013	140.820007	75.220001	179.7
------------	------------	------------	-----------	------------	-----------	-----------	------------	------------	-----------	-------

5 rows × 36 columns

In [9]:

```
stocks.tail()
```

Out[9]:

	AMZN	BA	DIS	IBM	KO	MSFT	AMZN	BA	DIS	
Date										
2018-12-21	1377.449951	295.930267	102.961685	89.908890	42.892387	94.791809	1377.449951	304.549988	104.220001	1
2018-12-24	1343.959961	285.834351	99.138412	87.177742	41.440701	90.835304	1343.959961	294.160004	100.349998	1
2018-12-26	1470.900024	305.044769	104.552246	90.273575	42.324337	97.040253	1470.900024	313.929993	105.830002	1
2018-12-27	1461.640015	308.163971	105.233910	92.210503	42.856316	97.638557	1461.640015	317.140015	106.519997	1
2018-12-28	1478.020020	307.425446	106.004501	91.602676	42.558773	96.876221	1478.020020	316.380005	107.300003	1

5 rows × 36 columns

In [10]:

```
stocks.to_csv("index_stocks.csv")
```

In [11]:

```
stocks = pd.read_csv("index_stocks.csv", header = [0, 1], index_col = [0], parse_dates = [1])
```

In [13]:

```
stocks.head(10)
```

Out[13]:

	AMZN	BA	DIS	IBM	KO	MSFT
Date						
2014-01-02	397.970001	136.669998	76.269997	177.370941	40.660000	37.160000
2014-01-03	396.440002	137.619995	76.110001	178.432129	40.459999	36.910000
2014-01-06	393.630005	138.410004	75.820000	177.820267	40.270000	36.130001
2014-01-07	398.029999	140.509995	76.339996	181.367111	40.389999	36.410000
2014-01-08	401.920013	140.820007	75.220001	179.703629	39.939999	35.759998
2014-01-09	401.010010	142.130005	74.900002	179.139572	39.730000	35.529999
2014-01-10	397.660004	141.899994	75.389999	179.024857	40.130001	36.040001
2014-01-13	390.980011	140.699997	73.269997	176.061188	39.529999	34.980000

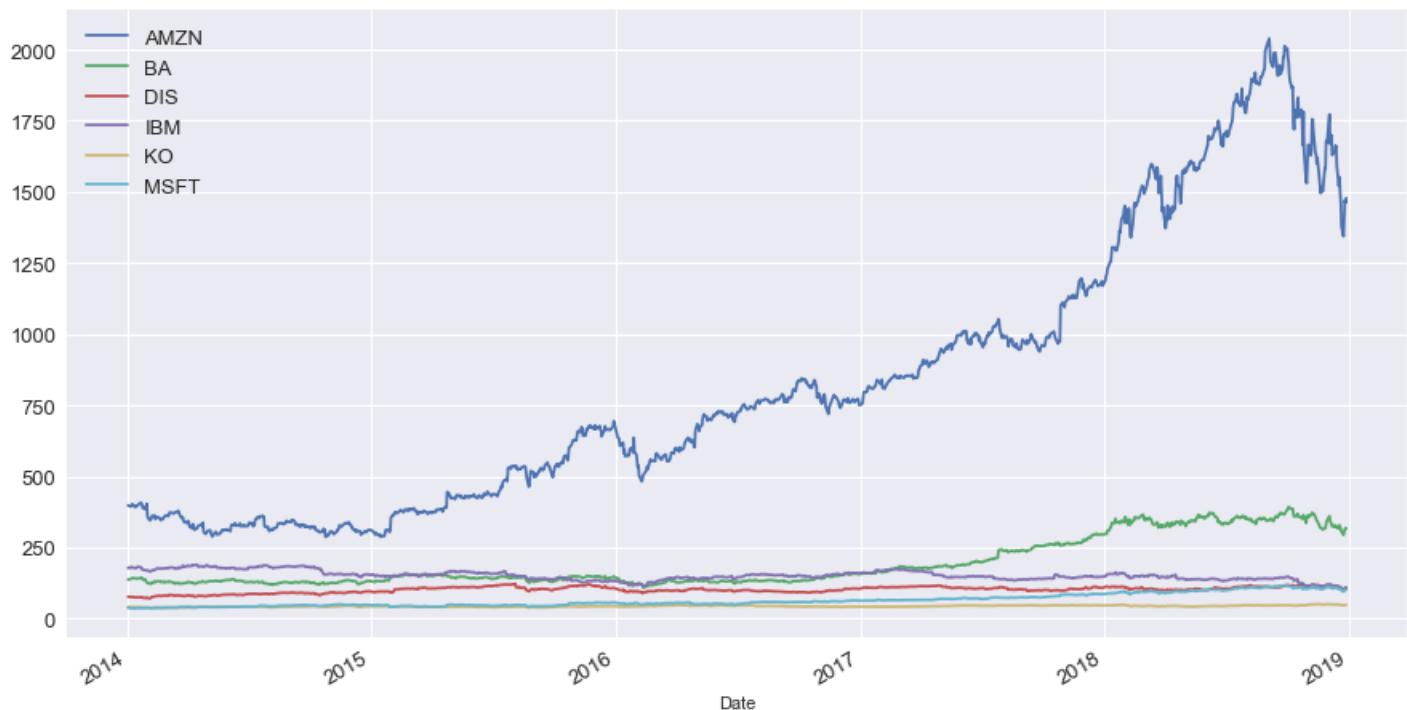
AMZN BA DIS IBM KO MSFT

Date

2014-01-14	397.540009	140.009995	74.449997	177.743790	39.689999	35.779999
2014-01-15	395.869995	140.619995	74.279999	179.483749	39.759998	36.759998

In [14]:

```
stocks.plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```



Remember

Make the Normalized chart by dividing each column by the index.

Don't forget to multiply by 100 to get base 100. `div()` and `.mul()` methods could have been used as well for the code below.

In [25]:

```
stocks_normal = (stocks.iloc[0:] / stocks.loc["2014-01-02"]) * 100
```

In [26]:

```
stocks_normal.head()
```

Out[26]:

AMZN BA DIS IBM KO MSFT

Date

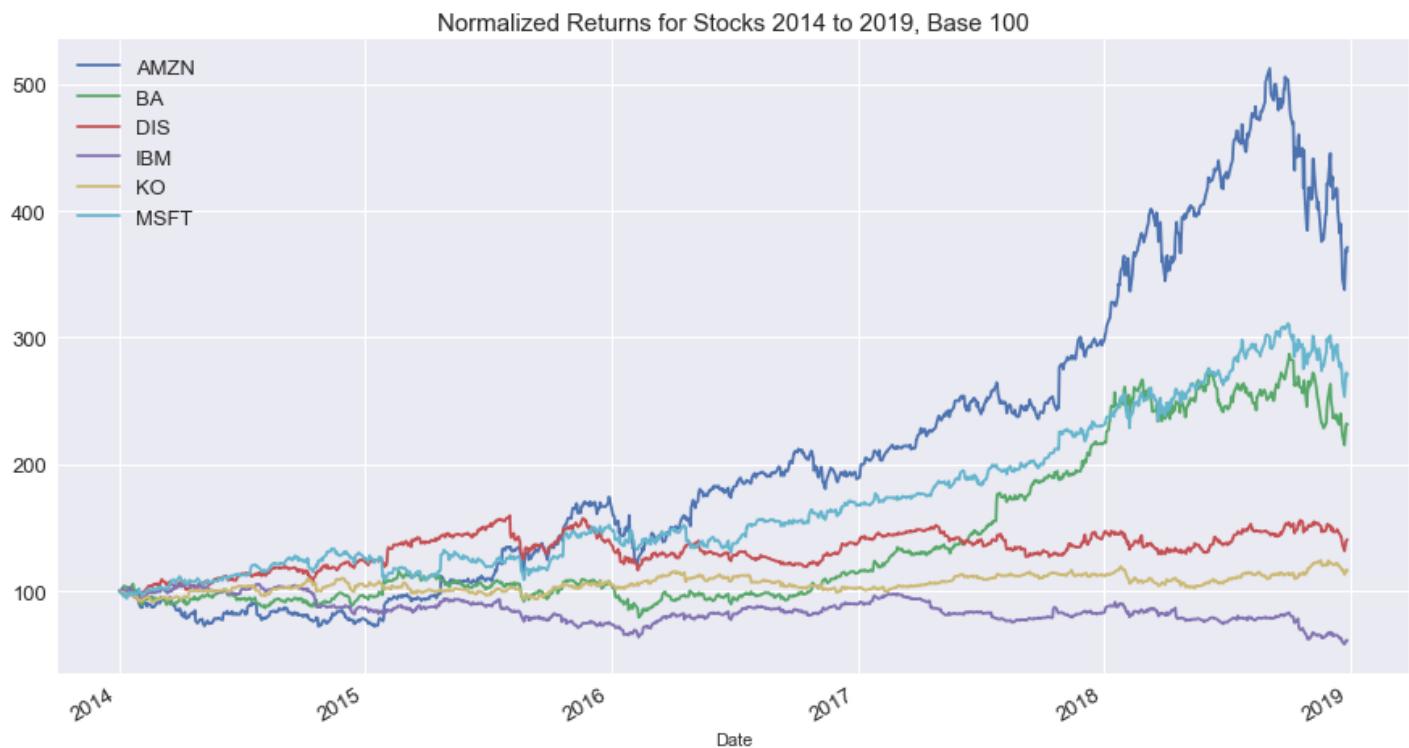
2014-01-02	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
2014-01-03	99.615549	100.695103	99.790224	100.598287	99.508114	99.327234
2014-01-06	98.909466	101.273144	99.409995	100.253325	99.040828	97.228206
2014-01-07	100.015076	102.809685	100.091779	102.253002	99.335956	97.981701

Date

2014-01-08	100.992540	103.036518	98.623318	101.315146	98.229215	96.232504
------------	------------	------------	-----------	------------	-----------	-----------

In [28]:

```
stocks_normal.plot(figsize = (15, 8), fontsize = 13)
plt.title("Normalized Returns for Stocks 2014 to 2019, Base 100", fontsize = 15)
plt.legend(fontsize = 13)
plt.show()
```



Price- Weighted Index

The constituents are weighted by their stock prices.

If stock A has a price of 2 and stock B has a price of 8.

This means that stock A is weighted 20% and stock B is weighted 80%. Weighting is found by `price_of_one_stock/ total_price_of_stock`.

A more general approach

Timestamp	Stock A	Stock B	Sum
0	5 0.33	10 0.67	15
1	Prices (weights)	8	8

Timestamp	Stock A	Stock B	Weighted Av.	Index
0	na	na	na	100.00
1	Returns	0.6	-0.2	0.067

$$0.6 * 0.33 + (-0.2) * 0.67 = 0.067$$

$$100 * (1 + 0.067) = 106.67$$

Advantages of Price Weighted Index

1. Easy to Compute and little data is needed.
2. Intuitive (one share per constituent).
3. Self-Rebalancing (minimum transaction costs and taxes).

Disadvantages

1. Concentrated positions in stocks with higher prices.
2. Marketcap is a better reflection of a company's economic importance.
3. Does not reflect a typical portfolio construction (equal number of shares).
4. Stock splits can complicate construction. (Not an issue because yfinance includes the impact of stock splits).

Creating Price-weighted Index

In [30]:

```
stocks.head()
```

Out[30]:

	AMZN	BA	DIS	IBM	KO	MSFT
Date						
2014-01-02	397.970001	136.669998	76.269997	177.370941	40.660000	37.160000
2014-01-03	396.440002	137.619995	76.110001	178.432129	40.459999	36.910000
2014-01-06	393.630005	138.410004	75.820000	177.820267	40.270000	36.130001
2014-01-07	398.029999	140.509995	76.339996	181.367111	40.389999	36.410000
2014-01-08	401.920013	140.820007	75.220001	179.703629	39.939999	35.759998

Making norm = stocks_normal

In [31]:

```
norm = stocks_normal
```

In [32]:

```
norm.head()
```

Out[32]:

	AMZN	BA	DIS	IBM	KO	MSFT
Date						
2014-01-02	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
2014-01-03	99.615549	100.695103	99.790224	100.598287	99.508114	99.327234
2014-01-06	98.909466	101.273144	99.409995	100.253325	99.040828	97.228206
2014-01-07	100.015076	102.809685	100.091779	102.253002	99.335956	97.981701
2014-01-08	100.992540	103.036518	98.623318	101.315146	98.229215	96.232504

The sum, sums up all the values accross the columns because there is only one of each stock in the price weighted option.

In [33]:

```
stocks.sum(axis = 1)
```

Out[33]:

```
Date
2014-01-02    866.100937
2014-01-03    865.972126
2014-01-06    862.080276
2014-01-07    873.047100
2014-01-08    873.363647
...
2018-12-21    2038.081131
2018-12-24    1981.399345
2018-12-26    2144.651409
2018-12-27    2142.786316
2018-12-28    2157.349300
Length: 1257, dtype: float64
```

Normalize the data by dividing all of the data by the first row of all of the values.

This means the first AMZN date price is used to divide the rest of Amazon future prices etc.

In [35]:

```
stocks.sum(axis = 1).div(stocks.sum(axis = 1)[0]).mul(100)
```

Out[35]:

```
Date
2014-01-02    100.000000
2014-01-03    99.985127
2014-01-06    99.535775
2014-01-07    100.802004
2014-01-08    100.838552
...
2018-12-21    235.316814
2018-12-24    228.772336
2018-12-26    247.621417
2018-12-27    247.406073
2018-12-28    249.087515
Length: 1257, dtype: float64
```

In [36]:

```
norm["PWI"] = stocks.sum(axis = 1).div(stocks.sum(axis = 1)[0]).mul(100)
```

In [37]:

```
norm.head()
```

Out[37]:

	AMZN	BA	DIS	IBM	KO	MSFT	PWI
Date							
2014-01-02	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
2014-01-03	99.615549	100.695103	99.790224	100.598287	99.508114	99.327234	99.985127
2014-01-06	98.909466	101.273144	99.409995	100.253325	99.040828	97.228206	99.535775
2014-01-07	100.015076	102.809685	100.091779	102.253002	99.335956	97.981701	100.802004
2014-01-08	100.992540	103.036518	98.623318	101.315146	98.229215	96.232504	100.838552

In [38]:

```
norm.plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```



In [39]:

```
norm.loc[:, ["AMZN", "PWI"]].plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```



```
In [41]: norm.loc[:, ["KO", "PWI"]].plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```



```
In [42]: stocks.head()
```

```
Out[42]:
```

	AMZN	BA	DIS	IBM	KO	MSFT
Date						
2014-01-02	397.970001	136.669998	76.269997	177.370941	40.660000	37.160000
2014-01-03	396.440002	137.619995	76.110001	178.432129	40.459999	36.910000
2014-01-06	393.630005	138.410004	75.820000	177.820267	40.270000	36.130001

AMZN BA DIS IBM KO MSFT

Date

2014-01-07	398.029999	140.509995	76.339996	181.367111	40.389999	36.410000
2014-01-08	401.920013	140.820007	75.220001	179.703629	39.939999	35.759998

In [44]:

```
stocks.div(stocks.sum(axis = 1), axis = "rows").head(10)
```

Out[44]:

	AMZN	BA	DIS	IBM	KO	MSFT
--	------	----	-----	-----	----	------

Date

2014-01-02	0.459496	0.157799	0.088061	0.204792	0.046946	0.042905
2014-01-03	0.457798	0.158920	0.087890	0.206048	0.046722	0.042623
2014-01-06	0.456605	0.160553	0.087950	0.206269	0.046713	0.041910
2014-01-07	0.455909	0.160942	0.087441	0.207740	0.046263	0.041705
2014-01-08	0.460198	0.161239	0.086127	0.205760	0.045731	0.040945
2014-01-09	0.459642	0.162911	0.085851	0.205332	0.045539	0.040725
2014-01-10	0.457004	0.163076	0.086641	0.205741	0.046119	0.041418
2014-01-13	0.457008	0.164461	0.085644	0.205794	0.046206	0.040887
2014-01-14	0.459470	0.161821	0.086048	0.205433	0.045873	0.041354
2014-01-15	0.456717	0.162234	0.085697	0.207071	0.045871	0.042410

In [45]:

```
weights_PWI = stocks.div(stocks.sum(axis = 1), axis = "rows")
```

In [46]:

```
weights_PWI.plot(figsize = (15, 8), fontsize = 13)
```

Out[46]:



Equal Weighted Indexes

Constituents are equal weighted in monetary terms.

Invest equal amounts in all of the securities.

An example would be to invest 500 each into two stocks when the amount that you have to invest totals 1,000.

Timestamp	Stock A	Stock B	Weighted Av.	Cum. Multiple	Index
0	na	na	na	na	100.00
1	Returns	0.6	-0.2	0.2	1.2 120.00
2		-0.1	0.3	0.1	1.32 132.00

$$(1 + 0.2) * (1 + 0.1) = 1.32$$

Equal Weighted Index Advantages

Higher degree of diversification (no concentrated positions).

Intuitive (equal investment amounts per constituent).

Equal Weighted Index Disadvantages

Biased to the performance of smaller stocks.

Marketcap better reflects a company's economic importance.

Does not reflect typical portfolio construction.

A real EWI requires constant rebalancing. That is selling winners, to buy losers. This creates high transaction costs and taxes.

Creating an Equal Weighted Index

In [50]:

```
stocks.head()
```

Out[50]:

	AMZN	BA	DIS	IBM	KO	MSFT
Date						
2014-01-02	397.970001	136.669998	76.269997	177.370941	40.660000	37.160000
2014-01-03	396.440002	137.619995	76.110001	178.432129	40.459999	36.910000
2014-01-06	393.630005	138.410004	75.820000	177.820267	40.270000	36.130001
2014-01-07	398.029999	140.509995	76.339996	181.367111	40.389999	36.410000
2014-01-08	401.920013	140.820007	75.220001	179.703629	39.939999	35.759998

In [51]:

```
norm.head()
```

Out[51]:

	AMZN	BA	DIS	IBM	KO	MSFT	PWI
Date							

	AMZN	BA	DIS	IBM	KO	MSFT	PWI
Date							
2014-01-02	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
2014-01-03	99.615549	100.695103	99.790224	100.598287	99.508114	99.327234	99.985127
2014-01-06	98.909466	101.273144	99.409995	100.253325	99.040828	97.228206	99.535775
2014-01-07	100.015076	102.809685	100.091779	102.253002	99.335956	97.981701	100.802004
2014-01-08	100.992540	103.036518	98.623318	101.315146	98.229215	96.232504	100.838552

```
In [52]: ret = stocks.pct_change().dropna()
```

In [53]: `ret.head()`

Out[53]:

Date						
2014-01-03	-0.003845	0.006951	-0.002098	0.005983	-0.004919	-0.006728
2014-01-06	-0.007088	0.005741	-0.003810	-0.003429	-0.004696	-0.021132
2014-01-07	0.011178	0.015172	0.006858	0.019946	0.002980	0.007750
2014-01-08	0.009773	0.002206	-0.014671	-0.009172	-0.011141	-0.017852
2014-01-09	-0.002264	0.009303	-0.004254	-0.003139	-0.005258	-0.006432

```
In [54]: ret["Mean ret"] = ret.mean(axis = 1)
```

In [55]: `ret.head()`

Date	2014-01-03	-0.003845	0.006951	-0.002098	0.005983	-0.004919	-0.006728	-0.000776
2014-01-06	-0.007088	0.005741	-0.003810	-0.003429	-0.004696	-0.021132	-0.005736	
2014-01-07	0.011178	0.015172	0.006858	0.019946	0.002980	0.007750	0.010647	
2014-01-08	0.009773	0.002206	-0.014671	-0.009172	-0.011141	-0.017852	-0.006810	
2014-01-09	-0.002264	0.009303	-0.004254	-0.003139	-0.005258	-0.006432	-0.002007	

```
In [56]: norm["EWI"] = 100
```

In [57]: norm.head()

	AMZN	BA	DIS	IBM	KO	MSFT	PWI	EWI
--	------	----	-----	-----	----	------	-----	-----

Date

2014-01-03	99.615549	100.695103	99.790224	100.598287	99.508114	99.327234	99.985127	100
2014-01-06	98.909466	101.273144	99.409995	100.253325	99.040828	97.228206	99.535775	100
2014-01-07	100.015076	102.809685	100.091779	102.253002	99.335956	97.981701	100.802004	100
2014-01-08	100.992540	103.036518	98.623318	101.315146	98.229215	96.232504	100.838552	100

cumprod() is the culmalative product.

In [58]: `ret.Mean_ret.add(1).cumprod().mul(100)`

Out[58]:

Date	
2014-01-03	99.922419
2014-01-06	99.349274
2014-01-07	100.407085
2014-01-08	99.723360
2014-01-09	99.523180
...	
2018-12-21	175.330385
2018-12-24	169.441550
2018-12-26	179.083167
2018-12-27	180.594645
2018-12-28	180.437837
Name: Mean_ret, Length: 1256, dtype: float64	

In [59]: `norm.iloc[1:, -1] = ret.Mean_ret.add(1).cumprod().mul(100)`

In [61]: `norm.head(10)`

Out[61]:

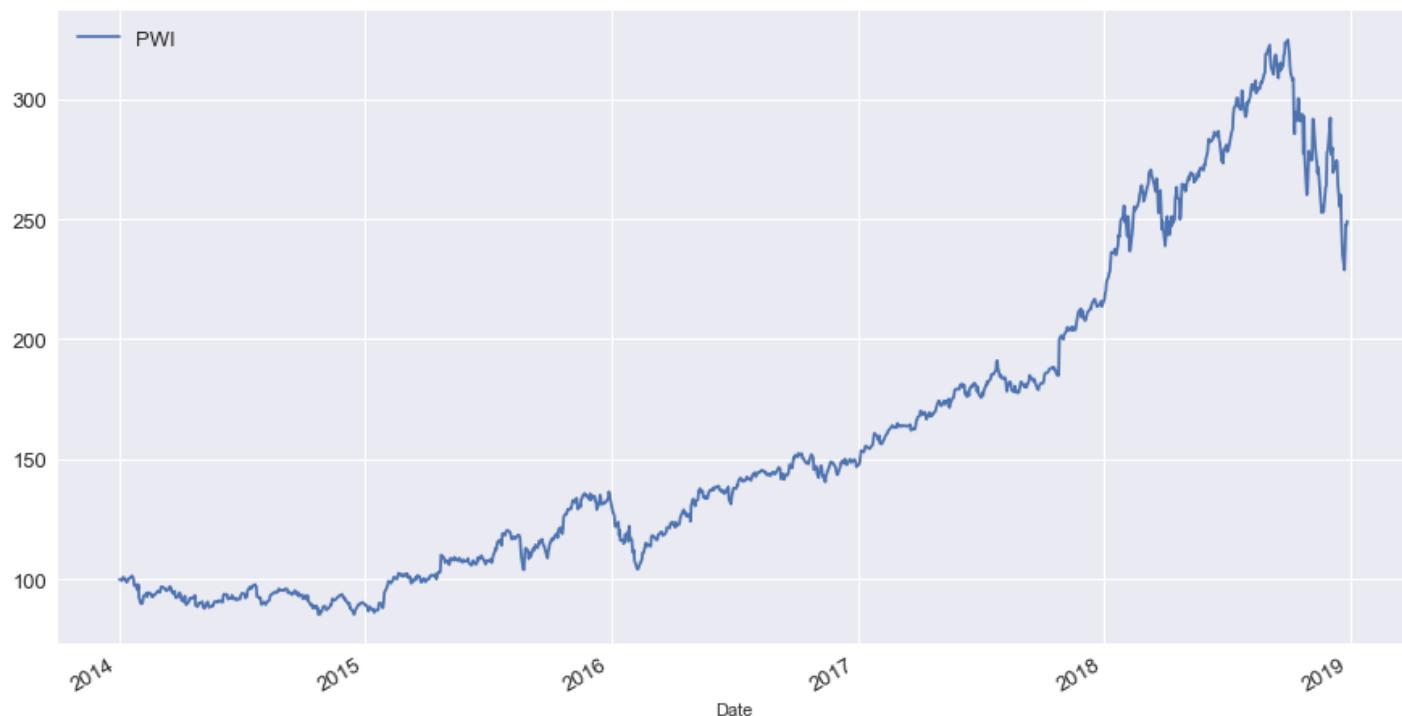
	AMZN	BA	DIS	IBM	KO	MSFT	PWI	EWI
Date								
2014-01-02	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
2014-01-03	99.615549	100.695103	99.790224	100.598287	99.508114	99.327234	99.985127	99.922419
2014-01-06	98.909466	101.273144	99.409995	100.253325	99.040828	97.228206	99.535775	99.349274
2014-01-07	100.015076	102.809685	100.091779	102.253002	99.335956	97.981701	100.802004	100.407085
2014-01-08	100.992540	103.036518	98.623318	101.315146	98.229215	96.232504	100.838552	99.723360
2014-01-09	100.763879	103.995029	98.203756	100.997137	97.712739	95.613560	100.731860	99.523180
2014-01-10	99.922105	103.826733	98.846208	100.932461	98.696511	96.986009	100.466911	99.860755
2014-01-13	98.243589	102.948708	96.066605	99.261574	97.220853	94.133476	98.778463	97.958522
2014-01-14	99.891953	102.443840	97.613741	100.210208	97.614360	96.286326	99.897570	99.010821
2014-01-15	99.472321	102.890171	97.390851	101.191181	97.786519	98.923570	100.077681	99.618334

In []: `norm.iloc[:, [0,1, 2, 3, 4, 5, 7]].plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()`

```
In [63]: norm.loc[:, ["AMZN", "EWI"]].plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```



```
In [64]: norm.iloc[:, -2].plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```



```
In [65]: weights_EWI = stocks.copy()
```

```
In [66]: weights_EWI.iloc[:, :] = (1/6)
```

```
In [69]: weights_EWI.head(10)
```

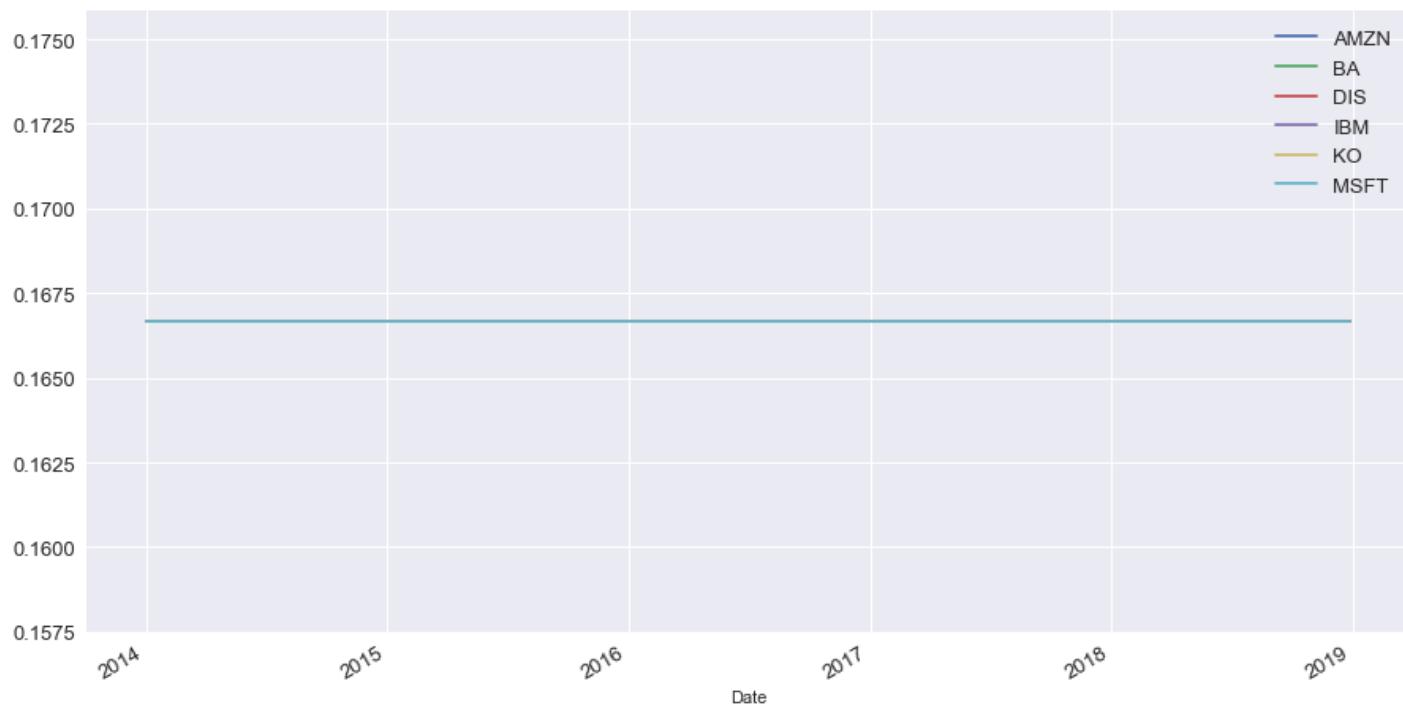
Out[69]:

AMZN BA DIS IBM KO MSFT

Date	AMZN	BA	DIS	IBM	KO	MSFT
2014-01-02	0.166667	0.166667	0.166667	0.166667	0.166667	0.166667
2014-01-03	0.166667	0.166667	0.166667	0.166667	0.166667	0.166667
2014-01-06	0.166667	0.166667	0.166667	0.166667	0.166667	0.166667
2014-01-07	0.166667	0.166667	0.166667	0.166667	0.166667	0.166667
2014-01-08	0.166667	0.166667	0.166667	0.166667	0.166667	0.166667
2014-01-09	0.166667	0.166667	0.166667	0.166667	0.166667	0.166667
2014-01-10	0.166667	0.166667	0.166667	0.166667	0.166667	0.166667
2014-01-13	0.166667	0.166667	0.166667	0.166667	0.166667	0.166667
2014-01-14	0.166667	0.166667	0.166667	0.166667	0.166667	0.166667
2014-01-15	0.166667	0.166667	0.166667	0.166667	0.166667	0.166667

In [70]:

```
weights_EWI.plot(figsize = (15,8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```



Value Weighted Index

Value Weighted Index is Indexes that are determined by Marketcap. The stocks with the biggest marketcap or economic importance are given a bigger weight.

Marketcap = Stock Price * Shares Outstanding

An “easy” Example (1)

Timestamp	Stock A	Stock B	Sum
0		5	10
1		8	16

Timestamp	Stock A	Stock B
0		10
1		4

Timestamp	Stock A	Stock B	Sum	Index
0	Market	50	40	90 100.00
1	Cap	80	32	112 124.44

$$100 * \frac{112}{90} = 124.44$$

If additional shares are added to the market.

Below is the INCORRECT WAY

An “easy” Example (2)

Timestamp	Stock A	Stock B	Sum
0		5	10
1		8	16

Timestamp	Stock A	Stock B
0		10
1		4

Timestamp	Stock A	Stock B	Sum	Index
0	Market	50	40	90 100.00
1	Cap	160	32	192 213.33

$$100 * \frac{192}{90} = 213.33$$

The correct way.

A more general approach (2)

Timestamp	Stock A	Stock B	Sum
0	Market	50 0.56	40 0.44
1	Cap weights	160	32

Timestamp	Stock A	Stock B	Sum
0	5	10	15
1	8	8	16

Timestamp	Stock A	Stock B	Weighted Av.	Index
0	na	na	na	100.00
1	0.6	-0.2	0.24	124.44

Advantages of Value Weighted Index

Best reflects the aggregate market/ average investor portfolio

Weights of the constituents are based on economic importance.

Does not require rebalancing.

Disadvantages of Value Weighted Indexes

Overconcentrated in a few stocks with a high Marketcap.

The highest value stocks may be the most overvalued. VWI perform poorly.

High data needs. (Shares outstanding/ free float etc.)

Creating Value-weighted Index (Part 1)

In [72]:

```
stocks.head()
```

Out[72]:

AMZN **BA** **DIS** **IBM** **KO** **MSFT**

Date	Open	High	Low	Close	Volume	Turnover
2014-01-02	397.970001	136.669998	76.269997	177.370941	40.660000	37.160000
2014-01-03	396.440002	137.619995	76.110001	178.432129	40.459999	36.910000
2014-01-06	393.630005	138.410004	75.820000	177.820267	40.270000	36.130001
2014-01-07	398.029999	140.509995	76.339996	181.367111	40.389999	36.410000
2014-01-08	401.920013	140.820007	75.220001	179.703629	39.939999	35.759998

In [73]:

norm_head()

Out[73]:

AMZN BA DIS IBM KO MSET PWI FWL

AMZN BA DIS IBM KO MSFT PWI EWI

Date

2014-01-02	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
2014-01-03	99.615549	100.695103	99.790224	100.598287	99.508114	99.327234	99.985127	99.922419
2014-01-06	98.909466	101.273144	99.409995	100.253325	99.040828	97.228206	99.535775	99.349274
2014-01-07	100.015076	102.809685	100.091779	102.253002	99.335956	97.981701	100.802004	100.407085
2014-01-08	100.992540	103.036518	98.623318	101.315146	98.229215	96.232504	100.838552	99.723360

In [75]:

```
listings = pd.read_csv(r"C:\Users\alonz\Documents\Finance_Data_Exc\Finance_Data_Exc\PART_1.csv")
```

In [76]:

```
listings.head()
```

Out[76]:

	Symbol	Exchange	Name	Last_Price	Market_Cap	ADR_TSO	IPO_Year	Sector	Industry
0	A	NYSE	Agilent Technologies, Inc.	81.68	2.593470e+10	NaN	1999.0	Capital Goods	Biotechnology: Laboratory Analytical Instruments
1	AA	NYSE	Alcoa Corporation	29.15	5.407810e+09	NaN	2016.0	Basic Industries	Aluminum
2	AABA	NASDAQ	Altaba Inc.	75.39	4.278113e+10	NaN	NaN	Technology	EDP Services
3	AAC	NYSE	AAC Holdings, Inc.	2.16	5.314109e+07	NaN	2014.0	Health Care	Medical Specialities
4	AAL	NASDAQ	American Airlines Group, Inc.	34.02	1.527687e+10	NaN	NaN	Transportation	Air Freight/Delivery Services

In [77]:

```
listings.tail()
```

Out[77]:

	Symbol	Exchange	Name	Last_Price	Market_Cap	ADR_TSO	IPO_Year	Sector	Industry
6847	ZUMZ	NASDAQ	Zumiez Inc.	26.72	6.819191e+08	NaN	2005.0	Consumer Services	Clothing/Shoe/Accessories
6848	ZUO	NYSE	Zuora, Inc.	19.79	2.147266e+09	NaN	2018.0	Technology	Computer Software: Prepackaged Software
6849	ZYME	NYSE	Zymeworks Inc.	15.74	5.040782e+08	NaN	2017.0	Health Care	Major Pharmaceuticals
6850	ZYNE	NASDAQ	Zynerba Pharmaceuticals, Inc.	7.85	1.653995e+08	NaN	2015.0	Health Care	Major Pharmaceuticals
6851	ZYXI	NASDAQ	Zynex, Inc.	5.02	1.618349e+08	NaN	NaN	Health Care	Biotechnology: Electromedical/Electrotherapy

In [80]:

```
listings.tail()
```

Out[80]:

	Symbol	Exchange	Name	Last_Price	Market_Cap	ADR_TSO	IPO_Year	Sector	Industry
6847	ZUMZ	NASDAQ	Zumiez Inc.	26.72	6.819191e+08	NaN	2005.0	Consumer Services	Clothing/Shoe/Accessories
6848	ZUO	NYSE	Zuora, Inc.	19.79	2.147266e+09	NaN	2018.0	Technology	Computer Software/Prepackaged Software
6849	ZYME	NYSE	Zymeworks Inc.	15.74	5.040782e+08	NaN	2017.0	Health Care	Major Pharmaceuticals
6850	ZYNE	NASDAQ	Zynerba Pharmaceuticals, Inc.	7.85	1.653995e+08	NaN	2015.0	Health Care	Major Pharmaceuticals
6851	ZYXI	NASDAQ	Zynex, Inc.	5.02	1.618349e+08	NaN	NaN	Health Care	Biotechnology/Electromedical/Electrotherapy

In [81]:

```
listings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6852 entries, 0 to 6851
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Symbol      6852 non-null   object 
 1   Exchange    6852 non-null   object 
 2   Name        6852 non-null   object 
 3   Last_Price  6745 non-null   float64
 4   Market_Cap 5954 non-null   float64
 5   ADR_TSO    140 non-null    float64
 6   IPO_Year   3105 non-null   float64
 7   Sector      5309 non-null   object 
 8   Industry    5309 non-null   object 
dtypes: float64(4), object(5)
memory usage: 481.9+ KB
```

In [83]:

```
listings.set_index("Symbol", inplace = True)
```

In [84]:

```
listings.head()
```

Out[84]:

	Exchange	Name	Last_Price	Market_Cap	ADR_TSO	IPO_Year	Sector	Industry
	Symbol							
A	NYSE	Agilent Technologies, Inc.	81.68	2.593470e+10	NaN	1999.0	Capital Goods	Biotechnology/Laboratory Analytical Instruments
AA	NYSE	Alcoa Corporation	29.15	5.407810e+09	NaN	2016.0	Basic Industries	Aluminum
AABA	NASDAQ	Altaba Inc.	75.39	4.278113e+10	NaN	NaN	Technology	EDP Services
AAC	NYSE	AAC Holdings, Inc.	2.16	5.314109e+07	NaN	2014.0	Health Care	Medical Specialities

Exchange	Name	Last_Price	Market_Cap	ADR_TSO	IPO_Year	Sector	Industry
----------	------	------------	------------	---------	----------	--------	----------

Symbol

AAL	NASDAQ	American Airlines Group, Inc.	34.02	1.527687e+10	NaN	NaN	Transportation	Air Freight/Delivery Services
------------	--------	-------------------------------	-------	--------------	-----	-----	----------------	-------------------------------

```
In [85]: ticker = ["AMZN", "BA", "DIS", "IBM", "KO", "MSFT"]
```

```
In [86]: listings = listings.loc[ticker, ["Last_Price", "Market_Cap"]]
```

```
In [87]: listings
```

```
Out[87]:      Last_Price  Market_Cap
```

Symbol

AMZN	1847.33	9.074138e+11
BA	364.94	2.060020e+11
DIS	117.16	2.106093e+11
IBM	143.02	1.272687e+11
KO	46.64	1.993935e+11
MSFT	120.19	9.221233e+11

Find shares outstanding by dividing marketcap by the last price.

```
In [88]: listings.Market_Cap.div(listings.Last_Price)
```

```
Out[88]:      Symbol
AMZN      4.912029e+08
BA        5.644820e+08
DIS       1.797621e+09
IBM       8.898663e+08
KO        4.275161e+09
MSFT      7.672213e+09
dtype: float64
```

```
In [89]: listings["Shares"] = listings.Market_Cap.div(listings.Last_Price)
```

```
In [90]: listings
```

```
Out[90]:      Last_Price  Market_Cap      Shares
```

Symbol

AMZN	1847.33	9.074138e+11	4.912029e+08
BA	364.94	2.060020e+11	5.644820e+08

	Last_Price	Market_Cap	Shares
--	------------	------------	--------

Symbol

DIS	117.16	2.106093e+11	1.797621e+09
IBM	143.02	1.272687e+11	8.898663e+08
KO	46.64	1.993935e+11	4.275161e+09
MSFT	120.19	9.221233e+11	7.672213e+09

In [91]:

stocks.head()

Out[91]:

	AMZN	BA	DIS	IBM	KO	MSFT
--	------	----	-----	-----	----	------

Date

2014-01-02	397.970001	136.669998	76.269997	177.370941	40.660000	37.160000
2014-01-03	396.440002	137.619995	76.110001	178.432129	40.459999	36.910000
2014-01-06	393.630005	138.410004	75.820000	177.820267	40.270000	36.130001
2014-01-07	398.029999	140.509995	76.339996	181.367111	40.389999	36.410000
2014-01-08	401.920013	140.820007	75.220001	179.703629	39.939999	35.759998

In [92]:

mcap = stocks.mul(listings.Shares, axis = "columns")

In [93]:

mcap.head()

Out[93]:

	AMZN	BA	DIS	IBM	KO	MSFT
--	------	----	-----	-----	----	------

Date

2014-01-02	1.954840e+11	7.714775e+10	1.371046e+11	1.578364e+11	1.738281e+11	2.850995e+11
2014-01-03	1.947325e+11	7.768400e+10	1.368170e+11	1.587807e+11	1.729730e+11	2.831814e+11
2014-01-06	1.933522e+11	7.812995e+10	1.362957e+11	1.582363e+11	1.721607e+11	2.771971e+11
2014-01-07	1.955135e+11	7.931536e+10	1.372304e+11	1.613925e+11	1.726738e+11	2.793453e+11
2014-01-08	1.974243e+11	7.949035e+10	1.352171e+11	1.599122e+11	1.707499e+11	2.743583e+11

In [94]:

mcap.tail()

Out[94]:

	AMZN	BA	DIS	IBM	KO	MSFT
--	------	----	-----	-----	----	------

Date

2018-12-21	6.766074e+11	1.719130e+11	1.873481e+11	9.438027e+10	2.033694e+11	7.536416e+11
2018-12-24	6.601570e+11	1.660480e+11	1.803913e+11	9.151330e+10	1.964864e+11	7.221854e+11
2018-12-26	7.225103e+11	1.772078e+11	1.902423e+11	9.476310e+10	2.006761e+11	7.715178e+11
2018-12-27	7.179618e+11	1.790198e+11	1.914826e+11	9.679635e+10	2.031984e+11	7.762746e+11
2018-12-28	7.260077e+11	1.785908e+11	1.928848e+11	9.615830e+10	2.017876e+11	7.702135e+11

```
In [95]: mcap.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1257 entries, 2014-01-02 to 2018-12-28
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
--- 
 0   AMZN    1257 non-null   float64
 1   BA      1257 non-null   float64
 2   DIS     1257 non-null   float64
 3   IBM     1257 non-null   float64
 4   KO      1257 non-null   float64
 5   MSFT    1257 non-null   float64
dtypes: float64(6)
memory usage: 101.0 KB
```

```
In [96]: mcap.plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```



```
In [97]: stocks.plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
```

```
Out[97]: <matplotlib.legend.Legend at 0x142f967c2b0>
```



Creating a Value-Weighted Index (Part 2)

In [98]:

```
mcap.head()
```

Out[98]:

	AMZN	BA	DIS	IBM	KO	MSFT
Date						
2014-01-02	1.954840e+11	7.714775e+10	1.371046e+11	1.578364e+11	1.738281e+11	2.850995e+11
2014-01-03	1.947325e+11	7.768400e+10	1.368170e+11	1.587807e+11	1.729730e+11	2.831814e+11
2014-01-06	1.933522e+11	7.812995e+10	1.362957e+11	1.582363e+11	1.721607e+11	2.771971e+11
2014-01-07	1.955135e+11	7.931536e+10	1.372304e+11	1.613925e+11	1.726738e+11	2.793453e+11
2014-01-08	1.974243e+11	7.949035e+10	1.352171e+11	1.599122e+11	1.707499e+11	2.743583e+11

In [99]:

```
mcap.sum(axis = 1)
```

Out[99]:

Date	
2014-01-02	1.026500e+12
2014-01-03	1.024169e+12
2014-01-06	1.015372e+12
2014-01-07	1.025471e+12
2014-01-08	1.017152e+12
...	
2018-12-21	2.087260e+12
2018-12-24	2.016781e+12
2018-12-26	2.156917e+12
2018-12-27	2.164734e+12
2018-12-28	2.165643e+12
Length: 1257, dtype: float64	

In [101...]:

```
mcap.div(mcap.sum(axis = 1), axis = "index").head(10)
```

Out[101...]:

	AMZN	BA	DIS	IBM	KO	MSFT
--	-------------	-----------	------------	------------	-----------	-------------

Date	AMZN	BA	DIS	IBM	KO	MSFT
------	------	----	-----	-----	----	------

Date

2014-01-02	0.190437	0.075156	0.133565	0.153762	0.169340	0.277739
2014-01-03	0.190137	0.075851	0.133588	0.155034	0.168891	0.276499
2014-01-06	0.190425	0.076947	0.134232	0.155841	0.169554	0.273001
2014-01-07	0.190657	0.077345	0.133822	0.157384	0.168385	0.272407
2014-01-08	0.194095	0.078150	0.132937	0.157216	0.167871	0.269732
2014-01-09	0.194314	0.079145	0.132822	0.157255	0.167556	0.268908
2014-01-10	0.191815	0.078658	0.133083	0.156440	0.168474	0.271529
2014-01-13	0.192585	0.079643	0.132078	0.157107	0.169467	0.269120
2014-01-14	0.193244	0.078212	0.132442	0.156525	0.167918	0.271659
2014-01-15	0.190811	0.077891	0.131027	0.156725	0.166797	0.276749

In [103...]

```
weights_vwi = mcap.div(mcap.sum(axis = 1), axis = "index")
```

In [105...]

```
weights_vwi.tail()
```

Out[105...]

AMZN	BA	DIS	IBM	KO	MSFT
------	----	-----	-----	----	------

Date

2018-12-21	0.324161	0.082363	0.089758	0.045217	0.097434	0.361067
2018-12-24	0.327332	0.082333	0.089445	0.045376	0.097426	0.358088
2018-12-26	0.334974	0.082158	0.088201	0.043935	0.093038	0.357695
2018-12-27	0.331663	0.082698	0.088456	0.044715	0.093868	0.358601
2018-12-28	0.335239	0.082465	0.089066	0.044402	0.093177	0.355651

In [106...]

```
weights_vwi.plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```



```
In [108...]: ret = stocks.pct_change().dropna()
ret.head(10)
```

```
Out[108...]:
```

	AMZN	BA	DIS	IBM	KO	MSFT
Date						
2014-01-03	-0.003845	0.006951	-0.002098	0.005983	-0.004919	-0.006728
2014-01-06	-0.007088	0.005741	-0.003810	-0.003429	-0.004696	-0.021132
2014-01-07	0.011178	0.015172	0.006858	0.019946	0.002980	0.007750
2014-01-08	0.009773	0.002206	-0.014671	-0.009172	-0.011141	-0.017852
2014-01-09	-0.002264	0.009303	-0.004254	-0.003139	-0.005258	-0.006432
2014-01-10	-0.008354	-0.001618	0.006542	-0.000640	0.010068	0.014354
2014-01-13	-0.016798	-0.008457	-0.028120	-0.016555	-0.014951	-0.029412
2014-01-14	0.016778	-0.004904	0.016105	0.009557	0.004048	0.022870
2014-01-15	-0.004201	0.004357	-0.002283	0.009789	0.001764	0.027390
2014-01-16	-0.000177	-0.002916	-0.000942	0.005433	-0.001258	0.003536

Shifts need to occur to take into account bank holidays.

```
In [122...]: # Preliminary step.
weights_vwi.shift()
```

```
Out[122...]:
```

	AMZN	BA	DIS	IBM	KO	MSFT
Date						
2014-01-02	NaN	NaN	NaN	NaN	NaN	NaN
2014-01-03	0.190437	0.075156	0.133565	0.153762	0.169340	0.277739
2014-01-06	0.190137	0.075851	0.133588	0.155034	0.168891	0.276499

AMZN BA DIS IBM KO MSFT

Date

2014-01-07	0.190425	0.076947	0.134232	0.155841	0.169554	0.273001
2014-01-08	0.190657	0.077345	0.133822	0.157384	0.168385	0.272407
...
2018-12-21	0.331466	0.081629	0.088851	0.044415	0.093884	0.359756
2018-12-24	0.324161	0.082363	0.089758	0.045217	0.097434	0.361067
2018-12-26	0.327332	0.082333	0.089445	0.045376	0.097426	0.358088
2018-12-27	0.334974	0.082158	0.088201	0.043935	0.093038	0.357695
2018-12-28	0.331663	0.082698	0.088456	0.044715	0.093868	0.358601

1257 rows × 6 columns

In [110...]

```
# Real step.  
weights_vwi.shift().dropna().head(10)
```

Out[110...]

AMZN BA DIS IBM KO MSFT

Date

2014-01-03	0.190437	0.075156	0.133565	0.153762	0.169340	0.277739
2014-01-06	0.190137	0.075851	0.133588	0.155034	0.168891	0.276499
2014-01-07	0.190425	0.076947	0.134232	0.155841	0.169554	0.273001
2014-01-08	0.190657	0.077345	0.133822	0.157384	0.168385	0.272407
2014-01-09	0.194095	0.078150	0.132937	0.157216	0.167871	0.269732
2014-01-10	0.194314	0.079145	0.132822	0.157255	0.167556	0.268908
2014-01-13	0.191815	0.078658	0.133083	0.156440	0.168474	0.271529
2014-01-14	0.192585	0.079643	0.132078	0.157107	0.169467	0.269120
2014-01-15	0.193244	0.078212	0.132442	0.156525	0.167918	0.271659
2014-01-16	0.190811	0.077891	0.131027	0.156725	0.166797	0.276749

In [111...]

```
ret.mul(weights_vwi.shift().dropna()).sum(axis = 1)
```

Out[111...]

Date

2014-01-03	-0.002271
2014-01-06	-0.008589
2014-01-07	0.009946
2014-01-08	-0.008112
2014-01-09	-0.003389
...	
2018-12-21	-0.035827
2018-12-24	-0.033766
2018-12-26	0.069485
2018-12-27	0.003624
2018-12-28	0.000420

Length: 1256, dtype: float64

In [114...]

```
norm["VWI"] = 100
norm.head()
```

Out[114...]

	AMZN	BA	DIS	IBM	KO	MSFT	PWI	EWI	VWI
Date									
2014-01-02	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100
2014-01-03	99.615549	100.695103	99.790224	100.598287	99.508114	99.327234	99.985127	99.922419	100
2014-01-06	98.909466	101.273144	99.409995	100.253325	99.040828	97.228206	99.535775	99.349274	100
2014-01-07	100.015076	102.809685	100.091779	102.253002	99.335956	97.981701	100.802004	100.407085	100
2014-01-08	100.992540	103.036518	98.623318	101.315146	98.229215	96.232504	100.838552	99.723360	100

In [116...]

```
ret.mul(weights_vwi.shift().dropna().sum(axis = 1).add(1).cumprod().mul(100)
```

Out[116...]

```
Date
2014-01-03    99.772853
2014-01-06    98.915891
2014-01-07    99.899709
2014-01-08    99.089325
2014-01-09    98.753515
...
2018-12-21    203.337475
2018-12-24    196.471598
2018-12-26    210.123411
2018-12-27    210.884852
2018-12-28    210.973420
Length: 1256, dtype: float64
```

Normalized prices of all constituents.

In [120...]

```
norm.iloc[1:, -1] = ret.mul(weights_vwi.shift().dropna().sum(axis = 1).add(1).cumprod().mul(100))
```

In [123...]

```
norm.head(10)
```

Out[123...]

	AMZN	BA	DIS	IBM	KO	MSFT	PWI	EWI	VWI
Date									
2014-01-02	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
2014-01-03	99.615549	100.695103	99.790224	100.598287	99.508114	99.327234	99.985127	99.922419	99.772853
2014-01-06	98.909466	101.273144	99.409995	100.253325	99.040828	97.228206	99.535775	99.349274	98.915891
2014-01-07	100.015076	102.809685	100.091779	102.253002	99.335956	97.981701	100.802004	100.407085	99.899709
2014-01-08	100.992540	103.036518	98.623318	101.315146	98.229215	96.232504	100.838552	99.723360	99.089325

Date

2014-01-02	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
2014-01-03	99.615549	100.695103	99.790224	100.598287	99.508114	99.327234	99.985127	99.922419	99.772853
2014-01-06	98.909466	101.273144	99.409995	100.253325	99.040828	97.228206	99.535775	99.349274	98.915891
2014-01-07	100.015076	102.809685	100.091779	102.253002	99.335956	97.981701	100.802004	100.407085	99.899709
2014-01-08	100.992540	103.036518	98.623318	101.315146	98.229215	96.232504	100.838552	99.723360	99.089325

	AMZN	BA	DIS	IBM	KO	MSFT	PWI	EWI	VWI
--	------	----	-----	-----	----	------	-----	-----	-----

Date

2014-01-09	100.763879	103.995029	98.203756	100.997137	97.712739	95.613560	100.731860	99.523180	98.753515
2014-01-10	99.922105	103.826733	98.846208	100.932461	98.696511	96.986009	100.466911	99.860755	99.204202
2014-01-13	98.243589	102.948708	96.066605	99.261574	97.220853	94.133476	98.778463	97.958522	97.148235
2014-01-14	99.891953	102.443840	97.613741	100.210208	97.614360	96.286326	99.897570	99.010821	98.441276
2014-01-15	99.472321	102.890171	97.390851	101.191181	97.786519	98.923570	100.077681	99.618334	99.277592

In [124...]

```
norm.iloc[:, [0, 1, 2, 3, 4, 5, 8]].plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```



Comparison of Weighting Methods

In [126...]

```
norm.head()
```

Out[126...]

	AMZN	BA	DIS	IBM	KO	MSFT	PWI	EWI	VWI
--	------	----	-----	-----	----	------	-----	-----	-----

Date

2014-01-02	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
2014-01-03	99.615549	100.695103	99.790224	100.598287	99.508114	99.327234	99.985127	99.922419	99.772853
2014-01-06	98.909466	101.273144	99.409995	100.253325	99.040828	97.228206	99.535775	99.349274	98.915891

AMZN

BA

DIS

IBM

KO

MSFT

PWI

EWI

VWI

Date

2014-01-07	100.015076	102.809685	100.091779	102.253002	99.335956	97.981701	100.802004	100.407085	99.899709
2014-01-08	100.992540	103.036518	98.623318	101.315146	98.229215	96.232504	100.838552	99.723360	99.089325

Price Weighted Index, Equal Weighted Index, and Market Value Weighted Index

In [128...]

```
norm.iloc[:, -3: ].plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```



Comparison of Which Stock Did Better

In [131...]

```
norm.iloc[:, :-3].plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```



In [132]:

```
weights_PWI.plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```

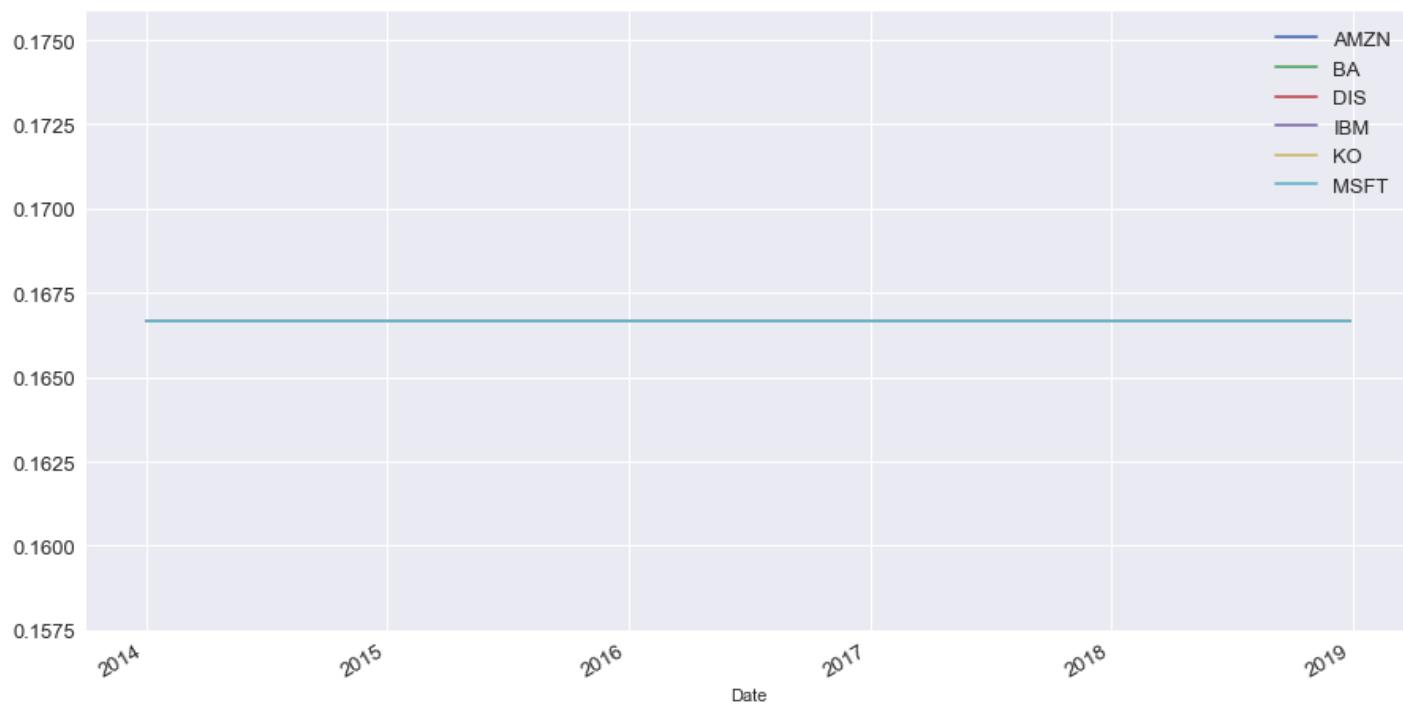


In [133]:

```
weights_vwi.plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```



```
In [134...]: weights_EWI.plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```



In [135...]: norm corr()

Out[135...]

	AMZN	BA	DIS	IBM	KO	MSFT	PWI	EWI	VWI
AMZN	1.000000	0.939706	0.534580	-0.539324	0.659279	0.982694	0.998044	0.985973	0.994244
BA	0.939706	1.000000	0.470091	-0.440646	0.611522	0.956008	0.957589	0.957193	0.959238
DIS	0.534580	0.470091	1.000000	-0.526799	0.447823	0.537984	0.535813	0.590452	0.556982
IBM	-0.539324	-0.440646	-0.526799	1.000000	-0.592244	-0.536878	-0.513335	-0.514637	-0.523827
KO	0.659279	0.611522	0.447823	-0.592244	1.000000	0.713287	0.656528	0.697296	0.688155

	AMZN	BA	DIS	IBM	KO	MSFT	PWI	EWI	VWI
MSFT	0.982694	0.956008	0.537984	-0.536878	0.713287	1.000000	0.987171	0.988939	0.994964
PWI	0.998044	0.957589	0.535813	-0.513335	0.656528	0.987171	1.000000	0.991183	0.997618
EWI	0.985973	0.957193	0.590452	-0.514637	0.697296	0.988939	0.991183	1.000000	0.995234
VWI	0.994244	0.959238	0.556982	-0.523827	0.688155	0.994964	0.997618	0.995234	1.000000

In [136...]

```
summary = norm.pct_change().dropna().agg(["mean", "std"]).T
```

In [137...]

```
summary.head()
```

Out[137...]

	mean	std
AMZN	0.001235	0.019527
BA	0.000777	0.014744
DIS	0.000343	0.011894
IBM	-0.000315	0.012609
KO	0.000157	0.008685

In [138...]

```
summary.columns = ["Return", "Risk"]
```

In [140...]

```
summary["Return"] = summary["Return"] * 252
summary["Risk"] = summary["Risk"] * np.sqrt(252)
```

In [141...]

```
summary
```

Out[141...]

	Return	Risk
AMZN	78.405211	0.309982
BA	49.358927	0.234048
DIS	21.766638	0.188818
IBM	-19.977262	0.200160
KO	9.945030	0.137876
MSFT	57.015781	0.231876
PWI	51.950710	0.214530
EWI	32.752387	0.151747
VWI	41.596074	0.174625

Annualized Return and Risk

Shows Single Stock Portfolios vs. The Indexes

In [143...]

```
summary.plot(kind = "scatter", x = "Risk", y = "Return", figsize = (15, 8), s = 50, fontsi
```

```

for i in summary.index:
    plt.annotate(i, xy = (summary.loc[i, "Risk"]+0.002, summary.loc[i, "Return"]+0.002), s
    plt.xlabel("ann. Risk(std)", fontsize = 15)
    plt.title("Risk/ Return", fontsize = 20)

```

Out[143... Text(0.5, 1.0, 'Risk/ Return')



Price Index vs. Performance/ Total Return Index

To see the total returns it must be done this way.

Simply adding the adjusted into the mix of last section data will not give the proper values.

In [145... import pandas as pd

In [146... stocks = pd.read_csv(r"C:\Users\alonz\index_stocks.csv", header = [0,1], index_col = [0],

In [147... stocks.head()

Out[147...

Date	Adj Close									
	AMZN	BA	DIS	IBM	KO	MSFT	AMZN	BA	DIS	
2014-01-02	397.970001	116.807945	70.192520	125.096390	31.200077	31.798315	397.970001	136.669998	76.269997	177.3
2014-01-03	396.440002	117.619896	70.045273	125.844833	31.046608	31.584404	396.440002	137.619995	76.110001	178.4
2014-01-06	393.630005	118.295044	69.778374	125.413277	30.900801	30.916939	393.630005	138.410004	75.820000	177.8

Adj Close

	AMZN	BA	DIS	IBM	KO	MSFT	AMZN	BA	DIS
--	------	----	-----	-----	----	------	------	----	-----

Date

2014-01-07	398.029999	120.089859	70.256950	127.914818	30.992886	31.156536	398.029999	140.509995	76.339996	181.3
2014-01-08	401.920013	120.354843	69.226196	126.741592	30.647587	30.600323	401.920013	140.820007	75.220001	179.7

5 rows × 36 columns

```
In [148]: adj_close = stocks["Adj Close"].copy()
```

```
In [149]: total_return = adj_close.pct_change().dropna()
```

```
In [150]: total_return.head()
```

```
Out[150]:
```

	AMZN	BA	DIS	IBM	KO	MSFT
--	------	----	-----	-----	----	------

Date

2014-01-03	-0.003845	0.006951	-0.002098	0.005983	-0.004919	-0.006727
2014-01-06	-0.007088	0.005740	-0.003810	-0.003429	-0.004696	-0.021133
2014-01-07	0.011178	0.015172	0.006859	0.019946	0.002980	0.007750
2014-01-08	0.009773	0.002207	-0.014671	-0.009172	-0.011141	-0.017852
2014-01-09	-0.002264	0.009303	-0.004254	-0.003139	-0.005258	-0.006432

```
In [151]: close = stocks["Close"].copy()
```

```
In [152]: weights = close.div(close.sum(axis = 1), axis = "index")
```

```
In [153]: weights.head()
```

```
Out[153]:
```

	AMZN	BA	DIS	IBM	KO	MSFT
--	------	----	-----	-----	----	------

Date

2014-01-02	0.459496	0.157799	0.088061	0.204792	0.046946	0.042905
2014-01-03	0.457798	0.158920	0.087890	0.206048	0.046722	0.042623
2014-01-06	0.456605	0.160553	0.087950	0.206269	0.046713	0.041910
2014-01-07	0.455909	0.160942	0.087441	0.207740	0.046263	0.041705
2014-01-08	0.460198	0.161239	0.086127	0.205760	0.045731	0.040945

```
In [154]: norm["PWI_perf"] = 100
norm.head()
```

Out[154...]

AMZN BA DIS IBM KO MSFT PWI EWI VWI

Date

Date	AMZN	BA	DIS	IBM	KO	MSFT	PWI	EWI	VWI
2014-01-02	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
2014-01-03	99.615549	100.695103	99.790224	100.598287	99.508114	99.327234	99.985127	99.922419	99.772853
2014-01-06	98.909466	101.273144	99.409995	100.253325	99.040828	97.228206	99.535775	99.349274	98.915891
2014-01-07	100.015076	102.809685	100.091779	102.253002	99.335956	97.981701	100.802004	100.407085	99.899709
2014-01-08	100.992540	103.036518	98.623318	101.315146	98.229215	96.232504	100.838552	99.723360	99.089325

In [155...]

norm.iloc[1:, -1] = total_return.mul(weights.shift().dropna()).sum(axis = 1).add(1).cumprod()

In [157...]

norm.head()

Out[157...]

AMZN BA DIS IBM KO MSFT PWI EWI VWI

Date

Date	AMZN	BA	DIS	IBM	KO	MSFT	PWI	EWI	VWI
2014-01-02	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
2014-01-03	99.615549	100.695103	99.790224	100.598287	99.508114	99.327234	99.985127	99.922419	99.772853
2014-01-06	98.909466	101.273144	99.409995	100.253325	99.040828	97.228206	99.535775	99.349274	98.915891
2014-01-07	100.015076	102.809685	100.091779	102.253002	99.335956	97.981701	100.802004	100.407085	99.899709
2014-01-08	100.992540	103.036518	98.623318	101.315146	98.229215	96.232504	100.838552	99.723360	99.089325

In [158...]

summary = norm.pct_change().dropna().agg(["mean", "std"]).T

In [159...]

summary["mean"] = summary["mean"] * 252
summary["std"] = summary["std"] * np.sqrt(252)

In [160...]

summary

Out[160...]

mean std

AMZN	0.311132	0.309982
BA	0.195869	0.234048
DIS	0.086376	0.188818
IBM	-0.079275	0.200160
KO	0.039464	0.137876

	mean	std
MSFT	0.226253	0.231876
PWI	0.206154	0.214530
EWI	0.129970	0.151747
VWI	0.165064	0.174625
PWI_perf	0.217541	0.214402

Dividends reinvested helps performance.

In [162]:

```
norm.iloc[:, [6, 9]].plot(figsize = (15, 8), fontsize = 13)
plt.legend(fontsize = 13)
plt.show()
```



In []:

In []:

In []:

What is a portfolio?

A portfolio is a collection of investment held by an investment company financial institution or individual. It can be stock bonds etc. Need to know the actual asset and the weight of the asset.

Financial Portfolios - Create Analyze and Optimize

Getting Data

In [2]:

```
import pandas as pd
import yfinance as yf
```

In [3]:

```
stock = yf.download(["AMZN", "BA", "DIS", "IBM", "KO", "MSFT"], start = "2014-01-01", end
[*****100%*****] 6 of 6 completed
```

In [4]:

```
stocks = pd.read_csv("index_stocks.csv", header = [0,1], index_col = [0], parse_dates = [0])
```

In [5]:

```
stocks.head()
```

Out[5]:

	Adj Close									
	AMZN	BA	DIS	IBM	KO	MSFT	AMZN	BA	DIS	
Date										
2014-01-02	397.970001	116.807945	70.192520	125.096390	31.200077	31.798315	397.970001	136.669998	76.269997	177.3
2014-01-03	396.440002	117.619896	70.045273	125.844833	31.046608	31.584404	396.440002	137.619995	76.110001	178.4
2014-01-06	393.630005	118.295044	69.778374	125.413277	30.900801	30.916939	393.630005	138.410004	75.820000	177.8
2014-01-07	398.029999	120.089859	70.256950	127.914818	30.992886	31.156536	398.029999	140.509995	76.339996	181.3
2014-01-08	401.920013	120.354843	69.226196	126.741592	30.647587	30.600323	401.920013	140.820007	75.220001	179.7

5 rows x 36 columns

In [6]:

```
stocks.tail()
```

Out[6]:

	Adj Close									
	AMZN	BA	DIS	IBM	KO	MSFT	AMZN	BA	DIS	
Date										
2018-12-21	1377.449951	295.930267	102.961685	89.908890	42.892387	94.791809	1377.449951	304.549988	104.220001	1
2018-12-24	1343.959961	285.834351	99.138412	87.177742	41.440701	90.835304	1343.959961	294.160004	100.349998	1

	AMZN	BA	DIS	IBM	KO	MSFT	AMZN	BA	DIS	Adj Close
--	------	----	-----	-----	----	------	------	----	-----	-----------

Date

2018-12-26	1470.900024	305.044769	104.552246	90.273575	42.324337	97.040253	1470.900024	313.929993	105.830002	1
2018-12-27	1461.640015	308.163971	105.233910	92.210503	42.856316	97.638557	1461.640015	317.140015	106.519997	1
2018-12-28	1478.020020	307.425446	106.004501	91.602676	42.558773	96.876221	1478.020020	316.380005	107.300003	1

5 rows × 36 columns

In [7]:

```
stocks = stocks["Adj Close"].copy()
```

In [8]:

```
stock.head()
```

Out[8]:

	AMZN	BA	DIS	IBM	KO	MSFT	AMZN	BA	DIS	Adj Close
Date										
2014-01-02	397.970001	116.807945	70.192513	125.096367	31.200064	31.798313	397.970001	136.669998	76.269997	177.3
2014-01-03	396.440002	117.619888	70.045273	125.844833	31.046606	31.584404	396.440002	137.619995	76.110001	178.4
2014-01-06	393.630005	118.295097	69.778381	125.413315	30.900805	30.916943	393.630005	138.410004	75.820000	177.8
2014-01-07	398.029999	120.089874	70.256935	127.914787	30.992887	31.156536	398.029999	140.509995	76.339996	181.3
2014-01-08	401.920013	120.354858	69.226189	126.741570	30.647591	30.600327	401.920013	140.820007	75.220001	179.7

5 rows × 36 columns

In [11]:

```
stocks.to_csv("port_stocks.csv")
```

Creating the Equal-Weighted Portfolio

In [35]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use("seaborn")
pd.options.display.float_format = "{:.2f}".format
```

In [36]:

```
stocks = pd.read_csv("port_stocks.csv", parse_dates = ["Date"], index_col = "Date")
```

In [37]:

```
stocks.head()
```

```
Out[37]:
```

	AMZN	BA	DIS	IBM	KO	MSFT
--	------	----	-----	-----	----	------

Date

2014-01-02	397.97	116.81	70.19	125.10	31.20	31.80
2014-01-03	396.44	117.62	70.05	125.84	31.05	31.58
2014-01-06	393.63	118.30	69.78	125.41	30.90	30.92
2014-01-07	398.03	120.09	70.26	127.91	30.99	31.16
2014-01-08	401.92	120.35	69.23	126.74	30.65	30.60

```
In [38]:
```

```
ret = stocks.pct_change().dropna()
```

```
In [39]:
```

```
ret.head()
```

```
Out[39]:
```

	AMZN	BA	DIS	IBM	KO	MSFT
--	------	----	-----	-----	----	------

Date

2014-01-03	-0.00	0.01	-0.00	0.01	-0.00	-0.01
2014-01-06	-0.01	0.01	-0.00	-0.00	-0.00	-0.02
2014-01-07	0.01	0.02	0.01	0.02	0.00	0.01
2014-01-08	0.01	0.00	-0.01	-0.01	-0.01	-0.02
2014-01-09	-0.00	0.01	-0.00	-0.00	-0.01	-0.01

```
In [40]:
```

```
ret.mean(axis = 1)
```

```
Out[40]:
```

```
Date
2014-01-03    -0.00
2014-01-06    -0.01
2014-01-07     0.01
2014-01-08    -0.01
2014-01-09    -0.00
...
2018-12-21    -0.03
2018-12-24    -0.03
2018-12-26     0.06
2018-12-27     0.01
2018-12-28    -0.00
Length: 1256, dtype: float64
```

```
In [41]:
```

```
no_assets = len(stocks.columns)
no_assets
```

```
Out[41]:
```

```
6
```

```
In [42]:
```

```
weights = [1/no_assets for i in range(no_assets)]
weights
```

```
Out[42]:
```

```
[0.1666666666666666,
 0.1666666666666666,
 0.1666666666666666,
 0.1666666666666666,
```

```
0.1666666666666666,  
0.1666666666666666]
```

```
In [44]: ret.mul(weights, axis = "columns").sum(axis = 1)
```

```
Out[44]: Date  
2014-01-03    -0.00  
2014-01-06    -0.01  
2014-01-07     0.01  
2014-01-08    -0.01  
2014-01-09    -0.00  
...  
2018-12-21    -0.03  
2018-12-24    -0.03  
2018-12-26     0.06  
2018-12-27     0.01  
2018-12-28    -0.00  
Length: 1256, dtype: float64
```

```
In [46]: ret.dot(weights)
```

```
Out[46]: Date  
2014-01-03    -0.00  
2014-01-06    -0.01  
2014-01-07     0.01  
2014-01-08    -0.01  
2014-01-09    -0.00  
...  
2018-12-21    -0.03  
2018-12-24    -0.03  
2018-12-26     0.06  
2018-12-27     0.01  
2018-12-28    -0.00  
Length: 1256, dtype: float64
```

.dot() is a matrix modification

```
In [47]: ret["EWP"] = ret.dot(weights)
```

```
In [48]: ret.head()
```

```
Out[48]:      AMZN   BA   DIS   IBM   KO   MSFT   EWP  
              Date  
2014-01-03  -0.00  0.01  -0.00  0.01  -0.00  -0.01  -0.00  
2014-01-06  -0.01  0.01  -0.00  -0.00  -0.00  -0.02  -0.01  
2014-01-07   0.01  0.02  0.01  0.02  0.00   0.01  0.01  
2014-01-08   0.01  0.00  -0.01  -0.01  -0.01  -0.02  -0.01  
2014-01-09  -0.00  0.01  -0.00  -0.00  -0.01  -0.01  -0.00
```

```
In [51]: summary = ret.agg(["mean", "std"]).T
```

```
In [52]: summary
```

```
Out[52]:
```

	mean	std
AMZN	0.00	0.02
BA	0.00	0.01
DIS	0.00	0.01
IBM	-0.00	0.01
KO	0.00	0.01
MSFT	0.00	0.01
EWP	0.00	0.01

```
In [57]:
```

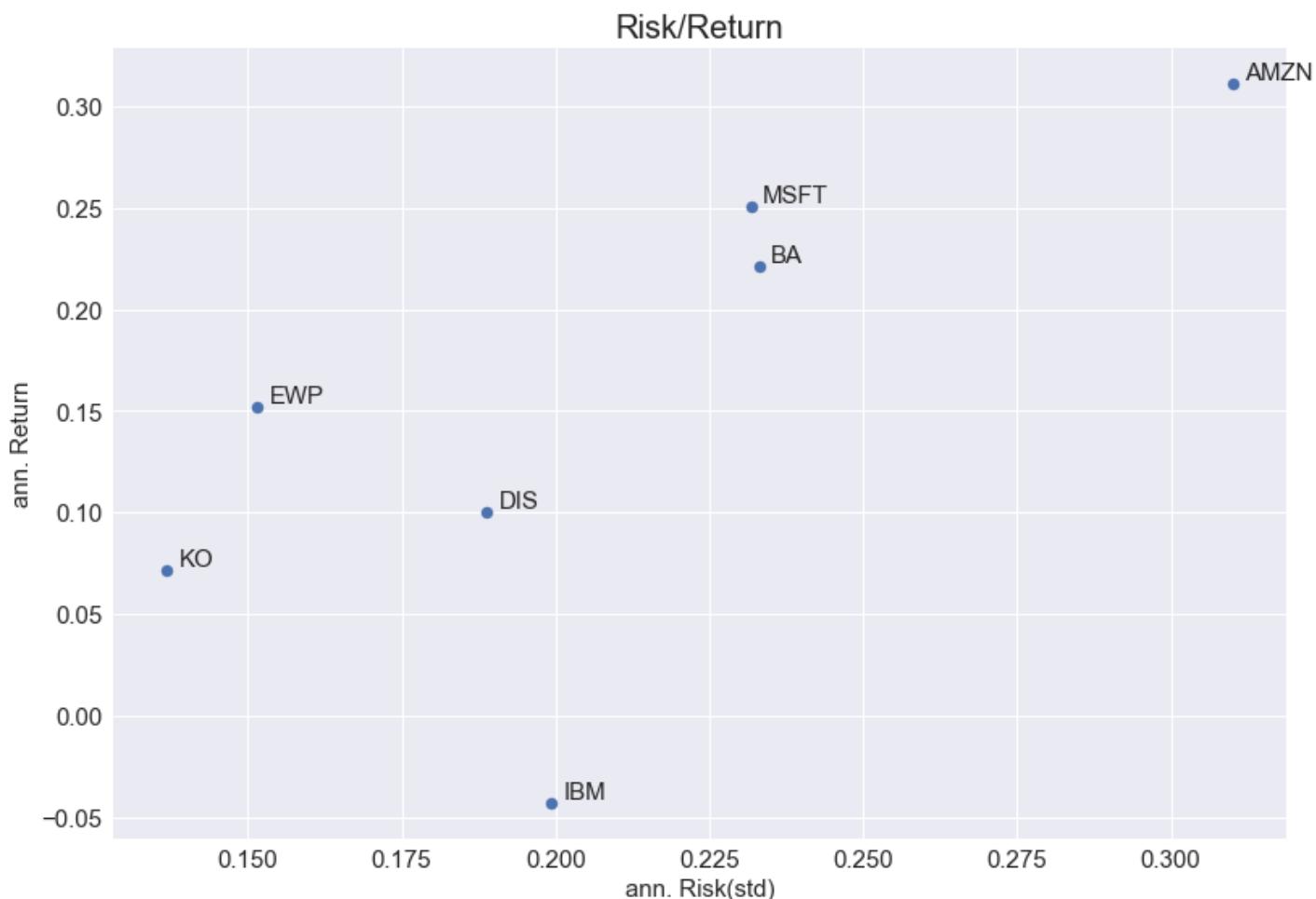
```
summary.columns = ["Return", "Risk"]
```

```
In [58]:
```

```
summary.Return = summary.Return*252
summary.Risk = summary.Risk * np.sqrt(252)
```

```
In [59]:
```

```
summary.plot(kind = "scatter", x = "Risk", y = "Return", figsize = (13, 9), s = 50, fontsize = 15)
for i in summary.index:
    plt.annotate(i, xy = (summary.loc[i, "Risk"]+0.002, summary.loc[i, "Return"]+0.002), s = 100)
plt.xlabel("ann. Risk(std)", fontsize = 15)
plt.ylabel("ann. Return", fontsize = 15)
plt.title("Risk/Return", fontsize = 20)
plt.show()
```



Create Many Random Portfolios

In [61]:

```
def ann_risk_return(returns_df):  
    summary = returns_df.agg([ "mean", "std" ]).T  
    summary.columns = [ "Return", "Risk" ]  
    summary.Return = summary.Return * 252  
    summary.Risk = summary.Risk * np.sqrt(252)  
    return summary
```

In [62]:

```
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
plt.style.use("seaborn")  
pd.options.display.float_format = "{:.2f}".format
```

In [63]:

```
stocks = pd.read_csv("port_stocks.csv", parse_dates = [ "Date" ], index_col = "Date")
```

In [64]:

```
stocks.head()
```

Out[64]:

	AMZN	BA	DIS	IBM	KO	MSFT
Date						
2014-01-02	397.97	116.81	70.19	125.10	31.20	31.80
2014-01-03	396.44	117.62	70.05	125.84	31.05	31.58
2014-01-06	393.63	118.30	69.78	125.41	30.90	30.92
2014-01-07	398.03	120.09	70.26	127.91	30.99	31.16
2014-01-08	401.92	120.35	69.23	126.74	30.65	30.60

In [66]:

```
ret = stocks.pct_change().dropna()  
ret.head()
```

Out[66]:

	AMZN	BA	DIS	IBM	KO	MSFT
Date						
2014-01-03	-0.00	0.01	-0.00	0.01	-0.00	-0.01
2014-01-06	-0.01	0.01	-0.00	-0.00	-0.00	-0.02
2014-01-07	0.01	0.02	0.01	0.02	0.00	0.01
2014-01-08	0.01	0.00	-0.01	-0.01	-0.01	-0.02
2014-01-09	-0.00	0.01	-0.00	-0.00	-0.01	-0.01

In [67]:

```
summary = ann_risk_return(ret)
```

In [68]:

```
summary
```

Out[68]:

	Return	Risk
AMZN	0.31	0.31

	Return	Risk
BA	0.22	0.23
DIS	0.10	0.19
IBM	-0.04	0.20
KO	0.07	0.14
MSFT	0.25	0.23

```
In [69]: noa = len(stocks.columns)
noa
```

```
Out[69]: 6
```

```
In [88]: nop = 100000
nop
```

```
Out[88]: 100000
```

```
In [89]: np.random.random(10* 6).reshape(10, 6)
```

```
Out[89]: array([[0.32534419, 0.4388816 , 0.03031282, 0.09983388, 0.84654819,
       0.10240211],
       [0.8670038 , 0.31243032, 0.67128825, 0.39890191, 0.21139735,
       0.76096397],
       [0.25168434, 0.69033252, 0.16086803, 0.89314458, 0.06728694,
       0.93780909],
       [0.07523895, 0.37748695, 0.39302596, 0.60237579, 0.76957771,
       0.21833182],
       [0.7894114 , 0.73964541, 0.77998638, 0.5072607 , 0.13983513,
       0.12643534],
       [0.87037476, 0.98161572, 0.21658619, 0.52518826, 0.8185745 ,
       0.03708083],
       [0.09800071, 0.12321758, 0.85742333, 0.80331768, 0.9964052 ,
       0.17424874],
       [0.2893227 , 0.09954133, 0.28827166, 0.96571067, 0.36551084,
       0.04794844],
       [0.77320928, 0.38881389, 0.89509051, 0.18048509, 0.70021812,
       0.77266336],
       [0.00374172, 0.65024409, 0.48092876, 0.65856373, 0.29710321,
       0.85409599]])
```

```
In [90]: np.random.random(noa * nop).reshape(nop, noa)
```

```
Out[90]: array([[0.14588602, 0.31781185, 0.83787801, 0.98753293, 0.11136098,
       0.22229965],
       [0.59743774, 0.55433511, 0.71576279, 0.73558453, 0.51053063,
       0.21331361],
       [0.4603237 , 0.10535305, 0.66965932, 0.09257737, 0.79316887,
       0.64802406],
       ...,
       [0.40613342, 0.19414528, 0.63536963, 0.07120736, 0.62890071,
       0.73809635],
       [0.79666497, 0.70576438, 0.28622446, 0.20960745, 0.01402744,
       0.83088061],
       [0.50591476, 0.93727327, 0.98756231, 0.0043833 , 0.45573235,
       0.04880503]])
```

```
In [91]: np.random.seed(123)
matrix = np.random.random(noa * nop).reshape(nop, noa)
```

The percent for the matrix must add up to one

```
In [92]: matrix
```

```
Out[92]: array([[0.69646919, 0.28613933, 0.22685145, 0.55131477, 0.71946897,
 0.42310646],
[0.9807642 , 0.68482974, 0.4809319 , 0.39211752, 0.34317802,
 0.72904971],
[0.43857224, 0.0596779 , 0.39804426, 0.73799541, 0.18249173,
 0.17545176],
...,
[0.51954687, 0.85226532, 0.87403327, 0.10039779, 0.25265273,
 0.01875777],
[0.64206112, 0.41780645, 0.76066273, 0.33994549, 0.1620518 ,
 0.94008933],
[0.23120286, 0.86951896, 0.50415836, 0.32237088, 0.25906289,
 0.90163486]])
```

Given below is the sum of each row.

With the sum of each row there must be a normalization by making everything equal to one. So just divide each row by the sum.

```
In [93]: matrix.sum(axis = 1, keepdims = True)
```

```
Out[93]: array([[2.90335017],
[3.61087108],
[1.99223329],
...,
[2.61765375],
[3.26261692],
[3.08794881]])
```

```
In [94]: weights = matrix / matrix.sum(axis = 1, keepdims = True)
```

```
In [95]: weights.sum(axis = 1, keepdims = True)
```

```
Out[95]: array([[1.],
[1.],
[1.],
...,
[1.],
[1.],
[1.]]))
```

```
In [96]: port_ret = ret.dot(weights.T)
```

```
In [97]: port_ret.head(10)
```

```
Out[97]: 0 1 2 3 4 5 6 7 8 9 ... 99990 99991 99992 99993 99994 99995
```

Date

Date	0	1	2	3	4	5	6	7	8	9	...	99990	99991	99992	99993	99994	99995
2014-01-03	-0.00	-0.00	0.00	-0.00	-0.00	0.00	0.00	-0.00	0.00	-0.00	...	0.00	-0.00	0.00	0.00	-0.00	-
2014-01-06	-0.01	-0.01	-0.01	-0.01	-0.01	-0.00	-0.00	-0.01	-0.01	-0.01	...	-0.00	-0.01	-0.00	-0.01	-0.01	-
2014-01-07	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	...	0.01	0.01	0.01	0.01	0.01	0.01
2014-01-08	-0.01	-0.00	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01	...	-0.01	-0.01	-0.00	-0.00	-0.01	-
2014-01-09	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	-0.00	...	0.00	-0.00	0.00	-0.00	-0.00	-
2014-01-10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	...	0.00	0.00	0.00	0.00	0.00	0.00
2014-01-13	-0.02	-0.02	-0.02	-0.02	-0.02	-0.02	-0.02	-0.02	-0.02	-0.02	...	-0.02	-0.02	-0.01	-0.02	-0.02	-
2014-01-14	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	...	0.01	0.01	0.01	0.01	0.01	0.01
2014-01-15	0.01	0.01	0.00	0.01	0.01	0.01	0.00	0.01	0.01	0.01	...	0.00	0.01	0.01	0.01	0.01	0.01
2014-01-16	0.00	0.00	0.00	0.00	0.00	0.00	-0.00	0.00	0.00	0.00	...	-0.00	0.00	0.00	0.00	0.00	0.00

10 rows × 100000 columns

In [98]: `port_summary = ann_risk_return(port_ret)`

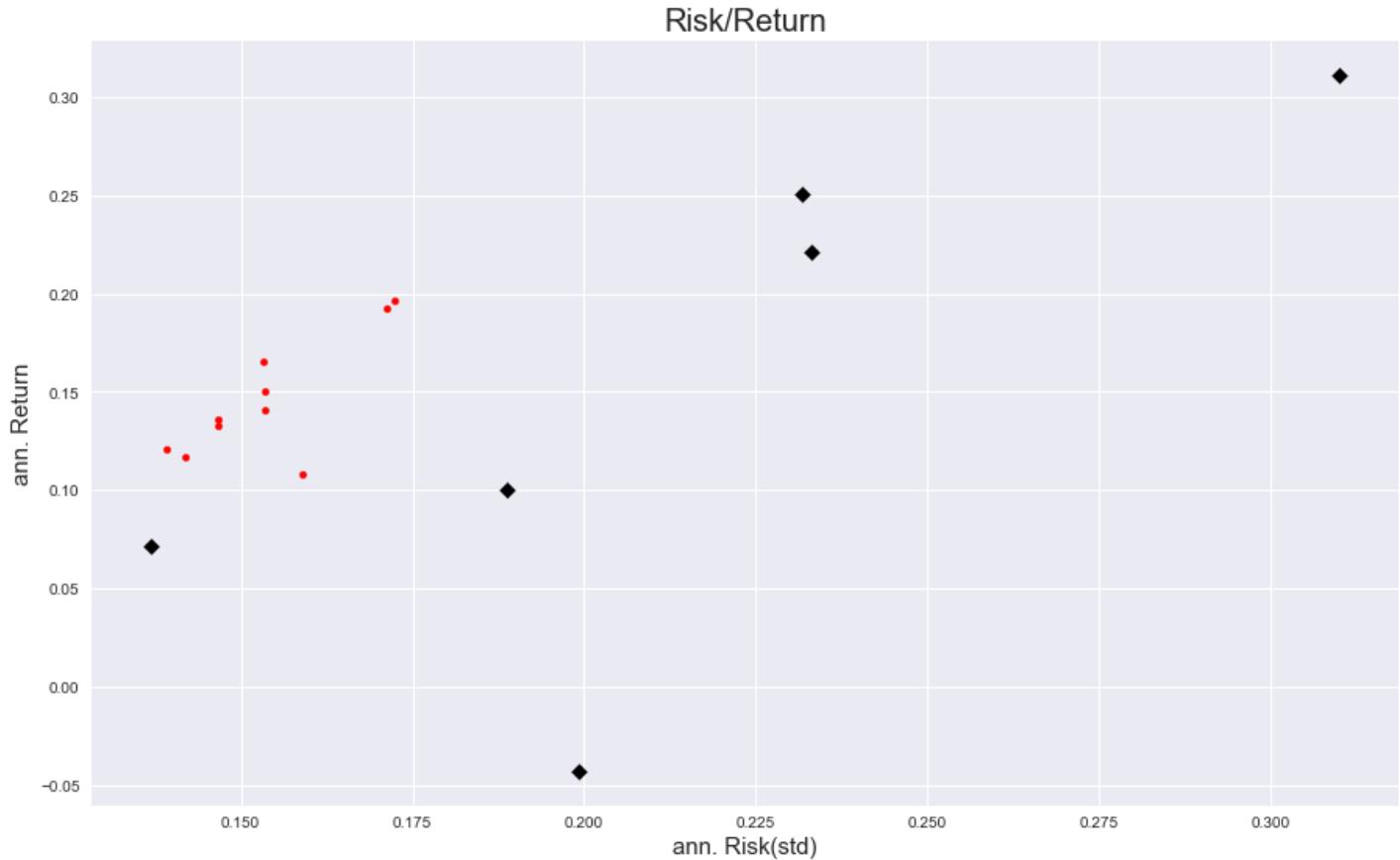
In [99]: `port_summary`

Out[99]:

	Return	Risk
0	0.15	0.15
1	0.19	0.17
2	0.11	0.16
3	0.13	0.15
4	0.20	0.17
...
99995	0.12	0.16
99996	0.18	0.16
99997	0.17	0.16
99998	0.18	0.17
99999	0.18	0.16

100000 rows × 2 columns

```
In [83]: plt.figure(figsize = (15, 9))
plt.scatter(port_summary.loc[:, "Risk"], port_summary.loc[:, "Return"], s = 20, color = "red")
plt.scatter(summary.loc[:, "Risk"], summary.loc[:, "Return"], s = 50, color = "black", marker = "diamond")
plt.xlabel("ann. Risk(std)", fontsize = 15)
plt.ylabel("ann. Return", fontsize = 15)
plt.title("Risk/Return", fontsize = 20)
plt.show()
```

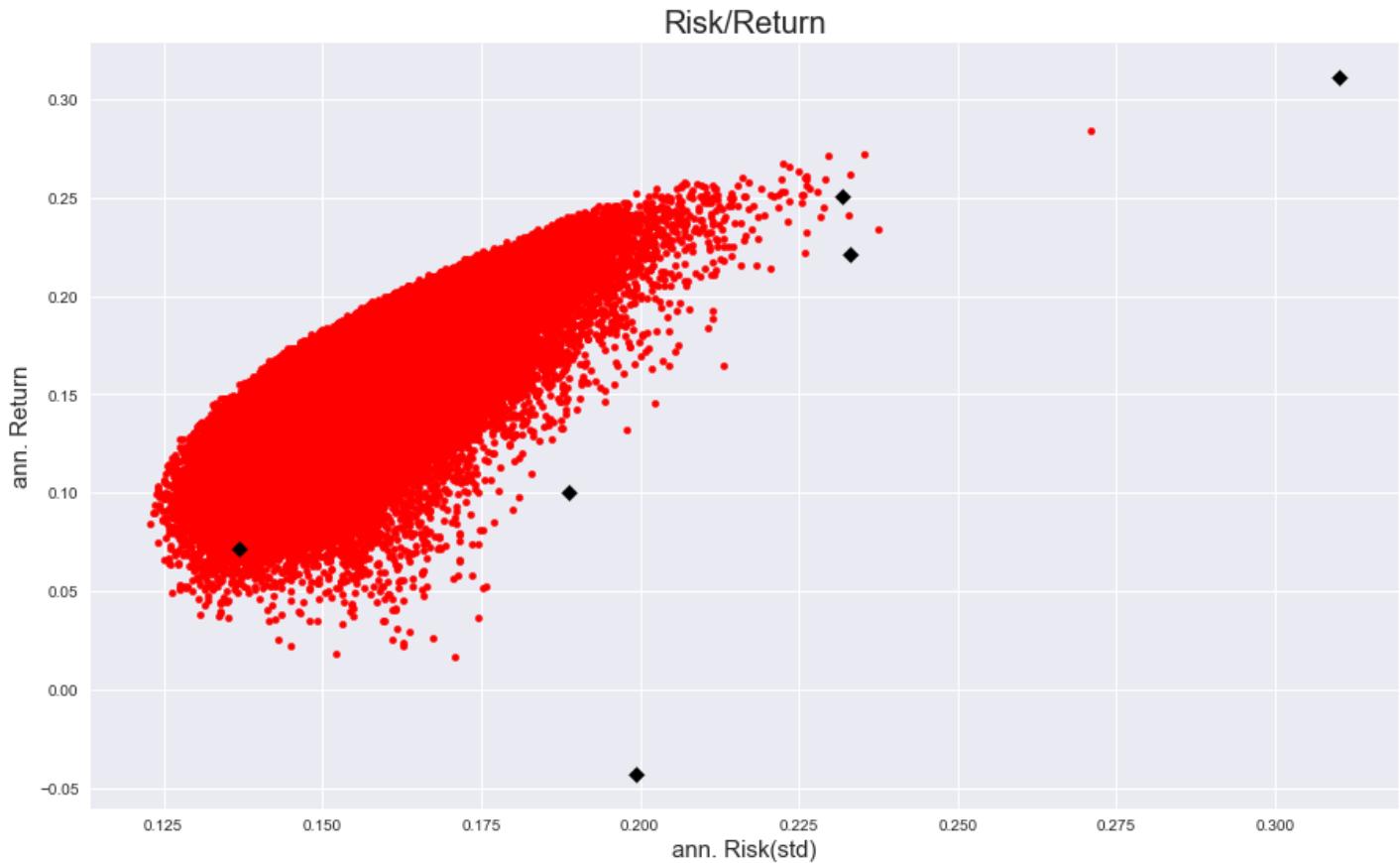


Increasing the nopol generates more portfolios.

The nopol was increased from 10 to 100,000. Then rerun the seed code. Then run the weights code. Just run everything below the nopol.

100,000 Portfolios

```
In [101...]: plt.figure(figsize = (15, 9))
plt.scatter(port_summary.loc[:, "Risk"], port_summary.loc[:, "Return"], s = 20, color = "red")
plt.scatter(summary.loc[:, "Risk"], summary.loc[:, "Return"], s = 50, color = "black", marker = "diamond")
plt.xlabel("ann. Risk(std)", fontsize = 15)
plt.ylabel("ann. Return", fontsize = 15)
plt.title("Risk/Return", fontsize = 20)
plt.show()
```



New Section

Sharpe Ratio and the Risk Free Asset

People would assume that a risk would be determined by having the Return (mean) / standard deviation. The higher values would mean that the returns are greater with less of a risk. William F. Sharpe however created an accurate way to gauge which portfolio is less risky.

Sharpe Ratio Formula

$$\text{Sharpe Ratio} = \frac{R_p - R_f}{\sigma_p}$$

R_p = return of portfolio

R_f = risk-free rate

σ_p = standard deviation of the portfolio's excess return

INSIDER

In [103...]

```
risk_free_return = 0.017
risk_free_risk = 0
```

```
In [104...]
```

```
rf = [risk_free_return, risk_free_risk]
```

Risk Free Assets

Risk free assets are government bonds from highly stable countries such as the United States or Germany. Nothing is risk free but that is considered risk free by convention.

Portfolio Analysis with The Sharpe Ratio

```
In [105...]
```

```
summary
```

```
Out[105...]
```

	Return	Risk
AMZN	0.31	0.31
BA	0.22	0.23
DIS	0.10	0.19
IBM	-0.04	0.20
KO	0.07	0.14
MSFT	0.25	0.23

```
In [106...]
```

```
port_summary.head()
```

```
Out[106...]
```

	Return	Risk
0	0.15	0.15
1	0.19	0.17
2	0.11	0.16
3	0.13	0.15
4	0.20	0.17

```
In [113...]
```

```
summary["Sharpe"] = (summary["Return"] . sub(rf[0])) / summary["Risk"]
```

```
In [124...]
```

```
port_summary["Sharpe"] = (port_summary["Return"] . sub(rf[0])) / port_summary["Risk"]
```

```
In [125...]
```

```
summary
```

```
Out[125...]
```

	Return	Risk	Sharpe
AMZN	0.31	0.31	0.95
BA	0.22	0.23	0.88
DIS	0.10	0.19	0.44
IBM	-0.04	0.20	-0.30
KO	0.07	0.14	0.40

	Return	Risk	Sharpe
--	--------	------	--------

MSFT	0.25	0.23	1.01
------	------	------	------

In [126...]

```
port_summary.head()
```

Out[126...]

	Return	Risk	Sharpe
--	--------	------	--------

0	0.15	0.15	0.87
1	0.19	0.17	1.03
2	0.11	0.16	0.57
3	0.13	0.15	0.79
4	0.20	0.17	1.04

In [127...]

```
port_summary.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100000 entries, 0 to 99999
Data columns (total 3 columns):
 #   Column   Non-Null Count   Dtype  
---  --  
 0   Return   100000 non-null   float64
 1   Risk     100000 non-null   float64
 2   Sharpe   100000 non-null   float64
dtypes: float64(3)
memory usage: 3.1 MB
```

In [128...]

```
port_summary.describe()
```

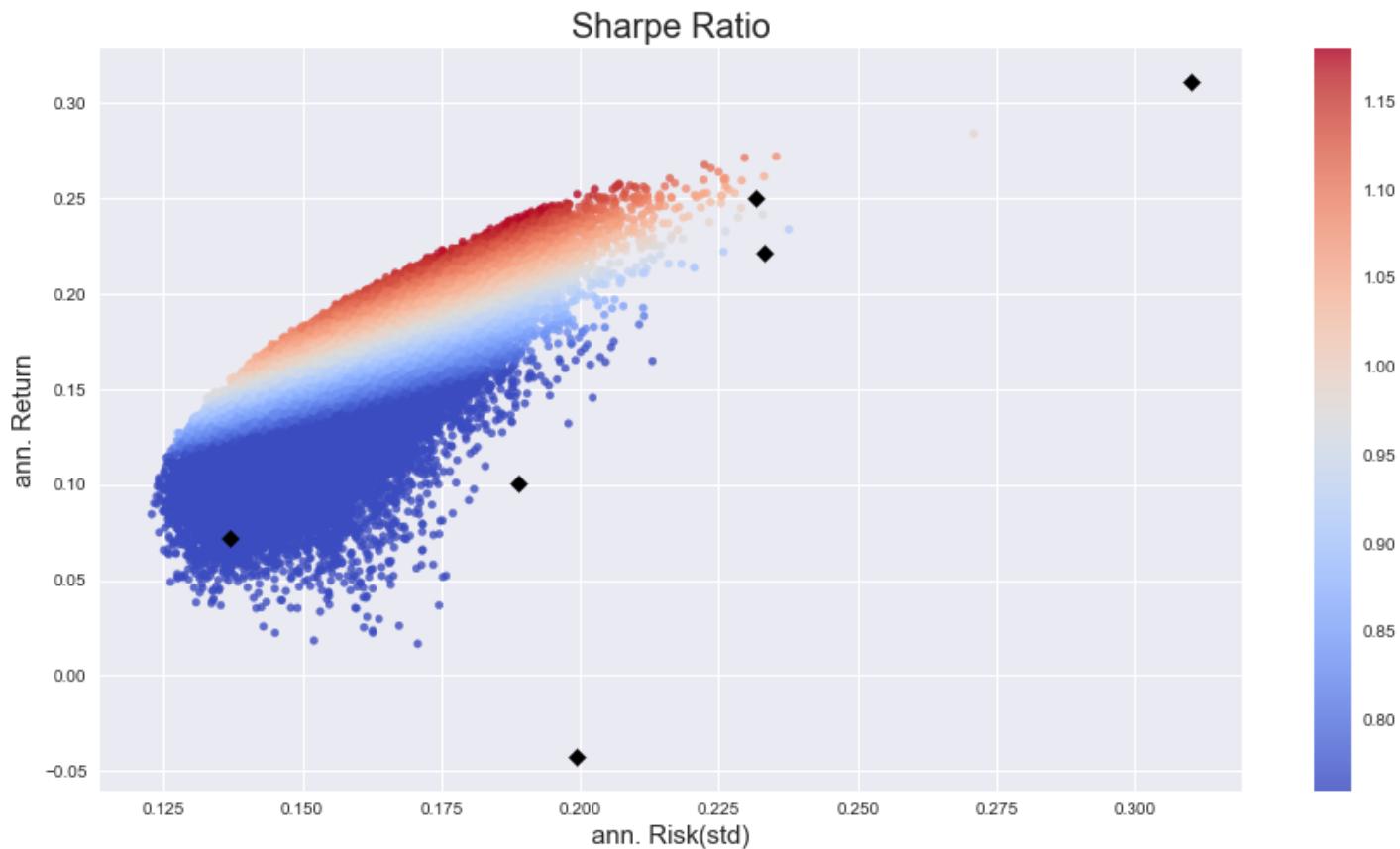
Out[128...]

	Return	Risk	Sharpe
count	1000000.00	1000000.00	1000000.00
mean	0.15	0.16	0.86
std	0.03	0.01	0.15
min	0.02	0.12	-0.00
25%	0.13	0.15	0.76
50%	0.15	0.16	0.87
75%	0.17	0.16	0.97
max	0.28	0.27	1.18

In [137...]

```
plt.figure(figsize = (15, 8))
plt.scatter(port_summary.loc[:, "Risk"], port_summary.loc[:, "Return"], s = 20,
            c = port_summary.loc[:, "Sharpe"], cmap = "coolwarm", vmin = 0.76, vmax = 1.18,
            plt.colorbar()
plt.scatter(summary.loc[:, "Risk"], summary.loc[:, "Return"], s = 50, marker = "D", c = "k"
plt.xlabel("ann. Risk(std)", fontsize = 15)
plt.ylabel("ann. Return", fontsize = 15)
plt.title("Sharpe Ratio", fontsize = 20)
plt.show()
```

```
C:\Users\alonz\AppData\Local\Temp\ipykernel_17904\86398422.py:4: MatplotlibDeprecationWarning: Auto-removal of grids by pcolor() and pcolormesh() is deprecated since 3.5 and will be removed two minor releases later; please call grid(False) first.  
plt.colorbar()
```



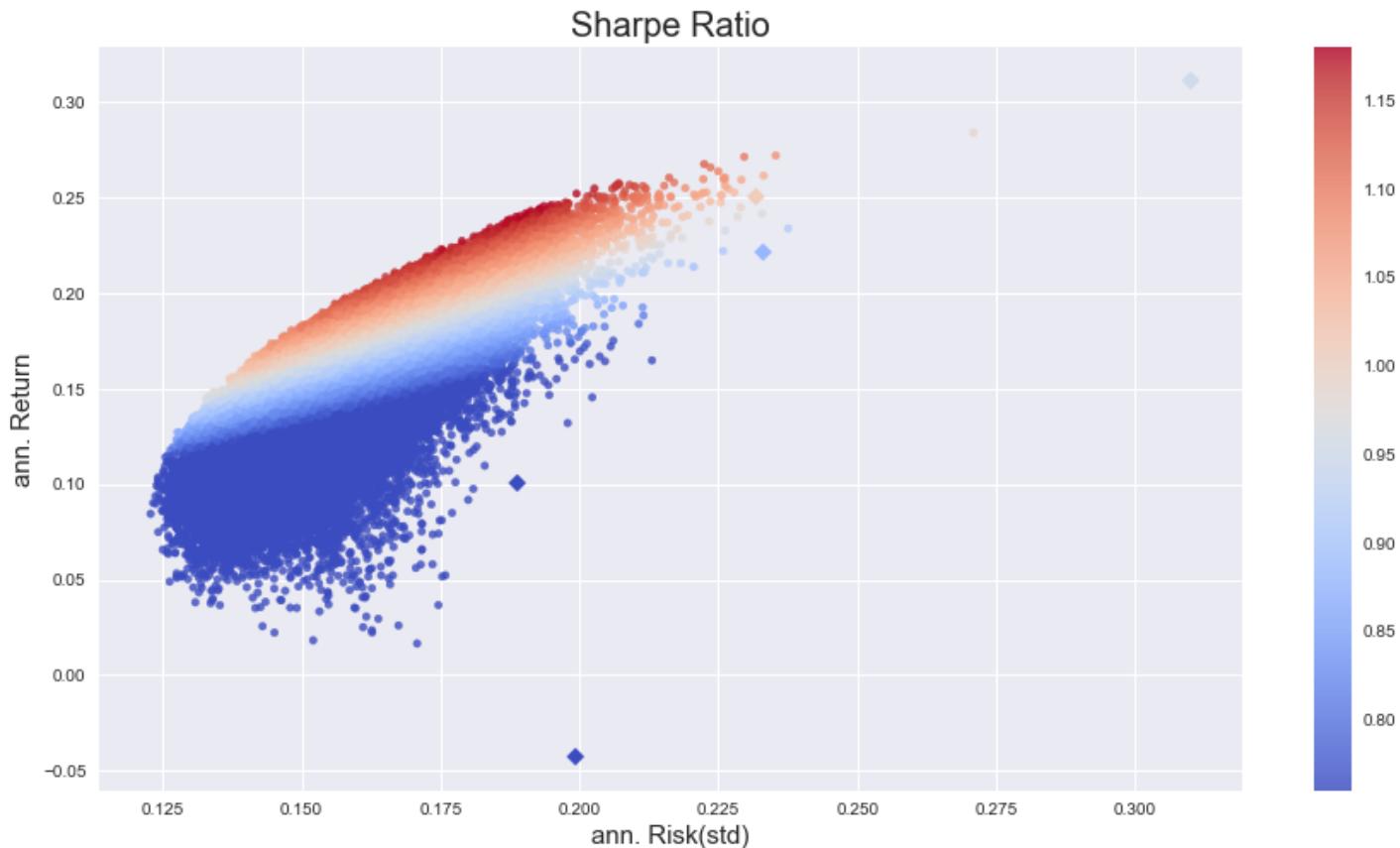
Colormaps for individual stocks.

In [133...]

```
plt.figure(figsize = (15, 8))  
plt.scatter(port_summary.loc[:, "Risk"], port_summary.loc[:, "Return"], s = 20,  
           c = port_summary.loc[:, "Sharpe"], cmap = "coolwarm", vmin = 0.76, vmax = 1.18,  
           plt.colorbar()  
plt.scatter(summary.loc[:, "Risk"], summary.loc[:, "Return"], s = 50, marker = "D",  
           c = summary.loc[:, "Sharpe"], cmap = "coolwarm", vmin = 0.76, vmax = 1.18,  
           plt.xlabel("ann. Risk(std)", fontsize = 15)  
plt.ylabel("ann. Return", fontsize = 15)  
plt.title("Sharpe Ratio", fontsize = 20)  
plt.show()
```

```
C:\Users\alonz\AppData\Local\Temp\ipykernel_17904\2626721727.py:4: MatplotlibDeprecationWarning: Auto-removal of grids by pcolor() and pcolormesh() is deprecated since 3.5 and will be removed two minor releases later; please call grid(False) first.
```

```
plt.colorbar()
```



Finding the Optimal Portfolio - The "Max Sharpe Ratio Portfolio"

In [138...]

```
port_summary.head()
```

Out[138...]

	Return	Risk	Sharpe
0	0.15	0.15	0.87
1	0.19	0.17	1.03
2	0.11	0.16	0.57
3	0.13	0.15	0.79
4	0.20	0.17	1.04

In [139...]

```
port_summary.describe()
```

Out[139...]

	Return	Risk	Sharpe
count	1000000.00	1000000.00	1000000.00
mean	0.15	0.16	0.86
std	0.03	0.01	0.15
min	0.02	0.12	-0.00
25%	0.13	0.15	0.76
50%	0.15	0.16	0.87
75%	0.17	0.16	0.97

	Return	Risk	Sharpe
max	0.28	0.27	1.18

In [140...]

weights

Out[140...]

```
array([[0.23988467, 0.09855488, 0.07813438, 0.18988917, 0.24780647,
       0.14573043],
       [0.27161429, 0.18965776, 0.13319     , 0.10859361, 0.09504023,
       0.20190411],
       [0.22014101, 0.02995528, 0.19979801, 0.37043624, 0.09160159,
       0.08806788],
       ...,
       [0.19847807, 0.32558367, 0.3338995 , 0.03835411, 0.09651877,
       0.00716587],
       [0.19679329, 0.12805869, 0.23314497, 0.10419412, 0.04966927,
       0.28813966],
       [0.07487263, 0.28158464, 0.16326642, 0.10439645, 0.08389481,
       0.29198504]])
```

idxmax() returns the index of the maximum value

In [141...]

```
msrp = port_summary.Sharpe.idxmax()
msrp
```

Out[141...]

68826

In [143...]

```
msrp_p = port_summary.loc[msrp]
msrp_p
```

Out[143...]

```
Return    0.24
Risk     0.19
Sharpe   1.18
Name: 68826, dtype: float64
```

In [144...]

```
msrp_w = weights[msrp, :]
msrp_w
```

Out[144...]

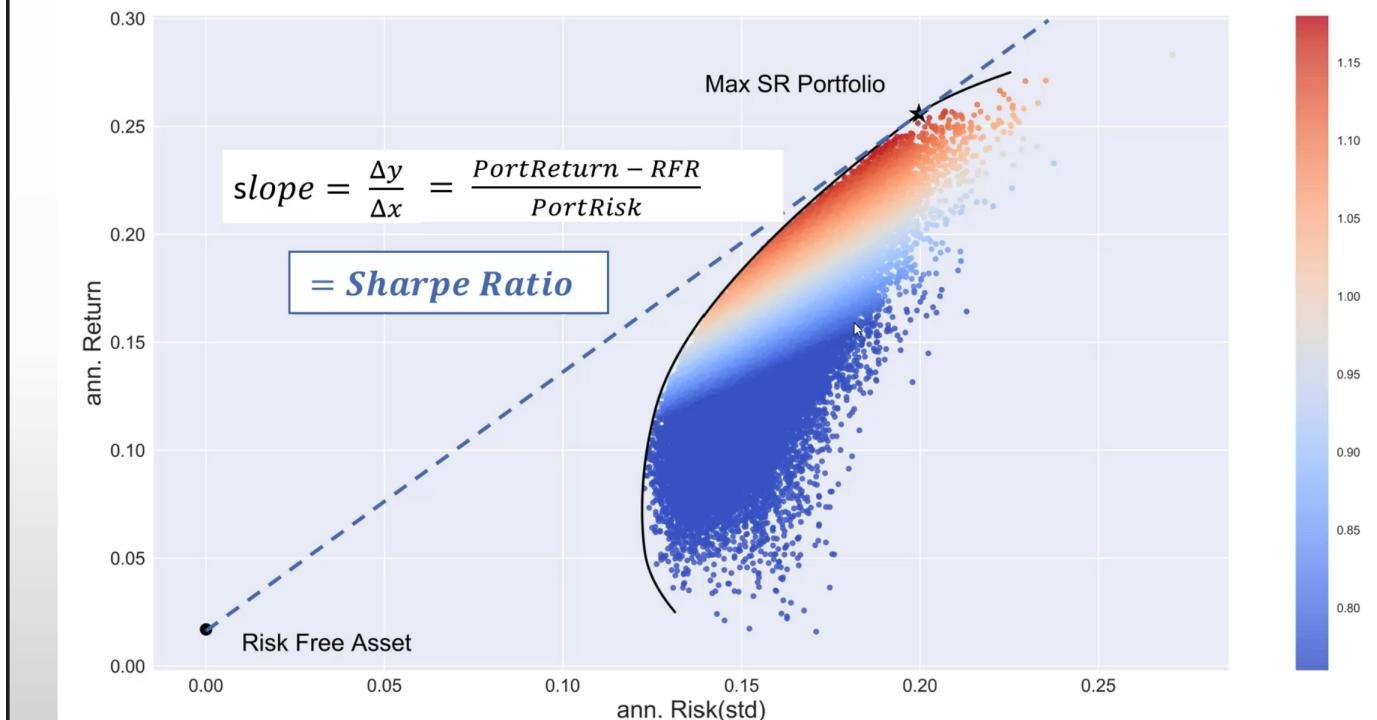
```
array([0.25239442, 0.32584579, 0.01031831, 0.00973711, 0.06671478,
       0.3349896])
```

In [145...]

```
pd.Series(index = stocks.columns, data = msrp_w)
```

Out[145...]

```
AMZN    0.25
BA      0.33
DIS     0.01
IBM     0.01
KO      0.07
MSFT    0.33
dtype: float64
```



In []:

Interacting Plotting with Plotly and Cufflinks

Creating Offline Graphs in Jupyter Notebooks

```
In [57]: ┌─▶ import pandas as pd
      import cufflinks as cf
```

```
In [58]: ┌─▶ stocks = pd.read_csv("stocks.csv", header = [0,1], index_col = [0], parse_
```

```
In [59]: ┌─▶ stocks.head()
```

Out[59]:

	AAPL	BA	DIS	IBM	KO	MSFT
Date						
2010-01-04	7.643214	56.180000	32.070000	126.625237	28.520000	30.950001
2010-01-05	7.656429	58.020000	31.990000	125.095604	28.174999	30.959999
2010-01-06	7.534643	59.779999	31.820000	124.282982	28.165001	30.770000
2010-01-07	7.520714	62.200001	31.830000	123.852776	28.094999	30.450001
2010-01-08	7.570714	61.599998	31.879999	125.095604	27.575001	30.660000

```
In [60]: ┌─▶ norm = stocks.div(stocks.iloc[0, :]).mul(100)
```

```
In [61]: ┌─▶ norm.head()
```

Out[61]:

	AAPL	BA	DIS	IBM	KO	MSFT
Date						
2010-01-04	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
2010-01-05	100.172893	103.275187	99.750546	98.792000	98.790318	100.032305
2010-01-06	98.579511	106.407972	99.220455	98.150247	98.755261	99.418416
2010-01-07	98.397266	110.715558	99.251638	97.810499	98.509814	98.384491
2010-01-08	99.051443	109.647558	99.407544	98.792000	96.686537	99.063002

Use iplot() for beautiful graphs

Must create an offline graph for iplot() use:

```
cf.go_offline()
```

```
In [62]: ► cf.go_offline()
```

Going online will break the code unless you have an iplot() account.

```
In [63]: ► #cf.go_online()
```

Permenatly offline for file

Works even when clearing the output and not running the other offline code.

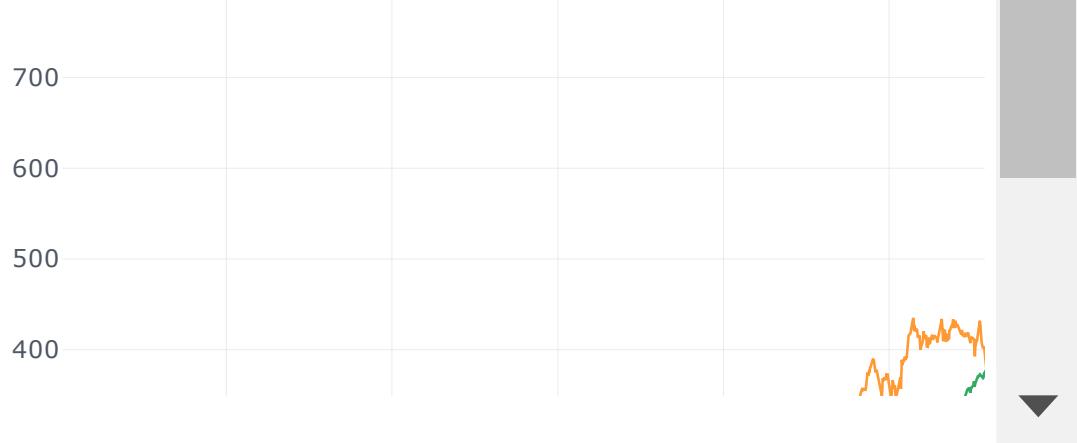
```
In [64]: ► cf.set_config_file(offline = True)
```

Interacting with iplot()

Can isolate to a single line by double clicking. Can disable or restore a graph by single clicking. Middle click and drag to make a smaller focused window. Double click to zoom back out. Same as the zoom button.

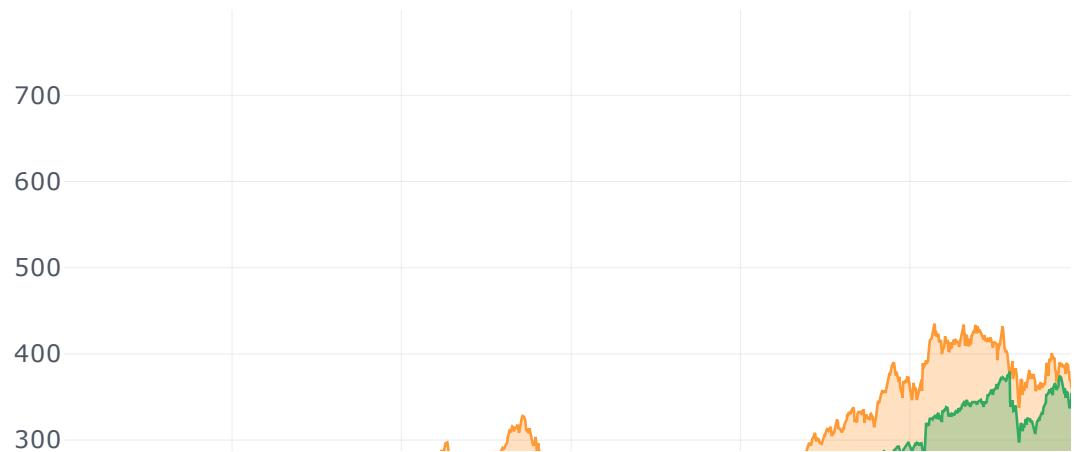
Press reset axis to undo zoom ass well.

```
In [65]: norm.iplot()
```



Filling a Graph

```
In [66]: norm.iplot(kind = "line", fill = True )
```



```
In [67]: ┌ # Run to get all of the assortments of colors.  
cf.colors.scales()
```

accent

blues

brbg

bugn

bupu

dark2

dflt

ggplot

gnbu

greens

greys

henanigans

oranges

original

orrd

paired

pastel1

pastel2

piyg

plotly

polar

prgn

pubu

pubugn

puor

purd

purples

rdbu

rdgy

rdpu

rdylbu

rdylgn

reds

set1

set2

set3

spectral

ylgn

ylgnbu

ylorbr

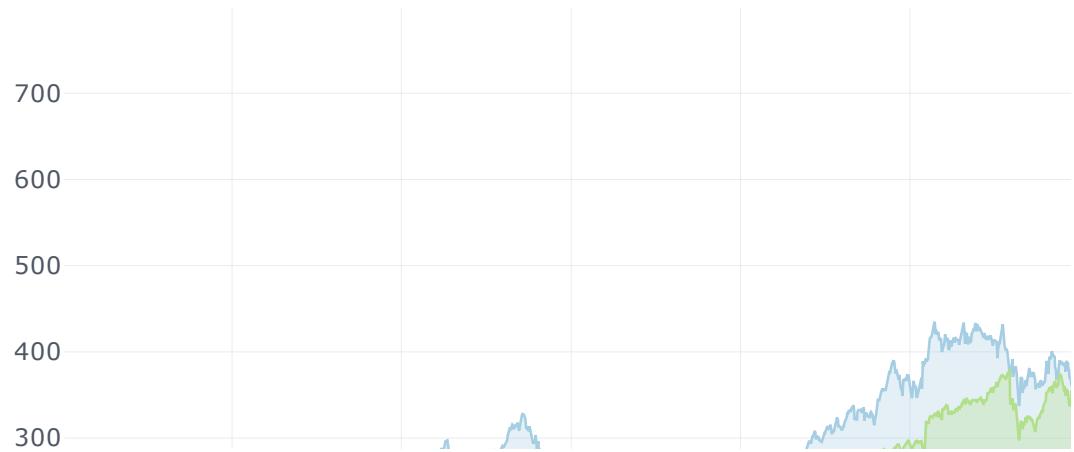
ylorrd



```
In [68]: norm.iplot(kind = "line", fill = True, colorscale = "greens")
```



```
In [69]: ┏ norm.iplot(kind = "line", fill = True, colorscale = "paired")
```

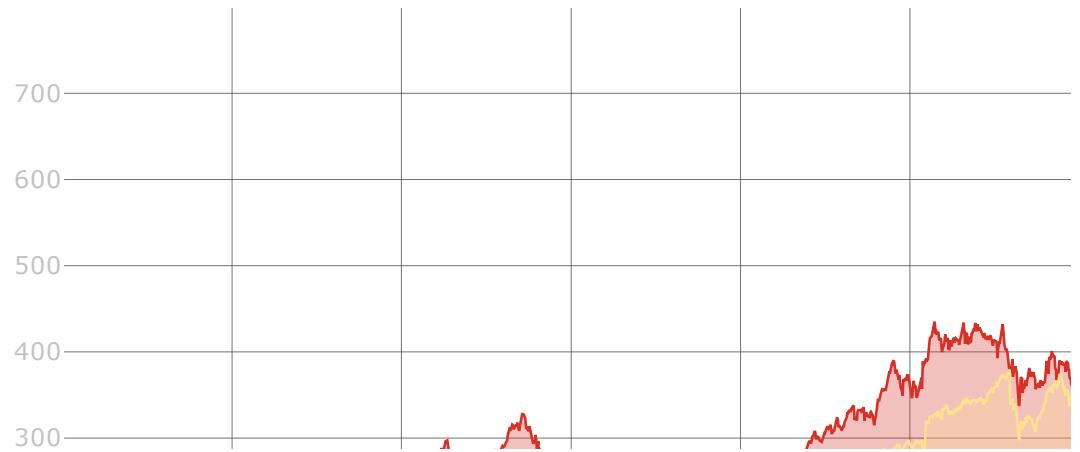


```
In [70]: ┏ # Get Additional Themes
cf.getThemes()
```

```
Out[70]: ['ggplot', 'pearl', 'solar', 'space', 'white', 'polar', 'henanigans']
```

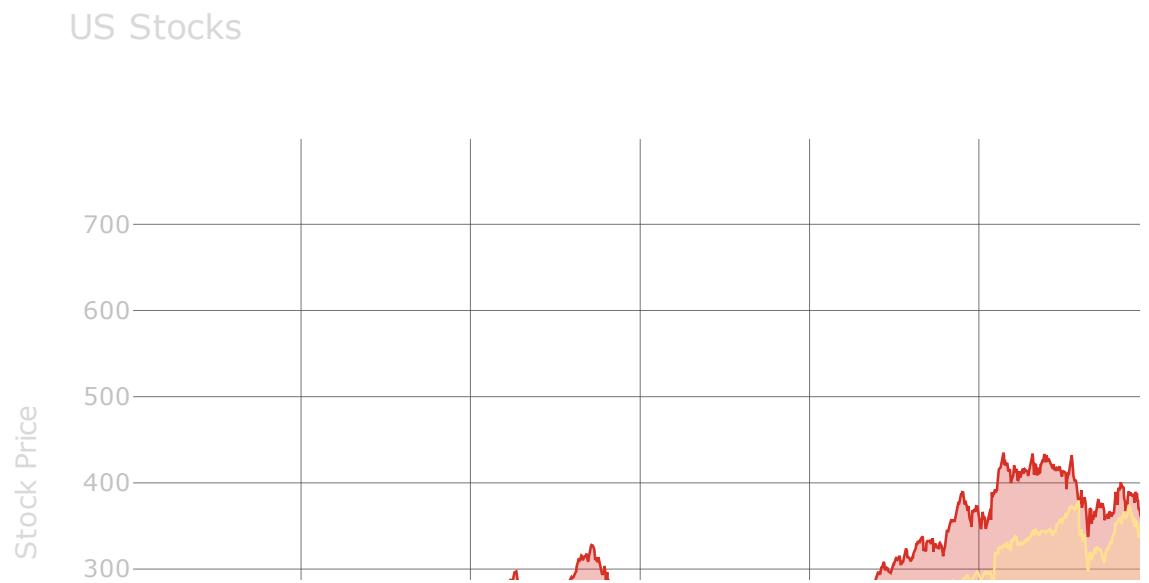
colorscale = "rdylbu" gives a bloomberg type graph

```
In [71]: norm.iplot(kind = "line", fill = True, colorscale = "rdylbu", theme = "sol
```



Can add a title to the graph also.

```
In [72]: norm.iplot(kind = "line", fill = True, colorscale = "rdylbu", theme = "solarized", title = "US Stocks", xTitle = "Time", yTitle = "Stock Price")
```



```
In [73]: norm[["AAPL", "BA"]].iplot(kind = "spread", fill = True, colorscale = "rdytitle = "AAPL vs. BA", xTitle = "Time", yTitle = "Stock Price")
```

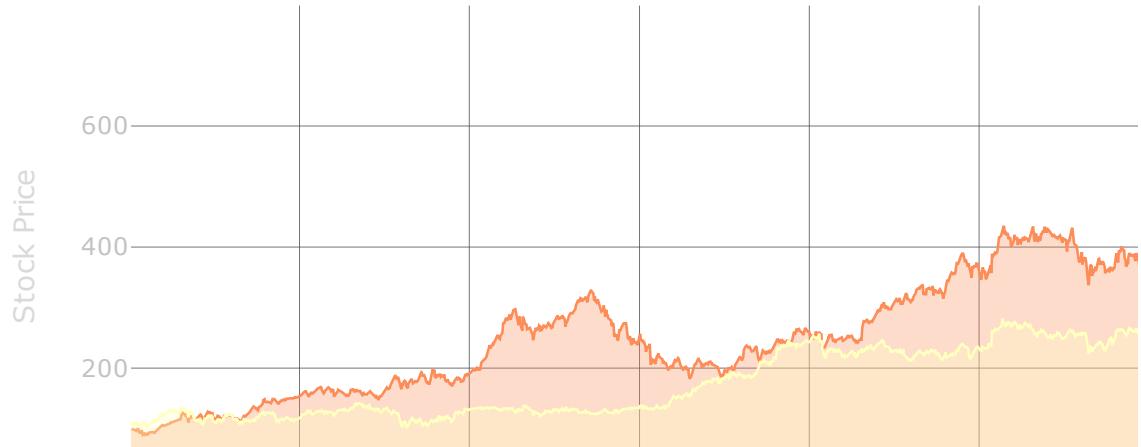
C:\Users\alonz\anaconda3\lib\site-packages\cufflinks\plotlytools.py:849:
FutureWarning:

The pandas.np module is deprecated and will be removed from pandas in a future version. Import numpy directly instead.

C:\Users\alonz\anaconda3\lib\site-packages\cufflinks\plotlytools.py:850:
FutureWarning:

The pandas.np module is deprecated and will be removed from pandas in a future version. Import numpy directly instead.

AAPL vs. BA



Creating Interactive Histograms

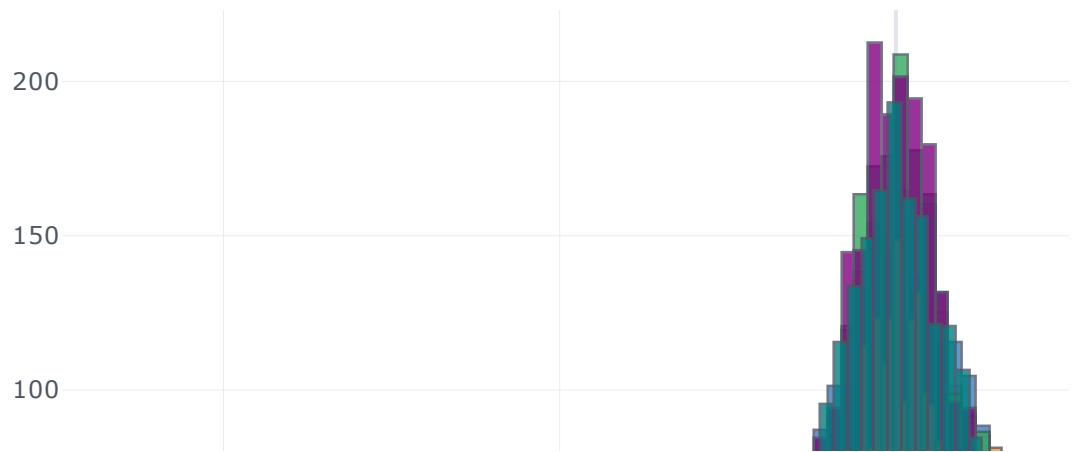
```
In [74]: └ stocks.head()
```

Out[74]:

	AAPL	BA	DIS	IBM	KO	MSFT
Date						
2010-01-04	7.643214	56.180000	32.070000	126.625237	28.520000	30.950001
2010-01-05	7.656429	58.020000	31.990000	125.095604	28.174999	30.959999
2010-01-06	7.534643	59.779999	31.820000	124.282982	28.165001	30.770000
2010-01-07	7.520714	62.200001	31.830000	123.852776	28.094999	30.450001
2010-01-08	7.570714	61.599998	31.879999	125.095604	27.575001	30.660000

```
In [75]: └ ret = stocks.pct_change().dropna()
```

```
In [76]: ret.iplot(kind = "histogram")
```

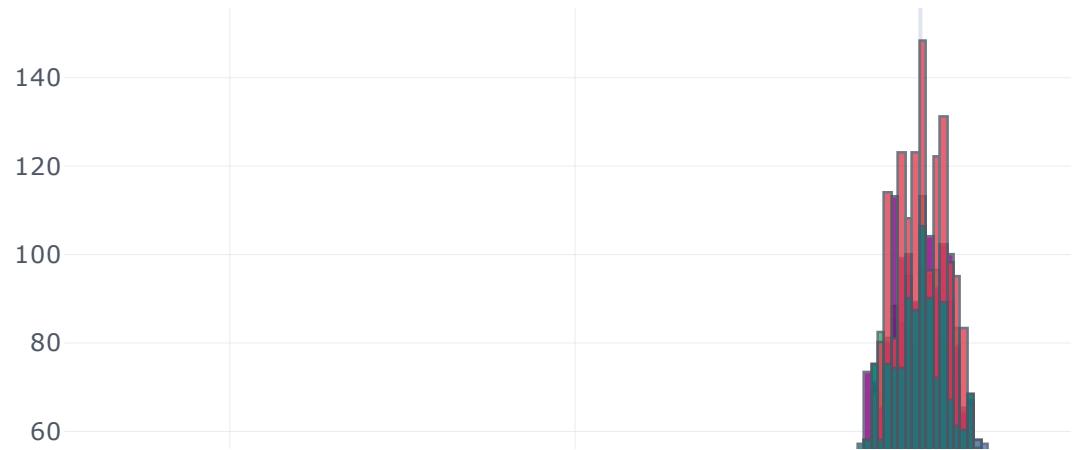


```
In [77]: ret.iplot(kind = "histogram", bins = 100, histnorm = None)
```



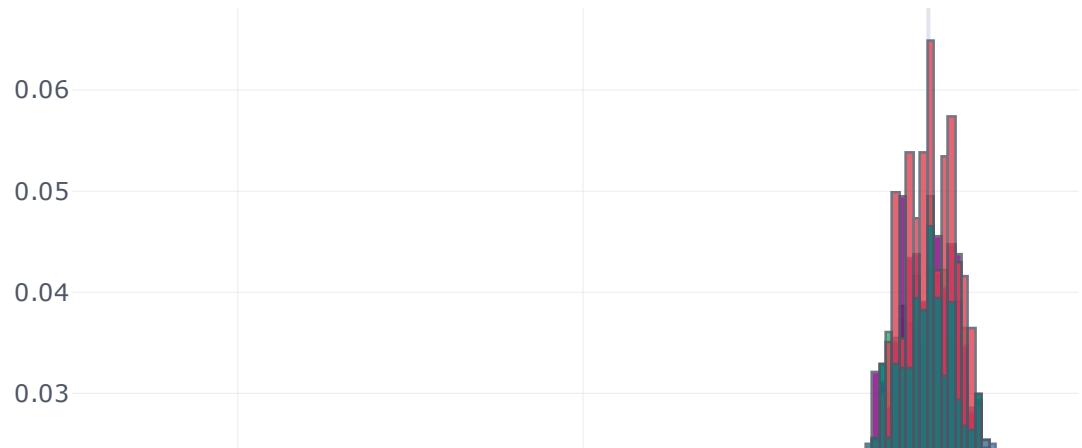
Create a universal bin size for the chart.

```
In [78]: ret.iplot(kind = "histogram", bins = (-0.15, 0.1, 0.001 ), histnorm = None)
```



histnorm = "probability"

```
In [79]: █ ret.iplot(kind = "histogram", bins = (-0.15, 0.1, 0.001 ), histnorm = "prc")
```



Interactive Candlestic and OHLC Charts

This stands for Open, High, Low, and Close.

```
In [80]: █ import pandas as pd
import cufflinks as cf
```

```
In [81]: █ stocks = pd.read_csv("stocks.csv", header = [0,1], index_col = [0], parse_
```

```
In [82]: ┌── stocks.head()
```

Out[82]:

	Adj Close				Close			
	AAPL	BA	DIS	IBM	KO	MSFT	AAPL	BA
Date								
2010-01-04	6.544689	43.777542	27.933924	82.858467	19.496908	23.855656	7.643214	56.180000
2010-01-05	6.556003	45.211342	27.864239	81.857529	19.261059	23.863369	7.656429	58.020000
2010-01-06	6.451723	46.582794	27.716160	81.325775	19.254217	23.716917	7.534643	59.779999
2010-01-07	6.439794	48.468548	27.724876	81.044273	19.206371	23.470268	7.520714	62.200001
2010-01-08	6.482609	48.001011	27.768423	81.857529	18.850885	23.632132	7.570714	61.599998

5 rows × 36 columns

```
In [83]: ┌── aapl = stocks.swaplevel(axis = 1).AAPL
```

```
In [84]: ┌── aapl.head()
```

Out[84]:

	Adj Close	Close	High	Low	Open	Volume
Date						
2010-01-04	6.544689	7.643214	7.660714	7.585000	7.622500	493729600
2010-01-05	6.556003	7.656429	7.699643	7.616071	7.664286	601904800
2010-01-06	6.451723	7.534643	7.686786	7.526786	7.656429	552160000
2010-01-07	6.439794	7.520714	7.571429	7.466071	7.562500	477131200
2010-01-08	6.482609	7.570714	7.571429	7.466429	7.510714	447610800

Can create candles by importing Open, High, Low, Close, information

```
In [85]: # Candle chart
aapl.loc["5-2017": "9-2017"].iplot(kind = "candle")
```



OHLC chart

```
In [86]: ┏ aapl.loc["5-2017": "9-2017"].iplot(kind = "ohlc")
```



Adding Simple Moving Averages and Bollinger Bands

```
In [87]: ┏ aapl.head()
```

Out[87]:

	Adj Close	Close	High	Low	Open	Volume
Date						
2010-01-04	6.544689	7.643214	7.660714	7.585000	7.622500	493729600
2010-01-05	6.556003	7.656429	7.699643	7.616071	7.664286	601904800
2010-01-06	6.451723	7.534643	7.686786	7.526786	7.656429	552160000
2010-01-07	6.439794	7.520714	7.571429	7.466071	7.562500	477131200
2010-01-08	6.482609	7.570714	7.571429	7.466429	7.510714	447610800

.qf has a lot of features. Press tab to see them all.

qf is a part of cufflinks.

```
In [88]: qf = cf.QuantFig(df = aapl.loc["5-2017": "9-2017"])
```

```
In [89]: type(qf)
```

```
Out[89]: cufflinks.quant_figure.QuantFig
```

```
In [90]: qf.iplot(title = "AAPL", name = "AAPL")
```

AAPL



```
In [91]: qf.add_sma(periods = 20)
```

```
In [92]: qf.add_bollinger_bands(periods = 20, boll_std = 2)
```

```
In [93]: ┌─ qf.iplot(title = "AAPL", name = "AAPL")
```

AAPL

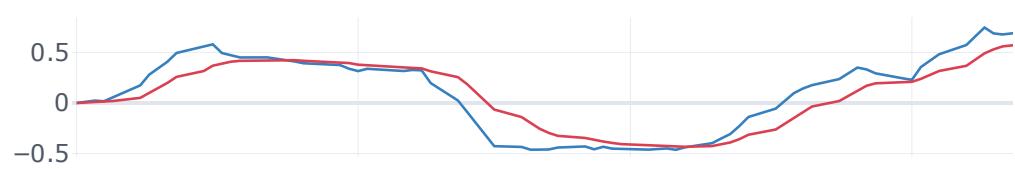


Adding More Technical Indicators

```
In [94]: ┌─ qf = cf.QuantFig(df = aapl.loc["5-2017": "9-2017"])
```

```
In [95]: ┌─ qf.add_bollinger_bands(periods = 20, boll_std = 2)
qf.add_sma(periods = 20)
qf.add_macd()
qf.add_volume()
qf.add_dmi()
```

```
In [96]: ► qf.iplot(title = "AAPL", name = "AAPL")
```



To add missing values use:

```
data_set[0,0] = pd.NA
```

```
In [ ]: ►
```

The Time Value of Money

Money is worth more today than it is at a later date. Money that is in one's posession now is able to be saved and invested unlike money that is promised to one in the future.

Formula:

$$FV = PV(1 + r)^n$$

The present value times the sum of one and the interest rate to the power of the number of periods is the future value of money.

Note that the interest compounds. This means that the interest earns interest.

Question 1

How much USD has to be saved to get a \$110 in one year if the interest p.a is 4.5%?

$$FV = 110$$

$$\text{Interest} = 4.5\%$$

$$PV = ?$$

$$\text{Periods} = 1$$

$$110 = PV(1+0.045)^1$$

$$110 = 1.045 / 1.045$$

$$/1.045$$

$$PV = \$105.26$$

Calculate Future Values (compounding)

In [3]:

```
import numpy as np
```

Question 1:

Calculate the future value of one period if you start out with 100 USD and save it for one year at an interest rate of 3%.

Use exponents with np.power(base_num, exponent)

```
np.power(base_num, exponent)
```

base ** power

Another way to find the power using exponents.

In [7]:

```
# While using np.power seems superfluous in this case it is a good habit to get into.  
100 * np.power(1 + 0.03, 1)
```

Out[7]: 103.0

Question 2:

You have 100 USD today and save it for three years at an interest rate of 3%.

In [10]:

```
100 * np.power(1 + 0.03, 3)
```

Out[10]: 109.2727

Discounting

Discounting is seeing how much money to have to save to meet your future value goals.

Calculate Present Values (discounting)

$$FV = PV(1 + r)^n$$

Question 3:

How many USD to save today at an interest rate of 4.5% p.a to get 110 USD in one year?

In [12]:

```
110 / np.power(1 + 0.045, 1)
```

Out[12]:

105.26315789473685

In [13]:

```
# Using this formula allows no need for re-evaluation for complex formulas.  
110 / np.power(1 + 0.045, 1)
```

Out[13]:

105.26315789473685

How many USD to save today at an interest rate of 4.5% p.a to get 110 USD in three years?

In [14]:

```
110 / np.power(1 + 0.045, 3)
```

Out[14]:

96.39262644604003

Solving for Interest Rate

$$FV = PV(1 + r)^n$$

$FV = (1 + r)^n$ $PV = 1 + r$ $r = (1/n)^{1/n} - 1$ $r \text{ percent} = (1/n)^{1/n} - 1 * 100$ Note that in order to actually have to do a radical (or at least a square root) the period has to be 2 or greater.

Radicals

Radicals is what square roots fall into. To do a cube root etc. do a power of a fraction.

Example find the cube root of 8:

```
In [16]: 8 ** (1/3)
```

```
Out[16]: 2.0
```

Today you receive the offer to deposit 90 USD in a savings account, getting back 93.5 USD back in one year. What is the interest rate.

```
In [20]:
```

```
# No square root because the period is only one. That is why it is best to think of it as
((93.5/90) ** (1/1) - 1) * 100
```

```
Out[20]:
```

```
3.8888888888888973
```

Today you receive the offer to deposit 90 in a savings account, getting back 93.5 USD in three years.

```
In [21]:
```

```
((93.5/90) ** (1/3) - 1) * 100
```

```
Out[21]:
```

```
1.2798463496144663
```

Using Variables

Variables can be used to make calculations more simplistic and intuitive.

```
In [24]:
```

```
pv = 100
r = 0.03
n = 3
```

Can make a variable for the answer.

```
In [27]:
```

```
fv = pv * (1 + r) ** n
fv
```

```
Out[27]:
```

```
109.2727
```

For the print statements several things can be printed by using a comma.

```
In [29]:
```

```
print(2+2, 100 + 800, "Different statements")
```

```
4 900 Different statements
```

Use `sep = "\n"` to create a new line between prints.

```
In [30]:
```

```
print(2+2, 100 + 800, "Different statements", sep = "\n")
```

```
4
900
Different statements
```

Cashflow Example

Example:

Today you have 100 USD in your savings account and you save another

- 10 USD in t1
- 20 USD in t2
- 50 USD in t3
- 30 USD in t4
- 25 USD in t5. (each cf at period's end)

Calculate the **FV** of your savings account **after 5 years** given an interest rate of **3% p.a.**

Formula:

$$FV_N = \sum_{t=0}^N CF_t * (1 + r)^{N-t}$$

The formula is basically saying to get the future value you need to sum up all of the other values interest that they have accumulated.

Today you agreed on a payout plan that guarantees payouts of

- 50 USD in t1,
- 60 USD in t2,
- 70 USD in t3,
- 80 USD in t4,
- 100 USD in t5. (each cf at period's end)

Calculate the Funding amount / **PV** that needs to be paid into the plan today (t0). Assume an interest rate of **4% p.a.**

In [60]:

```
cf = [50, 60, 70, 80, 100]
f = 1.04
```

Solving for CF in this case: $FV^{**}(1/N-t) = CF(1+r)$

In [61]:

```
PV = 0
for i in range(5):
    PV += cf[i] / f ** (i + 1)
print(PV)
```

```
48.07692307692307
103.55029585798815
165.78004096495218
234.16437624733024
316.3570869232654
```

Net present Value.

NPV uses the same formula. The only thing that is different is that the formula is used to see if the investment a company makes will recoup it through revenue.

The value of a company is the sum of their NPV.

Time or t is very important because it compares the opportunity cost of earning the money with putting it in the bank and earning interest, at the rate of return of the machine.

Formula:

$$NPV = I_o + \sum_{t=1}^N \frac{CF_t}{(1+r)^t}$$

NPV: Net Present Value
I_o: Initial Investment (negative)
CF_t: cashflow @ timestamp t
N: Total number of periods
r: required rate of return
t = timestamp (0, 1, ..., N)

Required Rate of Return (Cost of Capital)

$$NPV = I_o + \sum_{t=1}^N \frac{CF_t}{(1+r)^t}$$

WACC

Intuition behind the Required Rate of Return:

- Opportunity Costs: (Expected) Return of comparable / alternative Projects
- Weighted Average Costs to fund Capital Outflow I_o
 - Cost of Debt (Interest Rate charged by Bondholders / Banks)
 - Cost of Equity (Required Return by Shareholders)

The XYZ Company evaluates to buy an additional machine that will increase future profits/cashflows by

- 20 USD in t1,
- 50 USD in t2,
- 70 USD in t3,
- 100 USD in t4,
- 50 USD in t5. (each cf at period's end)

Note that the machine costs \$200 and the rate of return is 6% a year.

In [90]:

```
# The intial investment is the first period.
cf = [-200, 20, 50, 70, 100, 50]
f = 1.06
NPV = 0
```

Example Calculation: The initial slot in the list should have the start up cost. Note that the cf[i] only selects a slot in the index it is NOT multiplication. Let's start at slot 1 not slot 0. Note the r and the 1 has already been summed. cf[1] = 20 20 / (1.06¹) = 18.86 One more cf[2] = 50 50 / (1.06²) = 44.49

In [91]:

```
for i in range(6):
    NPV += cf[i] / f ** (i)
    print(cf[i] / f ** (i))
```

```
#NPV
print("NPV:", NPV)
```

```
-200.0
18.867924528301884
44.499822000711994
58.773349812261124
79.20936632380204
37.36290864330285
NPV: 38.71337130837991
```

What if the purchase price was 250 USD?

```
In [99]: # Modify the list by assigning a value to a particular slot.
cf[0] = -250
```

```
In [100... cf
Out[100... [-250, 20, 50, 70, 100, 50]
```

Reject the project due to NPV being negative.

That means capital cost is not covered.

```
In [103... NPV = 0
for i in range(6):
    NPV += cf[i] / f ** (i)
    print(cf[i] / f ** (i))
NPV
```

```
-250.0
18.867924528301884
44.499822000711994
58.773349812261124
79.20936632380204
37.36290864330285
-11.286628691620088
Out[103...]
```

Break Even for Projects

```
In [111... cf = [-200, 20, 50, 70, 100, 50]
```

```
In [113... cum_cf = 0
for i in range(len(cf)):
    cum_cf += cf[i]
    print(cum_cf)
    if cum_cf > 0:
        print("The Project's payback Period is {} Years!".format(i))
        break # Must terminate the loop once the break even point is met.
```

```
-200
-180
-130
-60
40
The Project's payback Period is 4 Years!
```

```
In [114...]: # New cf dataset
cf = [-200, -150, 50, 70, 100, 50]
```

```
In [115...]: cum_cf = 0
for i in range(len(cf)):
    cum_cf += cf[i]
    #print(cum_cf)
    if cum_cf > 0:
        print("The Project's Payback Period is {}".format(i))
        break
    elif cum_cf <= 0 and i == len(cf)-1:
        print("The Project does not break even.")
```

The Project does not break even.

Internal Rate of Return

Investments Projects – Internal Rate of Return (IRR)

Example:

The XYZ Company evaluates to buy an additional machine that will increase future profits/cashflows by
- 20 USD in t1,
- 50 USD in t2,
- 70 USD in t3,
- 100 USD in t4,
- 50 USD in t5. (each cf at period's end)

The machine costs 200 USD (Investment in to). Calculate the **Project's Internal Rate of Return (IRR)** and evaluate whether XYZ should pursue the project. XYZ's required rate of return (Cost of Capital) is 6%.

Formula:

$$NPV = I_0 + \sum_{t=1}^N \frac{CF_t}{(1 + IRR)^t} = 0$$

NPV: Net Present Value

I_0 : Initial Investment (negative)

CF_t : cashflow @ timestamp t

N: Total number of periods

IRR: Internal Rate of Return (NPV = 0)

t = timestamp (0, 1, ..., N)

Finding the IRR is not able to be done arithmetically. Guess and check will have to be employed in order to solve.

Iterative Process (Trial-and-Error):

1. Make a guess (IRR = 6%)
2. Calculate NPV based on guess
3. If NPV > 0: Increase guess (IRR = 6.1%)
4. If NPV < 0: Decrease guess (IRR = 5.9%)

Repeat Steps 1-4 until $NPV \approx 0$

Investments Projects and IRR

Simple Decision Rule:

Accept the Project if $IRR > \text{Required Rate of Return}$
Reject the Project if $IRR < \text{Required Rate of Return}$

Interpretation of IRR:

- (Hypothetical) Rate of Return where $NPV = 0$

Solving for a Project's IRR

The XYZ Company evaluates to buy an additional machine that will increase profits/cashflows by

- 20 USD in t_1 ,
- 50 USD in t_2 ,
- 70 USD in t_3 ,
- 100 USD in t_4 ,
- 50 USD in t_5 . (each cf at period's end)

The machine costs 200 USD (Investment in to).

- Calculate the Project's IRR and evaluate whether XYZ should pursue the project. XYZ's required rate of return is 6% p.a.

In [127...]

```
cf = [-200, 20, 50, 70, 100, 50]
```

In [128...]

```
guess = 0.06
```

In [129...]

```
f = 1 + guess
NPV = 0
for i in range(len(cf)):
    NPV += cf[i] / f **(i)
print(NPV)
```

```
38.71337130837991
```

In [130...]

```
guess += 0.01
guess
```

```
Out[130...]
```

```
0.06999999999999999
```

In [131...]

```
guess = 0.06
f = 1 + guess
step = 0.0001
NPV = 0
for i in range(len(cf)):
    NPV += cf[i] / f ** (i)
NPV
```

```
Out[131...]
```

```
38.71337130837991
```

Advanced TVM and Capital Budgeting with Numpy Financial Functions

Finance is no longer included in Numpy. Instead numpy has to be installed via Anaconda.

pip install numpy-financial

In [140...]

```
# After the installation numpy_financial can be installed
import numpy as np
import numpy_financial as npf
```

Evaluating Investments with npf.npv() and npf.irr()

The XYZ Company evaluates to buy a new machine that will increase profits/cashflows for XYZ by

- 20 USD in t1,
- 50 USD in t2,
- 70 USD in t3,
- 100 USD in t4 and
- 50 USD in t5.

The machine costs 200 USD (Investment in t0). Calculate the **Project's NPV** and **IRR** and evaluate whether XYZ should pursue the project.

XYZs required rate of return is 6%.

In [141...]

```
cf = np.array([-200, 20, 50, 70, 100, 50])
r = 0.06
```

Net Present Value with npf.npv(r, cf)

Must pass in the rate of return and the cashflow profile as numpy list. np.array([list])

```
In [143... npf.npv(r, cf)
```

```
Out[143... 38.71337130837991
```

```
In [144... npf.npv(r, cf)
```

```
Out[144... 38.71337130837991
```

Can check if the NPV is greater than zero.

```
In [145... npf.npv(r, cf) > 0
```

```
Out[145... True
```

```
In [146... npf.irr(cf)
```

```
Out[146... 0.1190693988331708
```

```
In [147... npf.irr(cf) > 0
```

```
Out[147... True
```

The XYZ Company issued a 10Y Senior Unsecured Bond one year ago with a Coupon Rate of 5.0% (annual payments in arrears). Today's Bond Price is 107.5 (per 100 par Value). Calculate the Bond's **current YTM**.

```
In [148... cf = [5] * 9
```

```
In [150... cf[-1] = cf[-1] + 100
```

```
In [151... cf.insert(0, -107.5)
```

```
In [152... cf
```

```
Out[152... [-107.5, 5, 5, 5, 5, 5, 5, 5, 5, 105]
```

```
In [153... npf.irr(cf)
```

```
Out[153... 0.039916847253554044
```

Annuities

Annuities are a stream of passive income, usually for retirees. The annuity will be funded by a patron and invested into diverse investments by a broker. After a certain period the investor will be eligible to reap a percentage of their investment each year in form of a payment.

Evaluating Annuities with np.fv() - funding phase

In [154...]

```
import numpy as np
```

You save 2,000 USD p.a. for the next 25 Years (payment at year end) and get an interest rate of 3% p.a. on your savings. What is the value of your savings account (**FV**) in 25 years?

In [155...]

```
PV = 0
cf = 2000
r = 0.03
n = 25
```

In [158...]

```
FV = npf.fv(rate = r, nper = n, pmt = cf, pv = PV)
```

In [159...]

```
FV
```

Out[159...]

```
-72918.52864361441
```

Same problem except start with a 10,000 USD balance.

In [167...]

```
PV = 10000
cf = 2000
r = 0.03
n = 25
```

In [168...]

```
# Gives the information at the end of the collection.
FV = npf.fv(rate = r, nper = n, pmt = cf, pv = PV)
```

In [172...]

```
## The negative sign shows money that is coming back. Think about a tax return.
# The amount you get could be positive if the sign of the cf was flipped.
# Both the cf and the pv must have the same sign.
```

In [173...]

```
FV
```

Out[173...]

```
-93856.30794015658
```

In []:

```
# End Section 9 | 86
```

In []:

In []:

Statistics

Important statistic terms will be defined here.

Definitions

Random variable. Describes an uncertain quantity or number.

Outcome is an observed value of a random variable.

Event is a single outcome or set of outcomes.

Discrete variables are outcomes that can be counted.

Continuous variables are outcomes that cannot be counted. For example the height of an elephant can vary virtually infinitely, without rounding and by using many decimals to measure their height.

Probability is the likelihood or odds that an event will occur. Rolling a six on a six sided die is 1/6. Flipping head has a 1/2 probability.

Frequency is the number of occurrences of an outcome/ event. Someone who rolls a dice 5x will roll different numbers at a different frequency.

Relative frequency is the percentages of the distribution of the outcomes (i.e a coin showed heads 65/ 100 times).

Probability Distribution is the sum of all of the outcomes, summed together the fractions equal one. For flipping the coin the probability distribution is 2 and $2/2 = 1$.

For rolling a dice the probability is 1/6 the probability distribution is 6/6 or 1.

For continuous variables such as height the values will form a bell shape curve. Most values will cluster around the center. The more extreme values will be around the edges.

Population is all of the members of a particular group.

Sample is the subset of the population that was actually examined for the research/survey.

Descriptive Statistics measures the whole population and processes the data in order to create statistics.

Inferential Statistics is when a subset of the population is chosen. The sample that is chosen is considered to be representative of the population. With that conclusions made using the sample is applied to the population.

Independent variables are factors that influence the dependent variables.

Dependent variables are results that are influenced by the independent variables. **Hypothesis** is using regression and other mathematical processes to assert that the one variable influences another.

Example an elephant height is the dependent variable. Its independent variables include. its Age, Gender, Parents, etc.

Estimate/Forecast is using the pre-established relationships between independent variables and dependent variables to predict the dependent variables outcome. An example would be using the gender, genetics, parents, etc. to predict how tall an elephant will be once it grows up. **Machine learning** can take pre-established relationships to make forecasts and assertions.

Importing CSV data as a List

What is the equally weighted return in 2017?

```
In [1]: import numpy as np  
np.set_printoptions(precision=2, suppress= True) I
```

Population: 2017 Price Return for all 500 Companies

```
In [2]: pop = np.loadtxt("SP500_pop.csv", delimiter = ",", usecols = 1)
```

```
In [3]: pop
```

```
Out[3]: array([ 4.69929763e-01,  1.14371386e-01, -4.10536897e-01,  4.61146607e-01,  
 5.44394762e-01,  1.74318967e-01,  6.63205538e-01,  4.85810987e-01,  
 3.07009306e-01,  7.02185527e-01,  2.25970807e-01, -1.22015334e-01,  
 1.40202374e-01,  1.09321663e-01,  4.16430212e-01,  1.24475791e-01,  
.....])
```

Calculating Percentile

Loading numpy files as text from CSV

must use:

```
np.loadtxt("file_name.csv", delimiter = ",", usecols = 1)
```

That works because delimiter knows how to separate by a certain string character. usecols must equal 1. It equals one because python is zero based and the prices are coming from the second column. Excel files can be imported as CSV even if they are not CSVs with np.loadtxt()

	A	B
1	A	0.46993
2	AAL	0.114371
3	AAP	-0.41054
4	AAPL	0.461147
5	ABBV	0.544395
6	ABC	0.174319
7	ABMD	0.663206
8	ABT	0.485811
9	ACN	0.307009
10	ADBE	0.702186
11	ADI	0.225971
12	ADM	-0.12202
13	ADP	0.140202

```
In [13]:
```

```
import numpy as np
```

```
np.set_printoptions(precision = 2, suppress = True)
pop = np.loadtxt(r"C:\Users\alonz\Documents\Course_Materials_Part2_bfin_bundle\Course_Mate
```

Percentile Calculation

Percentile finds a value that is equal or below a certain measure.

To calculate a percentile: np.percentile(dataset_variable, desired_percentile)

```
In [17]: # Must be turned into percentages
pop = pop * 100
```

```
In [18]: np.percentile(pop, 50)
```

```
Out[18]: 18.158807954190955
```

```
In [19]: np.median(pop)
```

```
Out[19]: 18.158807954190955
```

```
In [20]: np.percentile(pop, 5)
```

```
Out[20]: -14.45033274497672
```

```
In [21]: np.percentile(pop, 95)
```

```
Out[21]: 65.84576599196505
```

A Range of percentiles can be made by using a list

```
In [23]: np.percentile(pop, [25, 75])
```

```
Out[23]: array([ 4.21, 35.55])
```

```
In [22]: np.percentile(pop, [5, 95])
```

```
Out[22]: array([-14.45, 65.85])
```

```
In [24]: np.percentile(pop, [2.5, 97.5])
```

```
Out[24]: array([-23.42, 81.1])
```

New Section

Skew Refers to the Extent which a Distribution is not Symmetrical

Normal Distribution mean, median, and mode are the same.

Left Skew mean is less than the median and mode.

Right Skew means that the mean is greater than the median and the mode. **Easy Way to Determine Skew** see which tail is the longest.

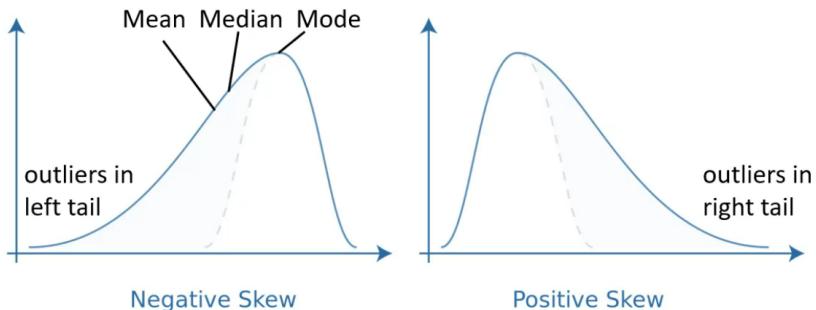
By Rodolfo Hermans (Godot) at en.wikipedia. - Own work; transferred from en.wikipedia by Rodolfo Hermans (Godot), CC BY-NC-SA 3.0. <https://commons.wikimedia.org/w/index.php?curid=4567445>

Higher Central Moments – skew

The **skew(ness)** refers to the extent to which a distribution is not symmetrical.

$$skew = \frac{1}{N} * \frac{\sum_{i=1}^N (X_i - \mu)^3}{\sigma^3}$$

σ : standard deviation
 μ : mean
 X_i : value of observation i
 N : total number of observations



Kurtosis.

Kurtosis is a measure of the degree to which a distribution is less "peaked" with "fat tails".

New Section

How to calculate Skew & Kurtosis with scipy

In [27]:

```
import numpy as np
import matplotlib.pyplot as plt
np.set_printoptions(precision = 2, suppress = True)
```

In [28]:

```
pop = np.loadtxt(r"C:\Users\alonz\Documents\Course_Materials_Part2_bfin_bundle\Course_Mate
delimiter = ", ", usecols = 1)
```

In [29]:

```
# Get percentages.
pop = pop * 100
```

In [34]:

```
# Preview fewer values to take up less space.
pop[0:10]
```

Out[34]:

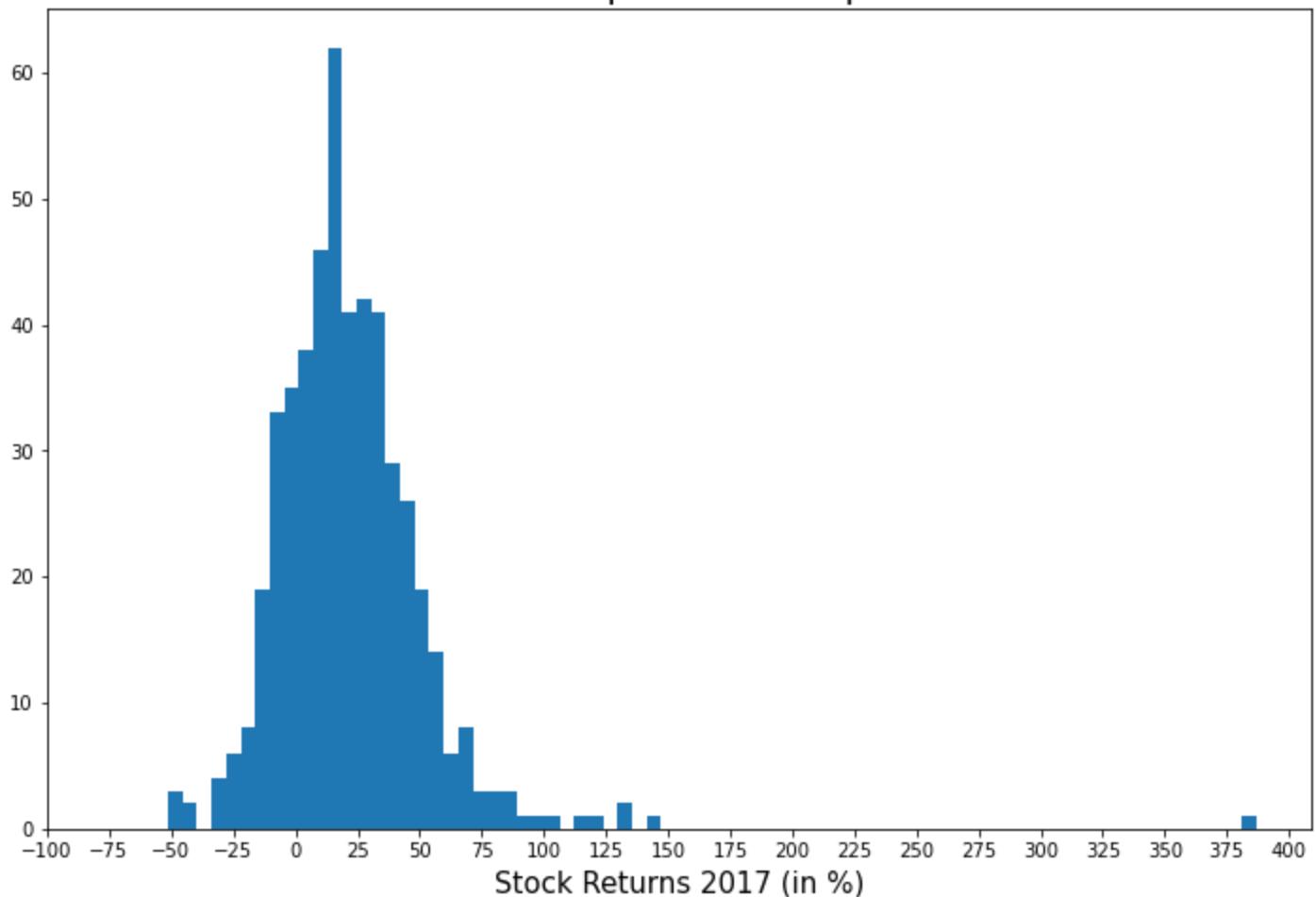
```
array([ 46.99,  11.44, -41.05,  46.11,  54.44,  17.43,  66.32,  48.58,
       30.7 ,  70.22])
```

In [33]:

```
plt.figure(figsize = (12, 8))
plt.hist(pop, bins = 75)
plt.title("Absolute Frequencies - Population", fontsize = 20)
plt.xlabel("Stock Returns 2017 (in %)", fontsize = 15)
```

```
plt.xticks(np.arange(-100, 401, 25))  
plt.show()
```

Absolute Frequencies - Population



Import scipy.stats to tell skew and Kurtosis

```
In [40]: import scipy.stats as stats
```

Skew

For **normal distribution** the **skew is about 0**. This shows that there is a right skew.
If the skew was negative the skew would be negative.

```
stats.skew(dataset)
```

```
In [46]: stats.skew(pop)
```

```
Out[46]: 3.69215566312915
```

It makes sense for the stock return skew to be right skewed. The stock can only lose a maximum (or gain/sustain) a -100% drop. A stock's gain is technically unlimited on the upside.

Kurtosis

The kurtosis setting has two different options. These are Fisher and Pearson. Press shift + tab for more info.

```
In [47]: # If normal the stat kurtosis would give zero.  
stats.kurtosis(pop, fisher = True)
```

```
Out[47]: 38.01022437524736
```

```
In [48]: stats.kurtosis(pop, fisher = False)
```

```
Out[48]: 41.01022437524736
```

Conclusion the difference is three so there are fat tails.

More information on Kurtosis: <https://www.investopedia.com/terms/k/kurtosis.asp>

New Section

Common Probability Distributions and Confidence Intervals

```
In [49]: import numpy as np
```

For `numpy.randint(size = int)`

You can pick how many random integers you want and create a random data set.

```
In [54]: np.random.randint(low = 1, high = 11, size = 10)
```

```
Out[54]: array([ 2,  3,  6,  2, 10,  6,  9,  7,  3,  5])
```

`random.random(size = int)` means not nec normal

```
In [55]: np.random.random(size = 10)
```

```
Out[55]: array([0.72, 0.96, 0.96, 0.51, 0.38, 0.93, 0.31, 0.07, 0.81, 0.37])
```

`random.uniform(size = int)` means that distribution is normal

```
In [60]: # The low is included the high is excluded.  
# Each number in the interval has an equal likelihood of being drawn.  
np.random.uniform(low = 1, high = 10, size = 10)
```

```
Out[60]: array([1.83, 7.59, 9.3 , 2.97, 3.45, 2.08, 3.71, 9.79, 6.36, 7.41])
```

```
In [57]: np.random.normal(size = 10)
```

```
Out[57]: array([-0.42, 0.96, 0.35, -0.8 , 1.22, 0.26, 0.5 , 0.84, -0.31, 0.08])
```

```
In [58]: np.random.normal(loc = 100, scale = 10, size = 10)
```

```
Out[58]: array([101.94, 93.74, 104.5 , 103.13, 105.92, 92. , 89.86, 98.47, 116.01, 90.03])
```

tab after random. ie. randomDOT

There are many options including binomial and chi-squared. That can be done.

Reproducibility with np.random.seed(int) before random.randint()

random.seed() allows randomized values to be reproducible by having their settings give a distinct value.

See that random.randint() gives the same output with random.seed(int)

```
In [70]: import numpy as np
```

```
In [71]: np.random.seed(123)
np.random.randint(low = 1, high = 11, size = 10)
```

```
Out[71]: array([ 3,  3,  7,  2,  4, 10,  7,  2,  1,  2])
```

```
In [72]: np.random.seed(123)
np.random.randint(low = 1, high = 11, size = 10)
```

```
Out[72]: array([ 3,  3,  7,  2,  4, 10,  7,  2,  1,  2])
```

```
In [73]: np.random.seed(123)
np.random.randint(low = 1, high = 11, size = 10)
```

```
Out[73]: array([ 3,  3,  7,  2,  4, 10,  7,  2,  1,  2])
```

Changing the random seed, with reproduced results.

```
In [74]: np.random.seed(5)
np.random.randint(low = 1, high = 11, size = 10)
```

```
Out[74]: array([ 4,  7,  7,  1, 10,  9,  5,  8,  1,  1])
```

```
In [75]: np.random.seed(5)
np.random.randint(low = 1, high = 11, size = 10)
```

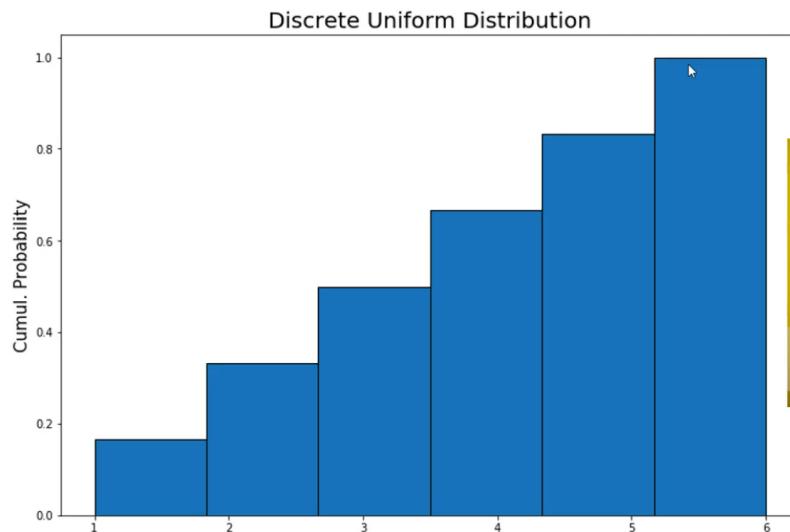
```
Out[75]: array([ 4,  7,  7,  1, 10,  9,  5,  8,  1,  1])
```

Types of Distributions

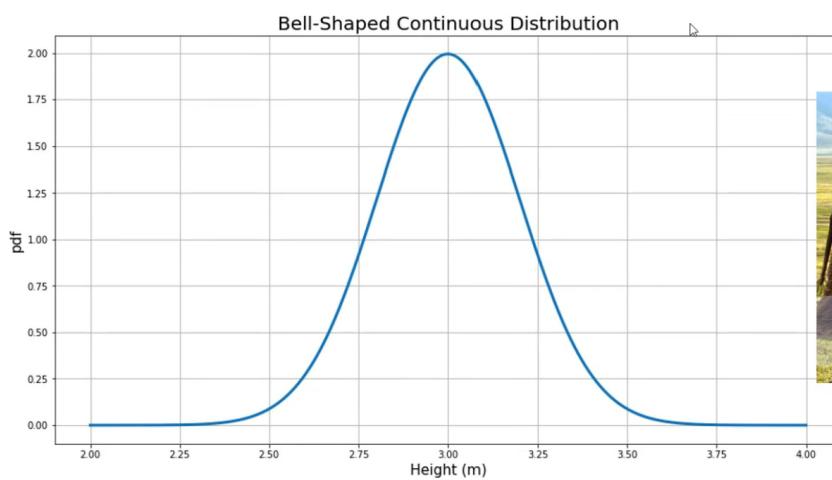
Discrete distributions are easily mapped and culmalatively equal one.

Culmalative distributions have a bell shaped distribution. The area under the bell curve is equal to one.

Discrete Uniform Distributions (cumulative)

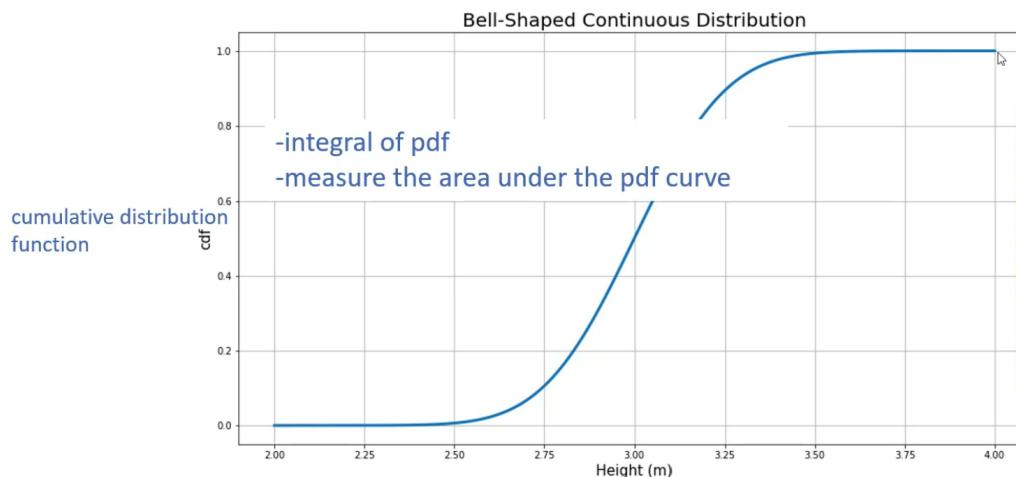


Continuous “bell-shaped” Distributions



Culmalative Continuous with Calculus

Continuous “bell-shaped” Distributions (cumulative)



Discrete Uniform Distributions

In [76]:

```
import numpy as np
import matplotlib.pyplot as plt
```

Simulating dice rolls.

In [77]:

```
np.random.randint(low = 1, high = 7, size = 10)
```

Out[77]:

```
array([5, 2, 6, 1, 4, 5, 6, 4, 2, 5])
```

In [121...]

```
np.random.seed(123)
a = np.random.randint(low = 1, high = 7, size = 1000000)
```

In [122...]

```
a.mean()
```

Out[122...]

```
3.500503
```

In [123...]

```
a.std()
```

Out[123...]

```
1.707455928271942
```

In [124...]

```
100000 / 6
```

Out[124...]

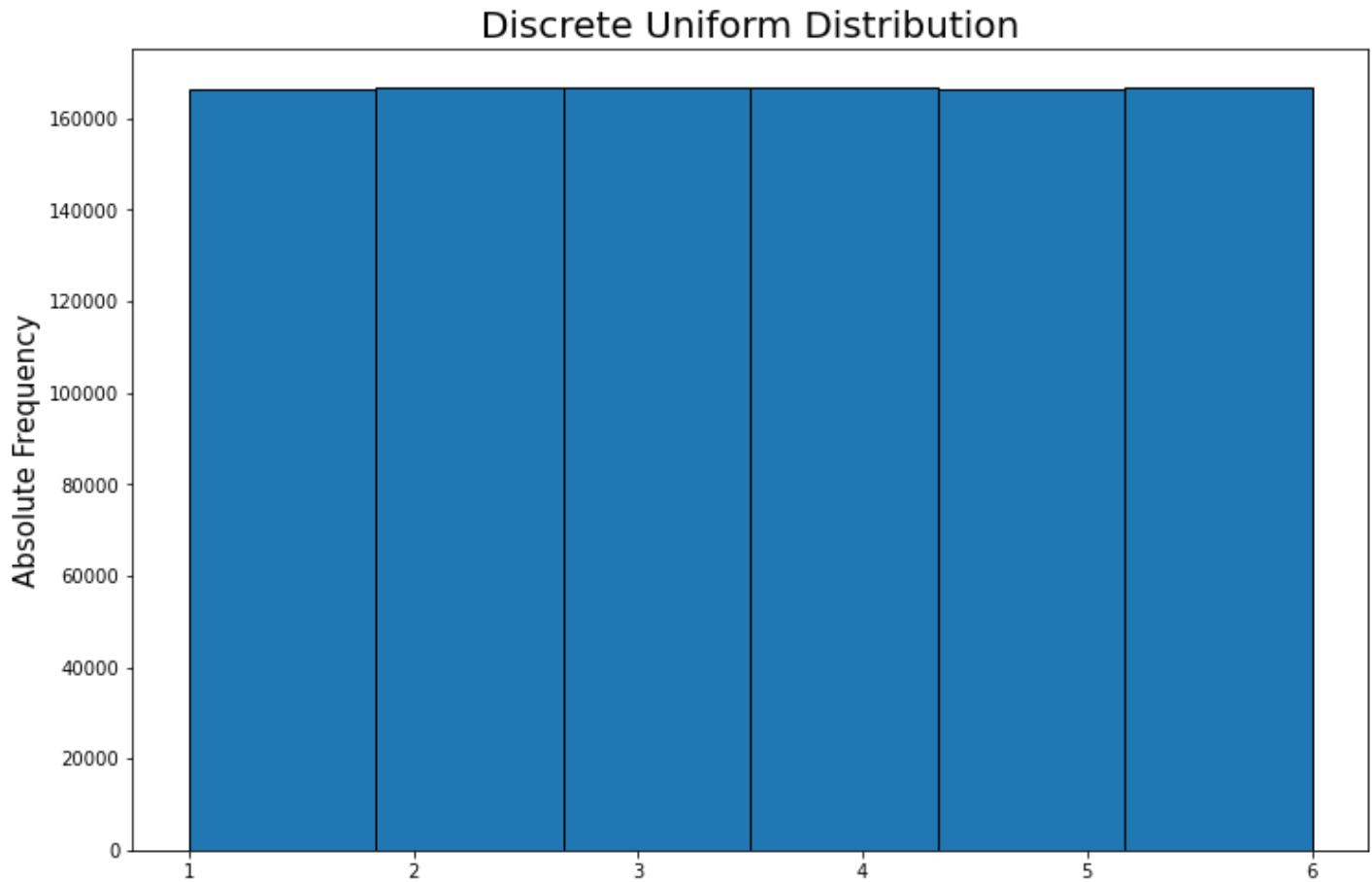
```
16666.666666666668
```

With a large sample size it can be seen how rolling a dice has an equal distribution.

In [125...]

```
plt.figure(figsize = (12, 8))
plt.hist(a, bins = 6, ec = "black")
plt.title("Discrete Uniform Distribution", fontsize = 20)
```

```
plt.ylabel("Absolute Frequency", fontsize = 15)  
plt.show()
```

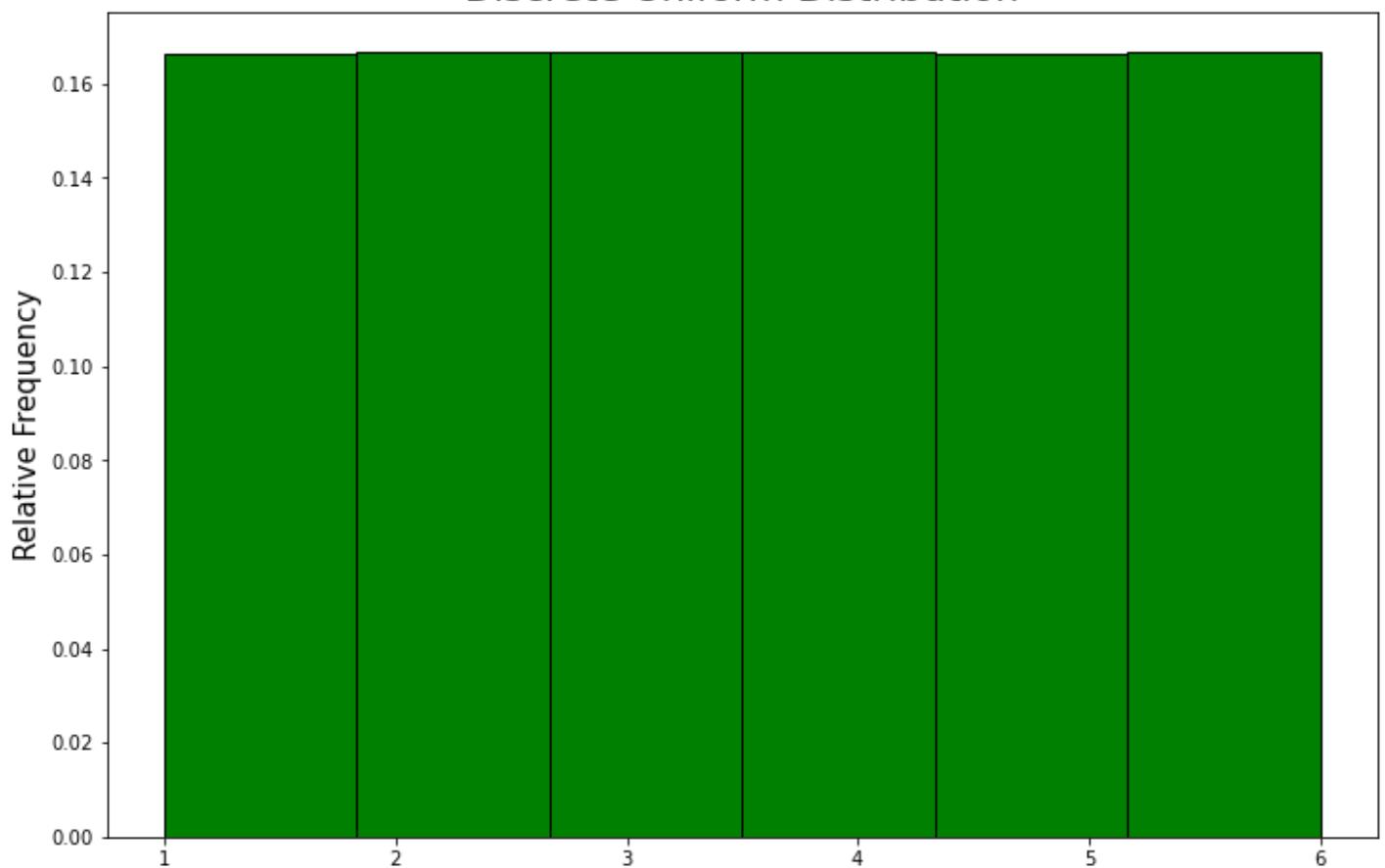


Correct Settings

In [126...]

```
# Correct Settings Discrete Distribution for random variable  
plt.figure(figsize = (12, 8))  
plt.hist(a, bins = 6, weights = np.ones(len(a)) / len(a), ec = "black", color = "green")  
plt.title("Discrete Uniform Distribution", fontsize = 20)  
plt.ylabel("Relative Frequency", fontsize = 15)  
plt.show()
```

Discrete Uniform Distribution

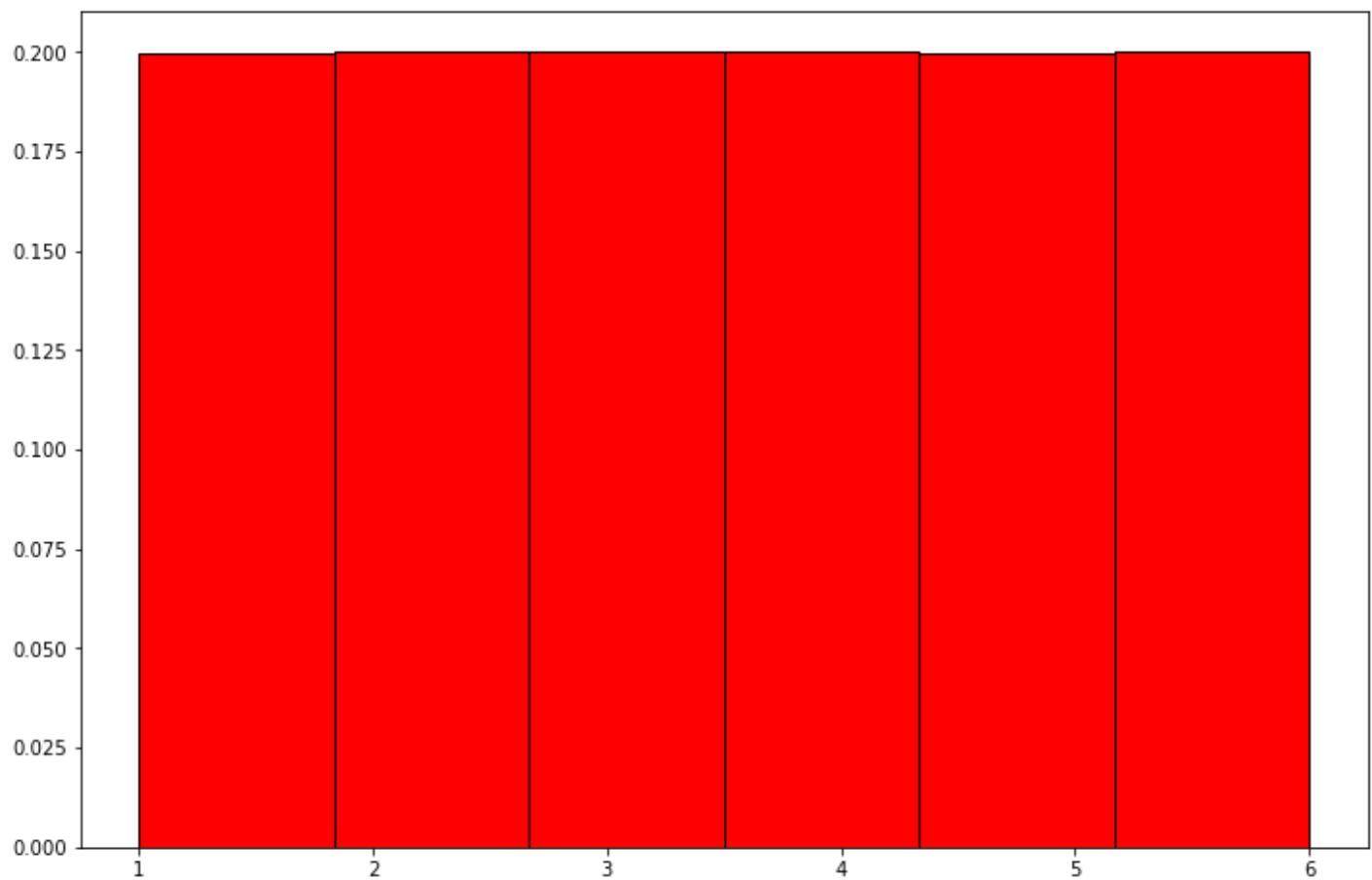


Below is incorrect. The density would have to be set to false etc. Culmalative would have to be set to True

In [127...]

```
# Below is incorrect.
plt.figure(figsize = (12, 8))
plt.hist(a, bins = 6, density = True, ec = "black", color = "red")
plt.title("Discrete Uniform Distribution", fontsize = 20)
plt.show()
```

Discrete Uniform Distribution



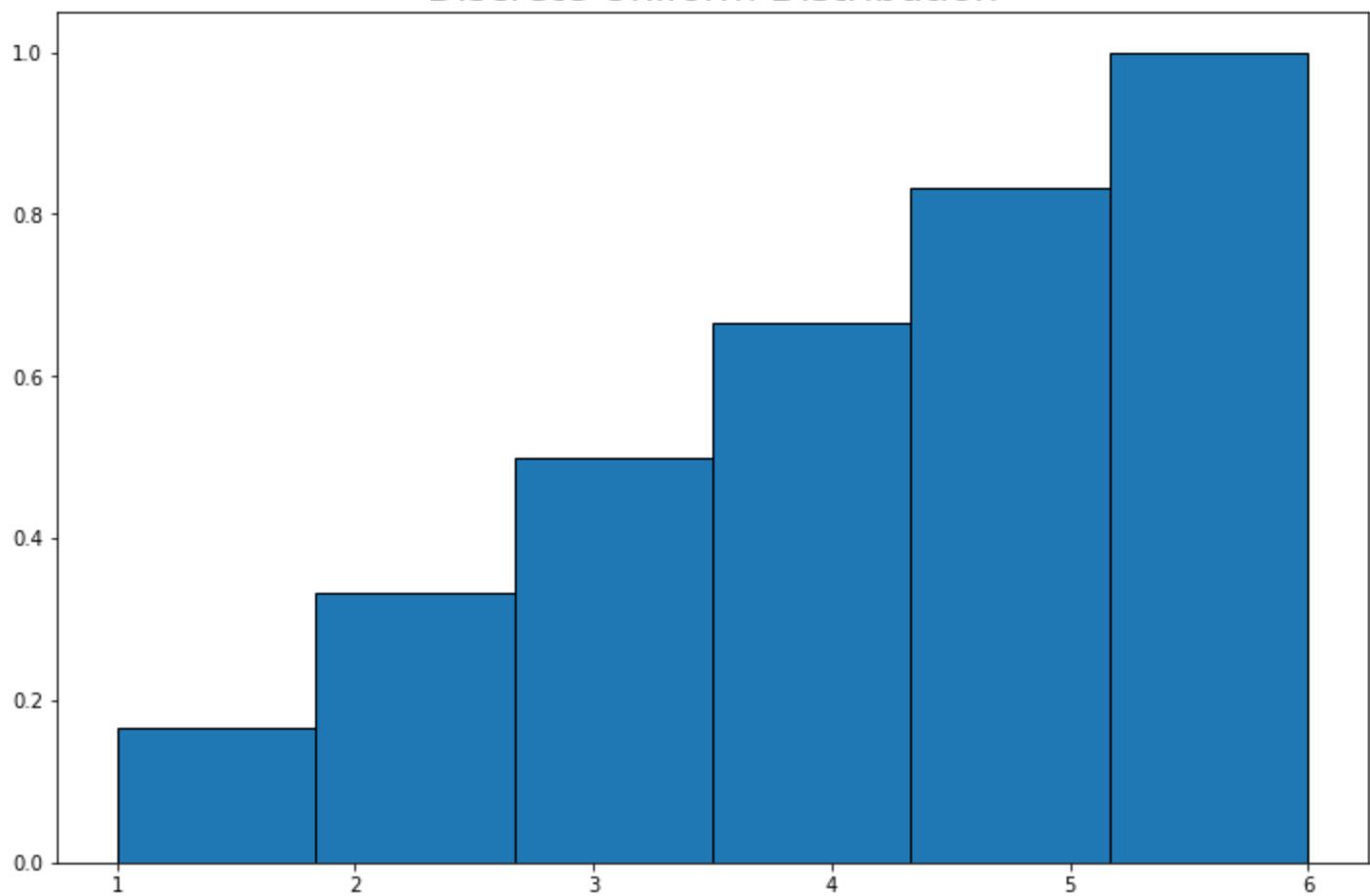
Discrete Uniform Distribution Correct Below

All of the values together are guaranteed to get 1 because it all of the options.

In [128...]

```
# This is correct
plt.figure(figsize = (12, 8))
plt.hist(a, bins = 6, density = True, cumulative = True, ec = "black")
plt.title("Discrete Uniform Distribution", fontsize = 20)
plt.show()
```

Discrete Uniform Distribution



New Section

Continuous Uniform Distribution

In [139...]

```
import numpy as np
import matplotlib.pyplot as plt
```

In [140...]

```
np.random.seed(123)
b = np.random.uniform(low = 0, high = 10, size = 1000000)
```

Use .size() to check the size

In [141...]

```
b.size
```

Out[141...]

1000000

In [142...]

```
b
```

Out[142...]

array([6.96, 2.86, 2.27, ..., 6.58, 4.58, 4.47])

In [143...]

```
b.mean()
```

Out[143...]

4.999334387281457

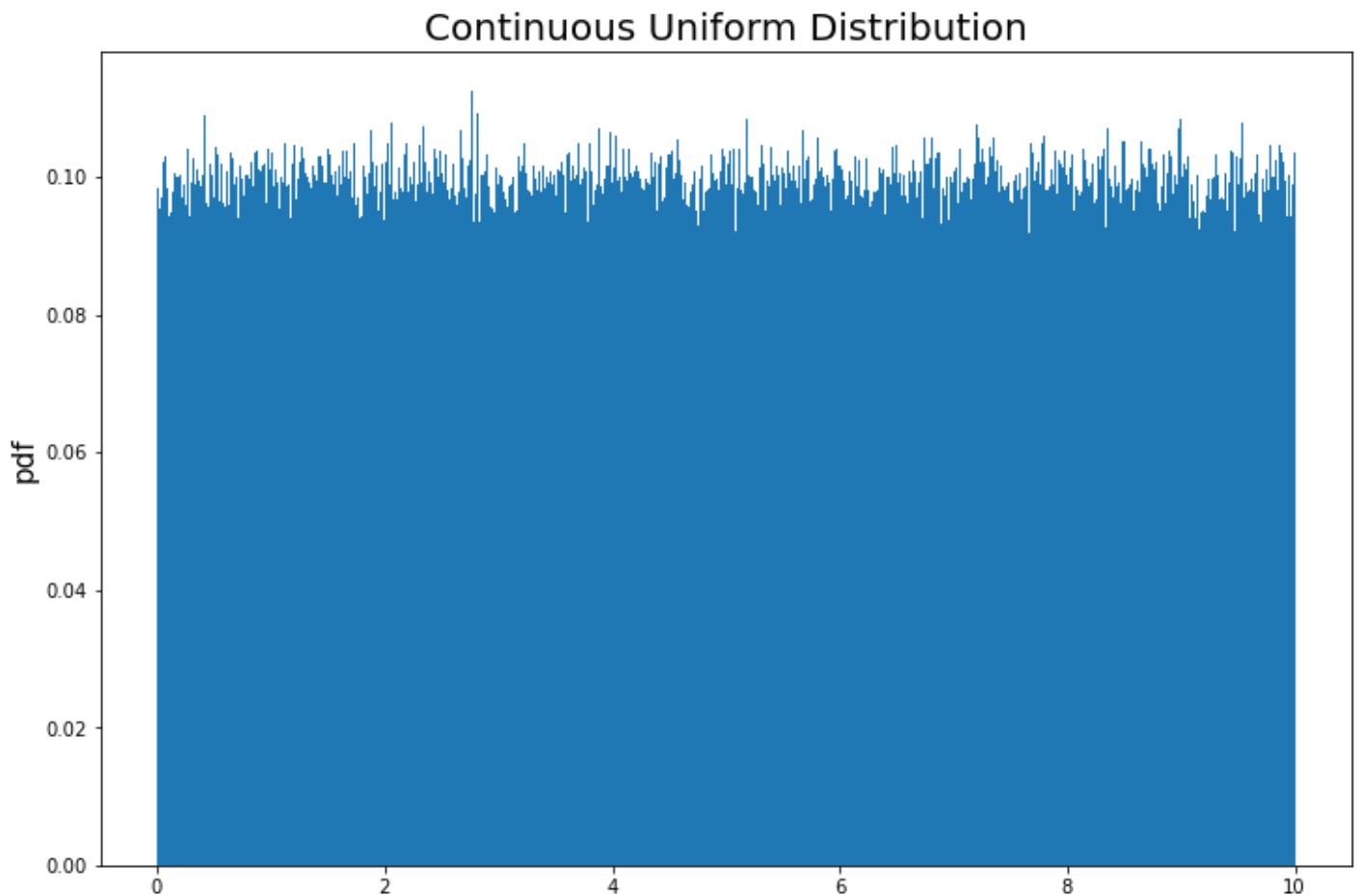
```
In [144...]: b.std()
```

```
Out[144...]: 2.8848661120984667
```

Plotting ten million data points

```
In [145...]:
```

```
# 10 * 0.01 = 1
plt.figure(figsize = (12, 8))
plt.hist(b, bins = 1000, density = True)
plt.title("Continuous Uniform Distribution", fontsize = 20)
plt.ylabel("pdf", fontsize = 15)
plt.show()
```

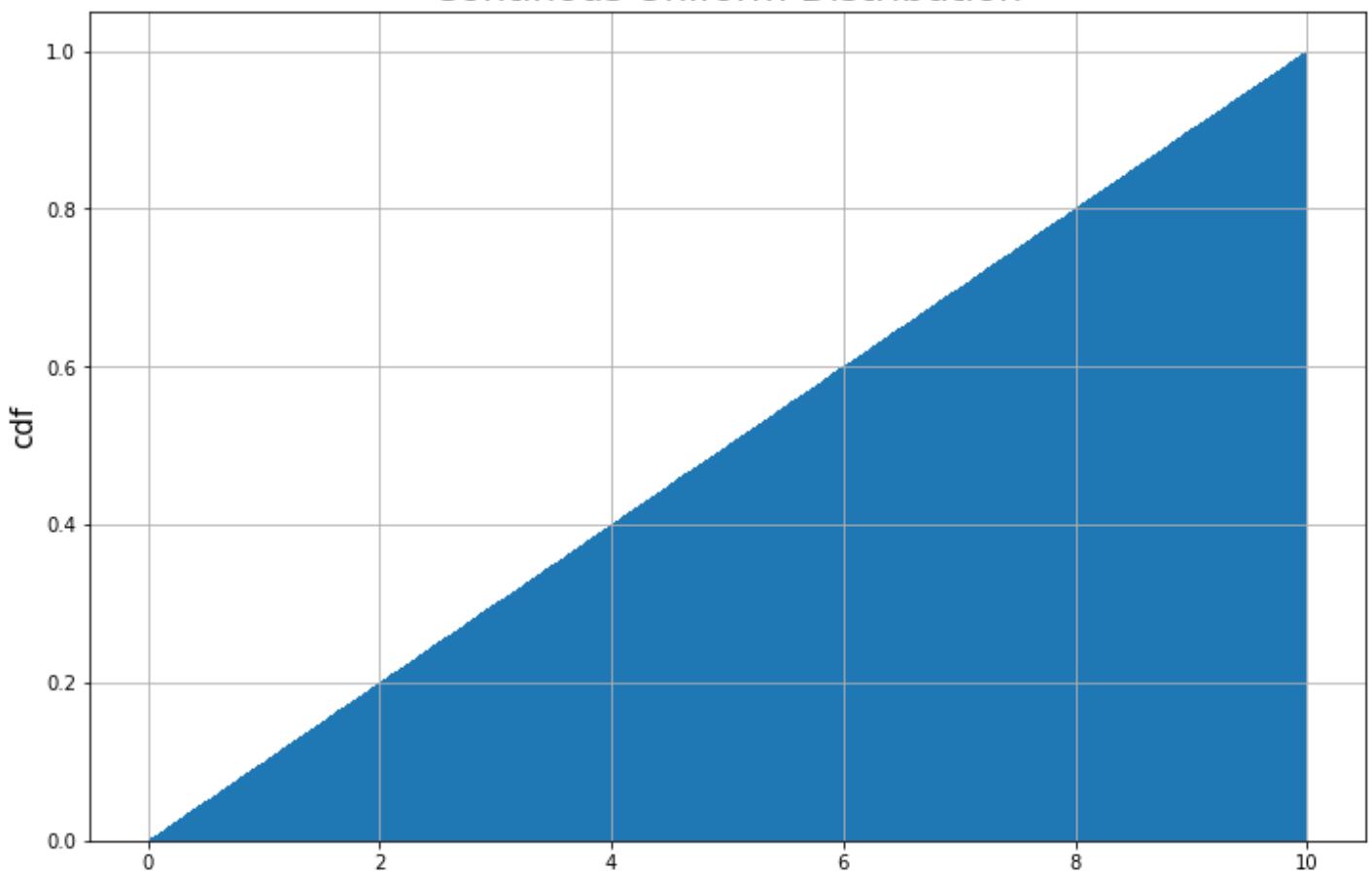


It looks this way because cumulative = True all of the values have to fall below a certain point.

```
In [146...]:
```

```
plt.figure(figsize = (12, 8))
plt.hist(b, bins = 1000, density = True, cumulative = True)
plt.grid()
plt.title("Continuous Uniform Distribution", fontsize = 20)
plt.ylabel("cdf", fontsize = 15)
plt.show()
```

Continuous Uniform Distribution



Normal Distribution

Normal distribution charts also mention the mean and the variance with the standard deviation.

It is completely described by the mean and standard deviation / variance.

Symmetrical around the mean and the skew is zero.

Kurtosis = 3 Excess Kurtosis = 0 | Kurtosis describes the height of the peak.

***The mean, median, and mode are at one point.**

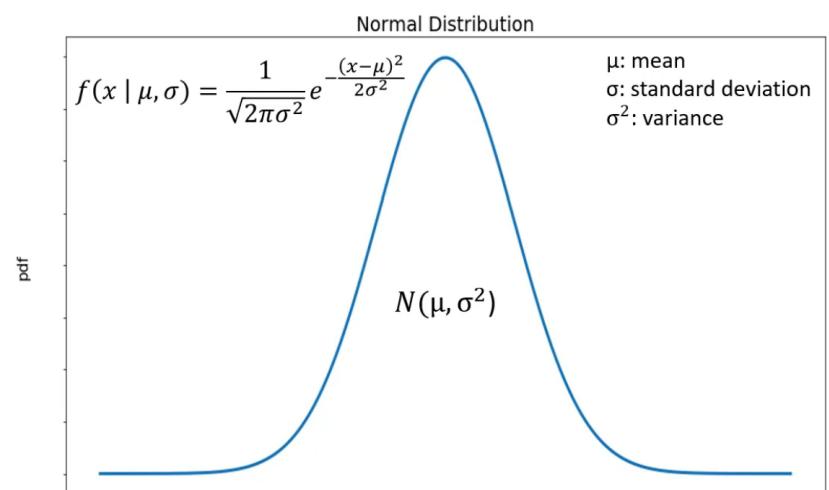
Theoretically the curve is infinite on both sides. There can always be a freakishly large item. It is impossible to know the maximum or the minimum.

Normal Distribution

The **Normal Distribution**:

Many random variables in **nature, science** and other fields (approx.) follow a Normal Distribution.

Properties are very useful for **inferential statistics** (Central Limit Theorem).

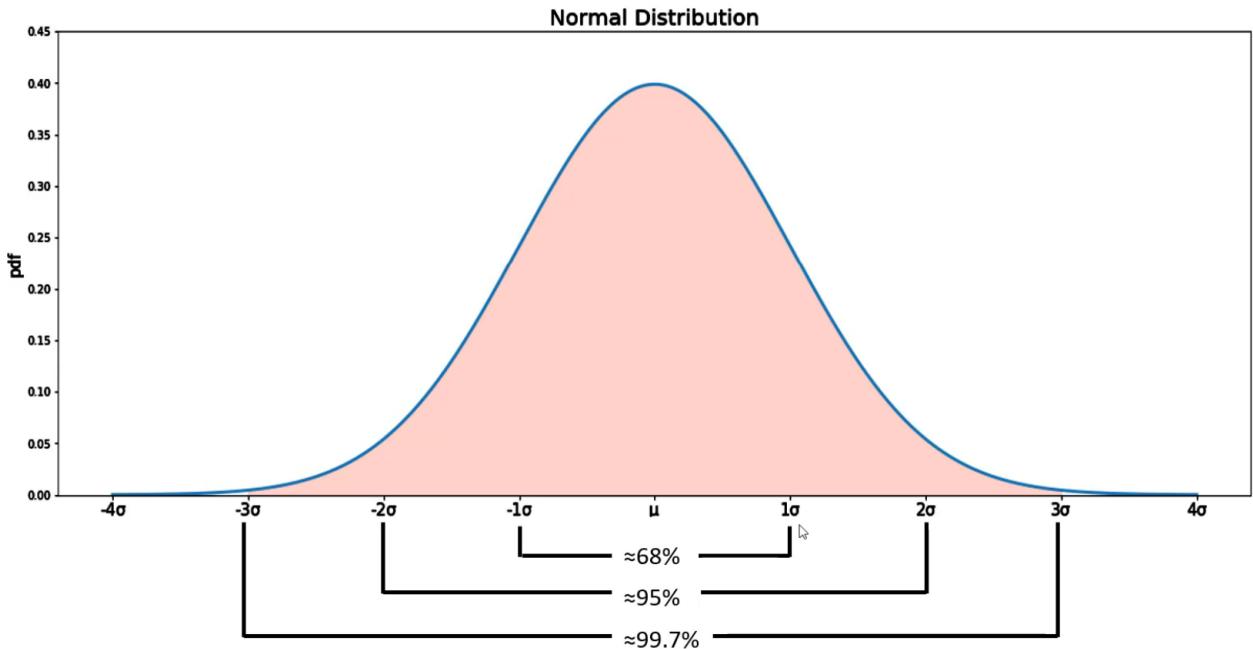


The 68, 95, 99% rule

This rule shows how the data is distributed. 1 standard deviation from the mean encompasses 68% of the data. Two standard deviations encompass 95% of the data. 3 standard deviations away encompasses 99% of the data.

Each std encompasses the positive and negative. For example 95% of the data falls between -2 std and +2 std.

Normal Distribution – the 68%, 95%, 99.7% rule



Creating normally distributed Random Variable

In [148...]

```
import numpy as np
import matplotlib.pyplot as plt
```

In [151...]

```
mu = 100
sigma = 2
size = 1000000
```

In [152...]

```
np.random.seed(123)
pop = np.random.normal(loc = mu, scale = sigma, size = size)
```

In [153...]

```
pop
```

Out[153...]

```
array([ 97.83, 101.99, 100.57, ..., 98.3 , 98.52, 97.41])
```

In [154...]

```
pop.mean()
```

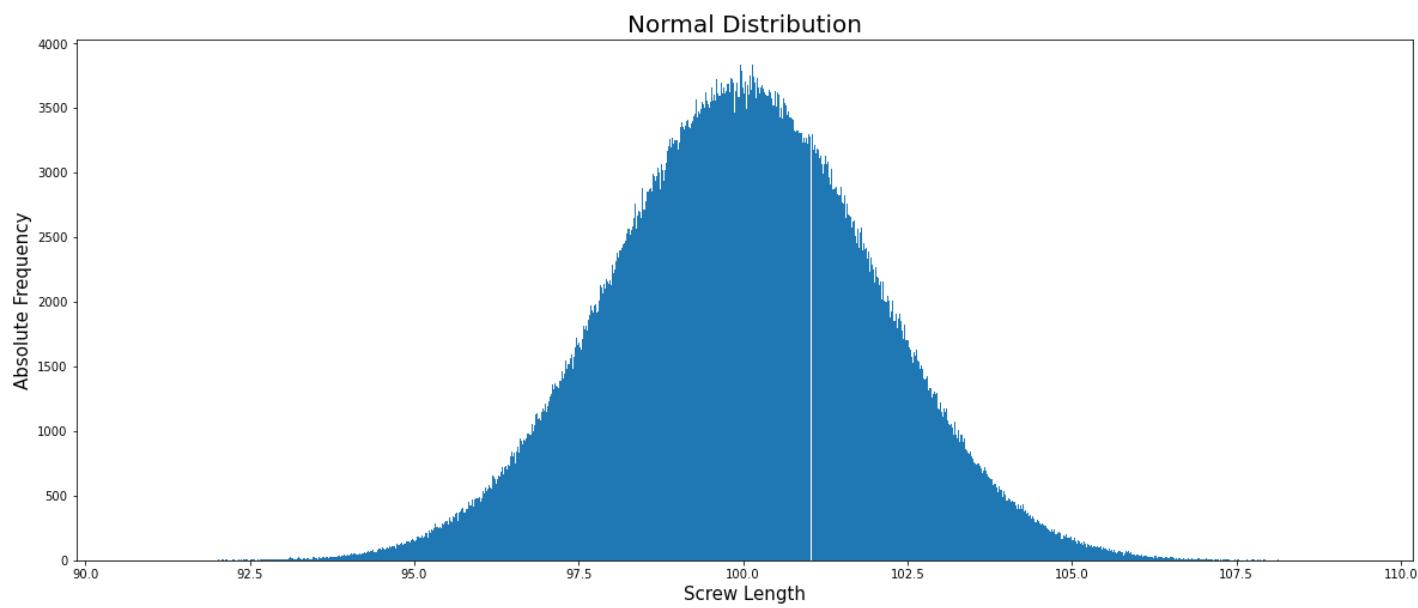
Out[154...]

```
100.00125900261469
```

In [155...]

```
plt.figure(figsize = (20, 8))
```

```
plt.hist(pop, bins = 1000)
plt.title("Normal Distribution", fontsize = 20)
plt.xlabel("Screw Length", fontsize = 15)
plt.ylabel("Absolute Frequency", fontsize = 15)
plt.show()
```



scipy to find skew and Kurtosis

```
In [159... import scipy.stats as stats
```

```
In [160... stats.skew(pop)
```

```
Out[160... 0.0002286155024783446
```

```
In [161... stats.kurtosis(pop)
```

```
Out[161... -0.005837588254301362
```

```
In [162... stats.kurtosis(pop, fisher = False)
```

```
Out[162... 2.9941624117456986
```

```
In [163... stats.describe(pop)
```

```
Out[163... DescribeResult(nobs=1000000, minmax=(90.78574017842989, 109.25515988525538), mean=100.00125900261469, variance=4.00207269828936, skewness=0.0002286155024783446, kurtosis=-0.005837588254301362)
```

Normal Distribution - Probability Density Function (pdf) with scipy.stats

```
In [164... import numpy as np
import scipy.stats as stats
import matplotlib.pyplot
```

```
In [166...]
```

```
mu = 100
sigma = 2
```

np.linspace() Creates evenly distributed numbers.

```
In [168...]
```

```
x = np.linspace(90, 110, 1000)
x[:40]
```

```
Out[168...]
```

```
array([90. , 90.02, 90.04, 90.06, 90.08, 90.1 , 90.12, 90.14, 90.16,
       90.18, 90.2 , 90.22, 90.24, 90.26, 90.28, 90.3 , 90.32, 90.34,
       90.36, 90.38, 90.4 , 90.42, 90.44, 90.46, 90.48, 90.5 , 90.52,
       90.54, 90.56, 90.58, 90.6 , 90.62, 90.64, 90.66, 90.68, 90.7 ,
       90.72, 90.74, 90.76, 90.78])
```

stats.norm.pdf means probability density function.

```
In [172...]
```

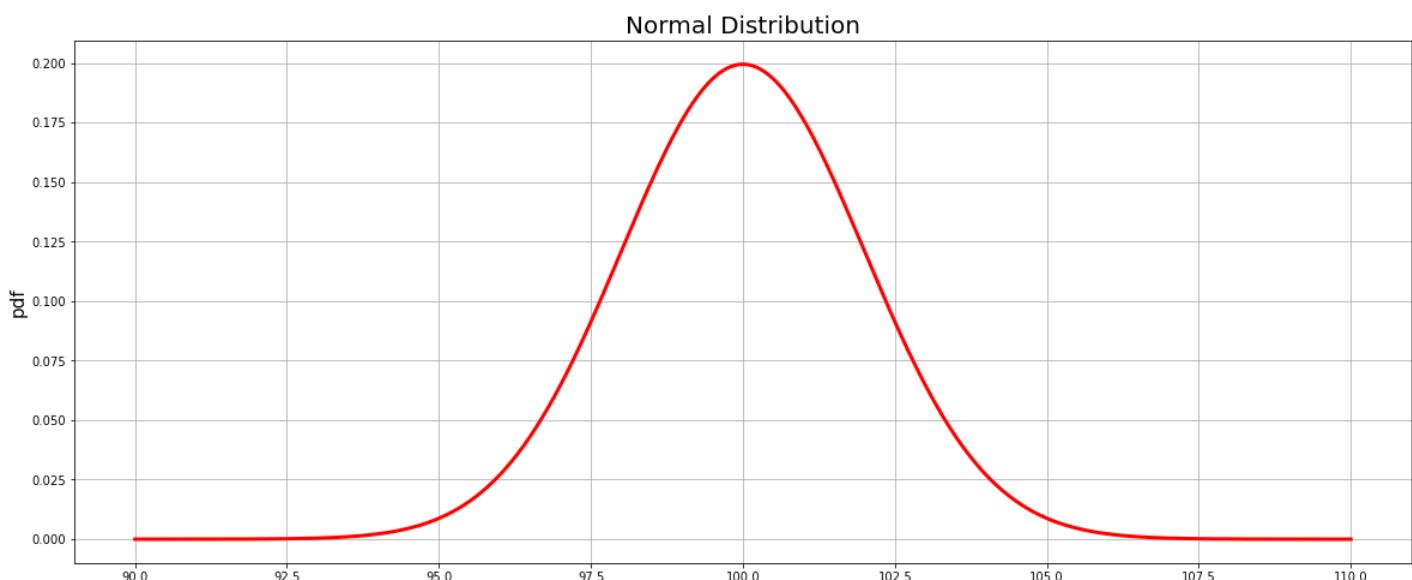
```
y = stats.norm.pdf(x, loc = mu, scale = sigma)
y[:10]
```

```
Out[172...]
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
In [173...]
```

```
plt.figure(figsize = (20, 8))
plt.plot(x, y, linewidth = 3, color = "red")
plt.grid()
plt.title("Normal Distribution", fontsize = 20)
plt.ylabel("pdf", fontsize = 15)
plt.show()
```



```
In [174...]
```

```
pop
```

```
Out[174...]
```

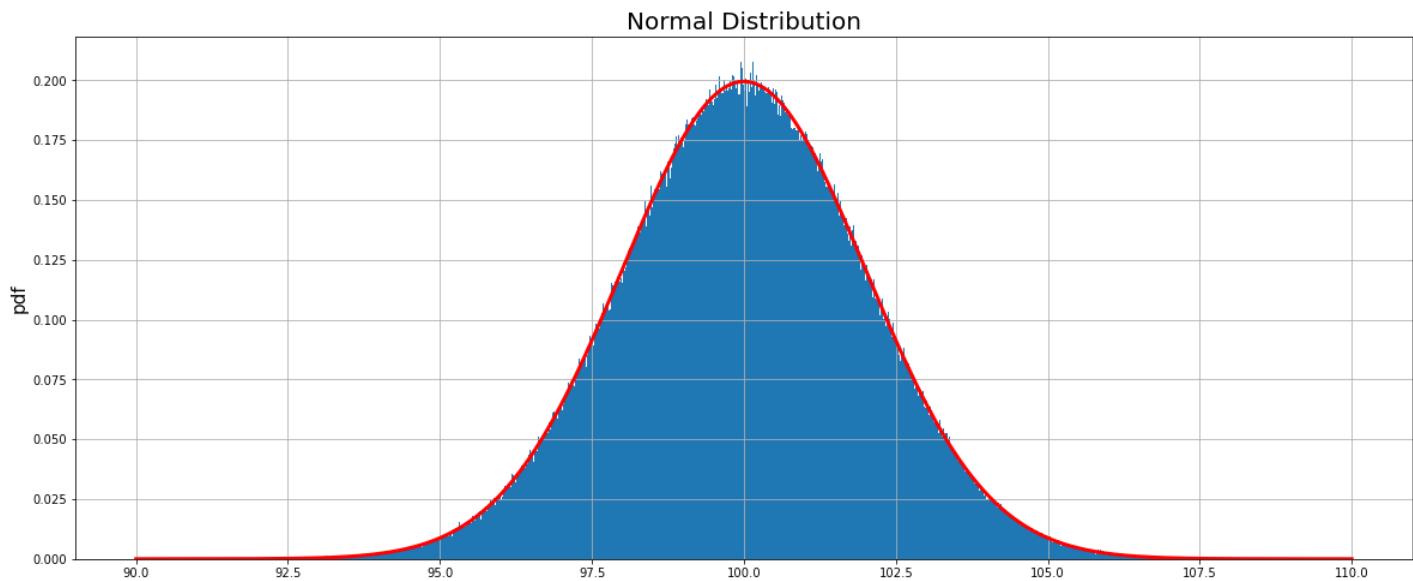
```
array([ 97.83, 101.99, 100.57, ..., 98.3 , 98.52, 97.41])
```

Plot a histogram on the graph.

```
In [175...]
```

```
plt.figure(figsize = (20, 8))
plt.hist(pop, bins = 1000, density = True)
plt.plot(x, y, linewidth = 3, color = "red")
plt.grid()
```

```
plt.title("Normal Distribution", fontsize = 20)
plt.ylabel("pdf", fontsize = 15)
plt.show()
```



Culmalative Distribution Function

In [184...]

```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
```

In [185...]

```
mu = 100
sigma = 2
```

In [186...]

```
x = np.linspace(90, 110, 1000)
```

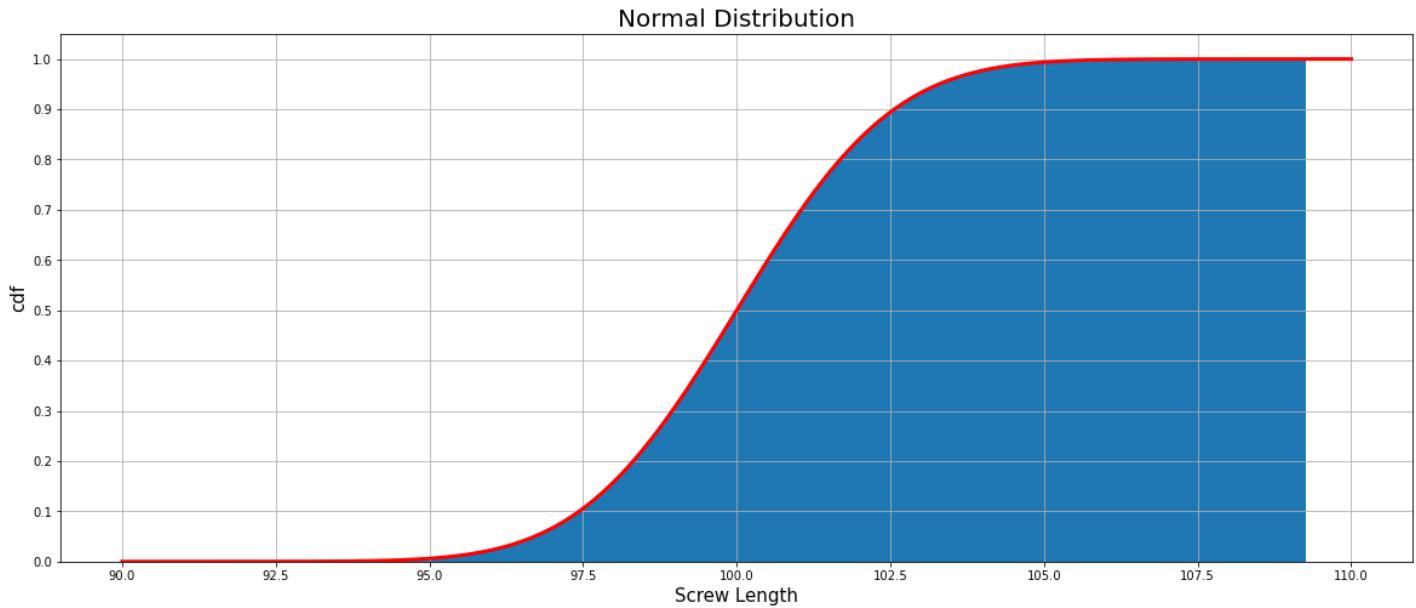
In [187...]

```
y = stats.norm.cdf(x, loc = mu, scale = sigma)
```

Cumulative Distribution Function with scripy.stats

In [190...]

```
plt.figure(figsize = (20, 8))
plt.hist(pop, bins = 1000, density = True, cumulative = True)
plt.plot(x, y, color = "red", linewidth = 3)
plt.grid()
plt.title("Normal Distribution", fontsize = 20)
plt.xlabel("Screw Length", fontsize = 15)
plt.ylabel("cdf", fontsize = 15)
plt.yticks(np.arange(0, 1.1, 0.1))
plt.show()
```



New Section

The Standard Normal Distribution and Z-Value

In [191...]

```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
```

In [192...]

```
pop
```

Out[192...]

```
array([ 97.83, 101.99, 100.57, ..., 98.3, 98.52, 97.41])
```

In [193...]

```
mu = pop.mean()
sigma = pop.std()
```

In [194...]

```
mu
```

Out[194...]

```
100.00125900261469
```

In [195...]

```
sigma
```

Out[195...]

```
2.0005171072042
```

In [196...]

```
pop[0]
```

Out[196...]

```
97.82873879339888
```

In [197...]

```
(pop[0] - mu) / sigma
```

Out[197...]

```
-1.085979320742718
```

In [198...]

```
pop[1]
```

```
Out[198... 101.99469089316717
```

```
In [199... (pop[1] - mu) / sigma
```

```
Out[199... 0.9964583073915245
```

```
In [200... (pop - mu) / sigma
```

```
Out[200... array([-1.09, 1., 0.28, ..., -0.85, -0.74, -1.29])
```

```
In [201... z = stats.zscore(pop)  
z
```

```
Out[201... array([-1.09, 1., 0.28, ..., -0.85, -0.74, -1.29])
```

```
In [202... round(z.mean(), 4)
```

```
Out[202... 0.0
```

```
In [203... z.std()
```

```
Out[203... 0.9999999999999999
```

```
In [204... stats.skew(z)
```

```
Out[204... 0.00022861550244412926
```

```
In [205... stats.kurtosis(z)
```

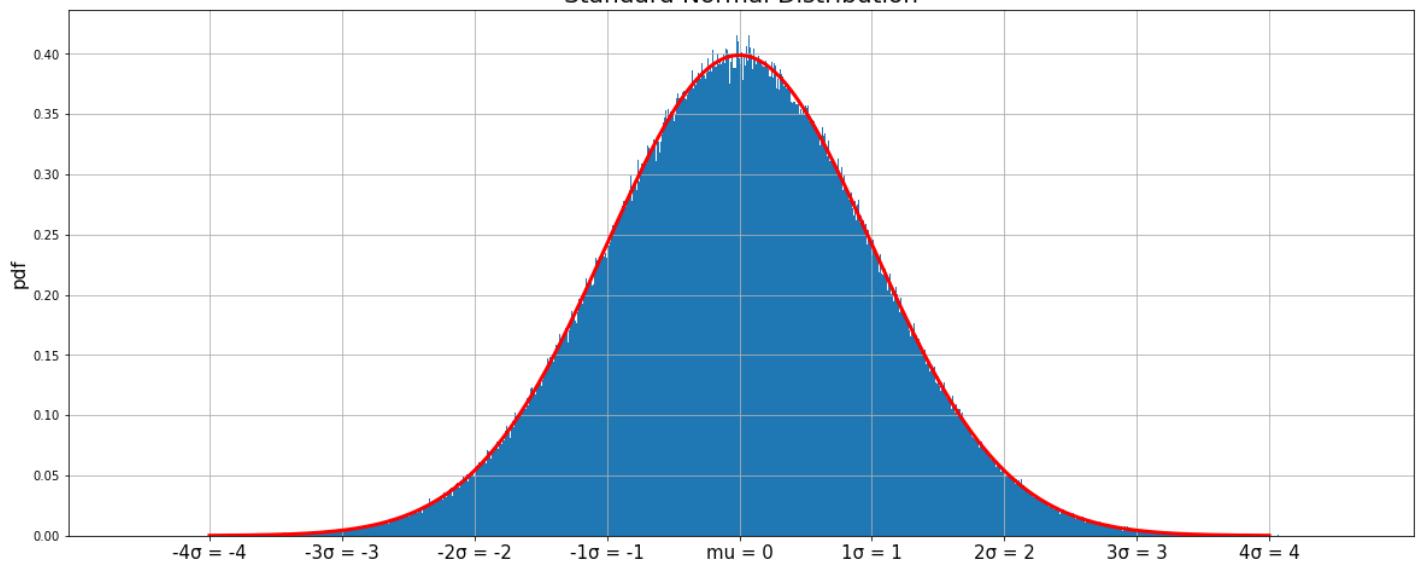
```
Out[205... -0.005837588254299586
```

```
In [206... x = np.linspace(-4, 4, 1000)
```

```
In [207... y = stats.norm.pdf(x, loc = 0, scale = 1)
```

```
In [211... plt.figure(figsize = (20, 8))  
plt.hist(z, bins = 1000, density = True)  
plt.grid()  
plt.plot(x, y, linewidth = 3, color = "red")  
plt.xticks(np.arange(-4, 5, 1),  
           labels = ["-4σ = -4", "-3σ = -3", "-2σ = -2", "-1σ = -1", "μ = 0", "1σ = 1", "2σ = 2", "3σ = 3"],  
           fontsize = 15)  
plt.title("Standard Normal Distribution", fontsize = 20)  
plt.ylabel("pdf", fontsize = 15)  
plt.show()
```

Standard Normal Distribution



Cumulative Version

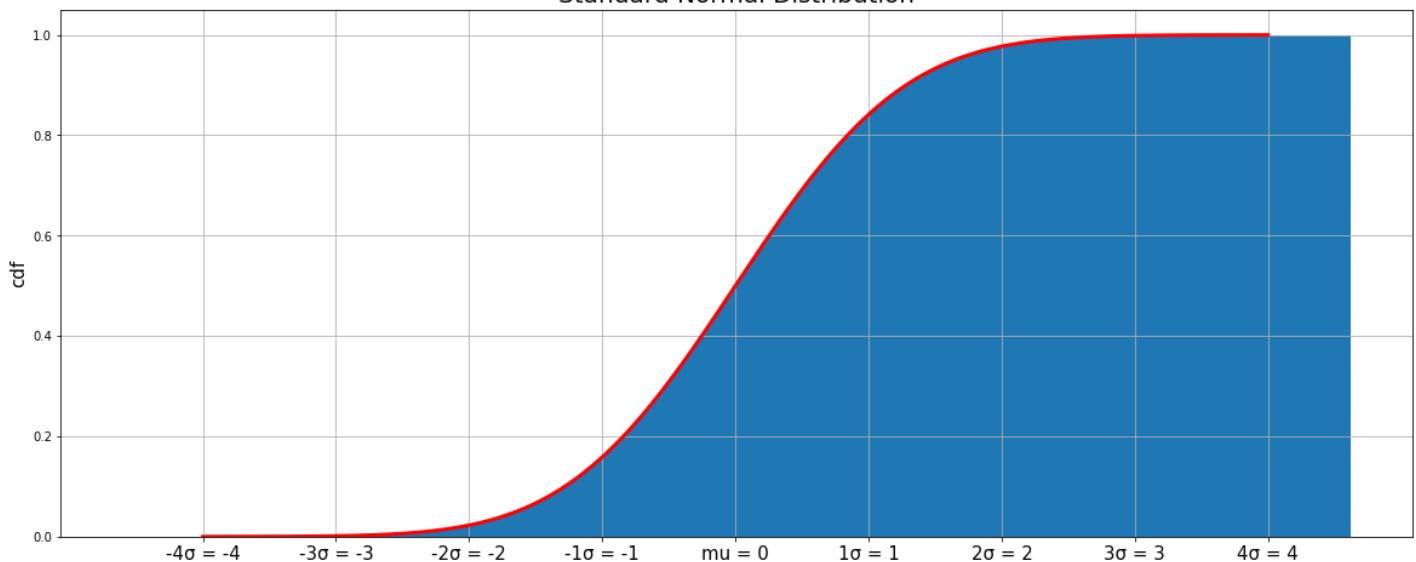
In [216]:

```
y = stats.norm.cdf(x)
```

In [219]:

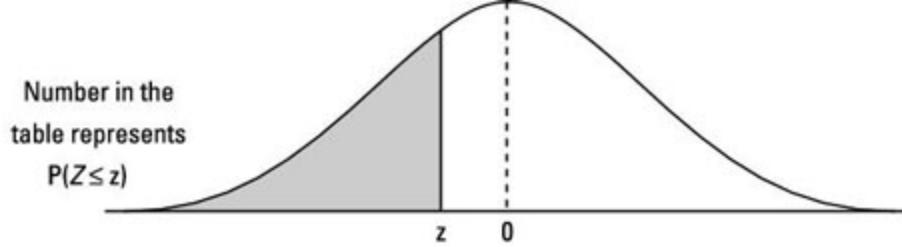
```
# Ran the code above
plt.figure(figsize = (20, 8))
#Change cumulative to true by adding the option.
plt.hist(z, bins = 1000, density = True, cumulative = True)
plt.grid()
plt.plot(x, y, linewidth = 3, color = "red")
plt.xticks(np.arange(-4, 5, 1),
           labels = ["-4σ = -4", "-3σ = -3", "-2σ = -2", "-1σ = -1", "mu = 0", "1σ = 1",
           fontsize = 15)
plt.title("Standard Normal Distribution", fontsize = 20)
#Changed y label from "pdf" to "cdf"
plt.ylabel("cdf", fontsize = 15)
plt.show()
```

Standard Normal Distribution



New Section

Probabilities and Z Score



z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
-3.6	.0002	.0002	.0001	.0001	.0001	.0001	.0001	.0001	.0001	.0001
-3.5	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0002
-3.4	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0003	.0002
-3.3	.0005	.0005	.0005	.0004	.0004	.0004	.0004	.0004	.0004	.0003
-3.2	.0007	.0007	.0006	.0006	.0006	.0006	.0006	.0005	.0005	.0005
-3.1	.0010	.0009	.0009	.0009	.0008	.0008	.0008	.0008	.0007	.0007
-3.0	.0013	.0013	.0013	.0012	.0012	.0011	.0011	.0011	.0010	.0010
-2.9	.0019	.0018	.0018	.0017	.0016	.0016	.0015	.0015	.0014	.0014
-2.8	.0026	.0025	.0024	.0023	.0023	.0022	.0021	.0021	.0020	.0019
-2.7	.0035	.0034	.0033	.0032	.0031	.0030	.0029	.0028	.0027	.0026
-2.6	.0047	.0045	.0044	.0043	.0041	.0040	.0039	.0038	.0037	.0036
-2.5	.0062	.0060	.0059	.0057	.0055	.0054	.0052	.0051	.0049	.0048
-2.4	.0082	.0080	.0078	.0075	.0073	.0071	.0069	.0068	.0066	.0064
-2.3	.0107	.0104	.0102	.0099	.0096	.0094	.0091	.0089	.0087	.0084
-2.2	.0139	.0136	.0132	.0129	.0125	.0122	.0119	.0116	.0113	.0110
-2.1	.0179	.0174	.0170	.0166	.0162	.0158	.0154	.0150	.0146	.0143
-2.0	.0228	.0222	.0217	.0212	.0207	.0202	.0197	.0192	.0188	.0183
-1.9	.0287	.0281	.0274	.0268	.0262	.0256	.0250	.0244	.0239	.0233
-1.8	.0359	.0351	.0344	.0336	.0329	.0322	.0314	.0307	.0301	.0294
-1.7	.0446	.0436	.0427	.0418	.0409	.0401	.0392	.0384	.0375	.0367
-1.6	.0548	.0537	.0526	.0516	.0505	.0495	.0485	.0475	.0465	.0455
-1.5	.0668	.0655	.0643	.0630	.0618	.0606	.0594	.0582	.0571	.0559
-1.4	.0808	.0793	.0778	.0764	.0749	.0735	.0721	.0708	.0694	.0681
-1.3	.0968	.0951	.0934	.0918	.0901	.0885	.0869	.0853	.0838	.0823
-1.2	.1151	.1131	.1112	.1093	.1075	.1056	.1038	.1020	.1003	.0985
-1.1	.1357	.1335	.1314	.1292	.1271	.1251	.1230	.1210	.1190	.1170
-1.0	.1587	.1562	.1539	.1515	.1492	.1469	.1446	.1423	.1401	.1379
-0.9	.1841	.1814	.1788	.1762	.1736	.1711	.1685	.1660	.1635	.1611
-0.8	.2119	.2090	.2061	.2033	.2005	.1977	.1949	.1922	.1894	.1867
-0.7	.2420	.2389	.2358	.2327	.2296	.2266	.2236	.2206	.2177	.2148
-0.6	.2743	.2709	.2676	.2643	.2611	.2578	.2546	.2514	.2483	.2451
-0.5	.3085	.3050	.3015	.2981	.2946	.2912	.2877	.2843	.2810	.2776
-0.4	.3446	.3409	.3372	.3336	.3300	.3264	.3228	.3192	.3156	.3121
-0.3	.3821	.3783	.3745	.3707	.3669	.3632	.3594	.3557	.3520	.3483
-0.2	.4207	.4168	.4129	.4090	.4052	.4013	.3974	.3936	.3897	.3859
-0.1	.4602	.4562	.4522	.4483	.4443	.4404	.4364	.4325	.4286	.4247
-0.0	.5000	.4960	.4920	.4880	.4840	.4801	.4761	.4721	.4681	.4641

In [220]:

```
import numpy as np
import scipy.stats as stats
```

In [221]:

```
stats.norm.cdf(-1, loc = 0, scale = 1)
```

```
Out[221... 0.15865525393145707
```

```
In [223... 1 - stats.norm.cdf(-1)
```

```
Out[223... 0.8413447460685429
```

```
In [224... stats.norm.cdf(1)
```

```
Out[224... 0.8413447460685429
```

```
In [225... 1 - stats.norm.cdf(1)
```

```
Out[225... 0.15865525393145707
```

```
In [226... stats.norm.cdf(1) - stats.norm.cdf(-1)
```

```
Out[226... 0.6826894921370859
```

```
In [227... stats.norm(-2)
```

```
Out[227... <scipy.stats._distn_infrastructure.rv_frozen at 0x1ca8531d2e0>
```

```
In [228... 1 - stats.norm.cdf(2)
```

```
Out[228... 0.02275013194817921
```

```
In [229... stats.norm.cdf(2)
```

```
Out[229... 0.9772498680518208
```

```
In [230... stats.norm.cdf(2) - stats.norm.cdf(-2)
```

```
Out[230... 0.9544997361036416
```

```
In [231... stats.norm.cdf(0)
```

```
Out[231... 0.5
```

```
In [232... pop
```

```
Out[232... array([ 97.83, 101.99, 100.57, ..., 98.3 , 98.52, 97.41])
```

```
In [233... minus_two_sigma = pop.mean() - 2 * pop.std()  
minus_two_sigma
```

```
Out[233... 96.00022478820628
```

```
In [234...]: # Mean gives relative frequency.  
(pop < minus_two_sigma).mean()
```

```
Out[234...]: 0.022783
```

```
In [235...]: stats.norm.cdf(x = 105, loc = pop.mean(), scale = pop.std())
```

```
Out[235...]: 0.9937679406729488
```

```
In [236...]: z = (105 - pop.mean()) / pop.std()
```

```
In [237...]: stats.norm.cdf(z)
```

```
Out[237...]: 0.9937679406729488
```

Working from ppf to Z

```
In [238...]: stats.norm.ppf(0.5, loc = 0, scale = 1)
```

```
Out[238...]: 0.0
```

Z scores

```
In [241...]: stats.norm.ppf(0.05)
```

```
Out[241...]: -1.6448536269514729
```

```
In [240...]: stats.norm.ppf(0.95)
```

```
Out[240...]: 1.6448536269514722
```

5% of all of the scores are equal or less than 96.71

```
In [242...]: stats.norm.ppf(loc = pop.mean(), scale = pop.std(), q = 0.05)
```

```
Out[242...]: 96.71070118305138
```

```
In [243...]: stats.norm.ppf(loc = pop.mean(), scale = pop.std(), q = 0.95)
```

```
Out[243...]: 103.29181682217798
```

Confidence Intervals

```
In [244...]: import numpy as np  
import scipy.stats as stats  
import matplotlib.pyplot
```

The ABC Company produces screws. The length of the screws follows a **Normal Distribution** with **mean 100** (millimeters) and **standard deviation 2** (millimeters). Determine the **Confidence Interval** around the mean where **90%** of all observations can be found.

```
In [245...]: conf = 0.90
```

```
In [246...]: tails = (1 - conf) / 2
tails
```

```
Out[246...]: 0.04999999999999999
```

```
In [247...]: left = stats.norm.ppf(tails)
left
```

```
Out[247...]: -1.6448536269514729
```

```
In [248...]: right = stats.norm.ppf(1 - tails)
right
```

```
Out[248...]: 1.6448536269514722
```

```
In [249...]: stats.norm.interval(conf)
```

```
Out[249...]: (-1.6448536269514729, 1.6448536269514722)
```

```
In [250...]: left, right = stats.norm.interval(conf)
```

```
In [251...]: left
```

```
Out[251...]: -1.6448536269514729
```

```
In [252...]: right
```

```
Out[252...]: 1.6448536269514722
```

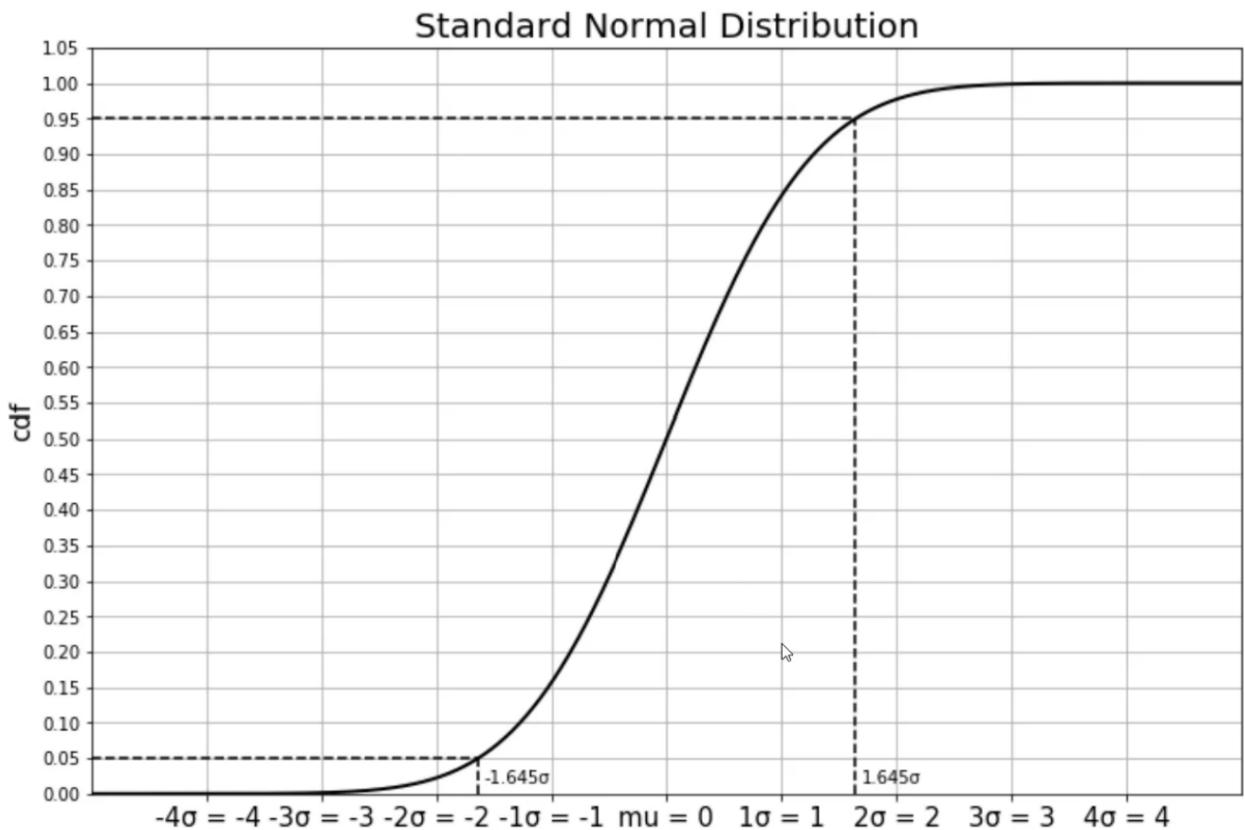
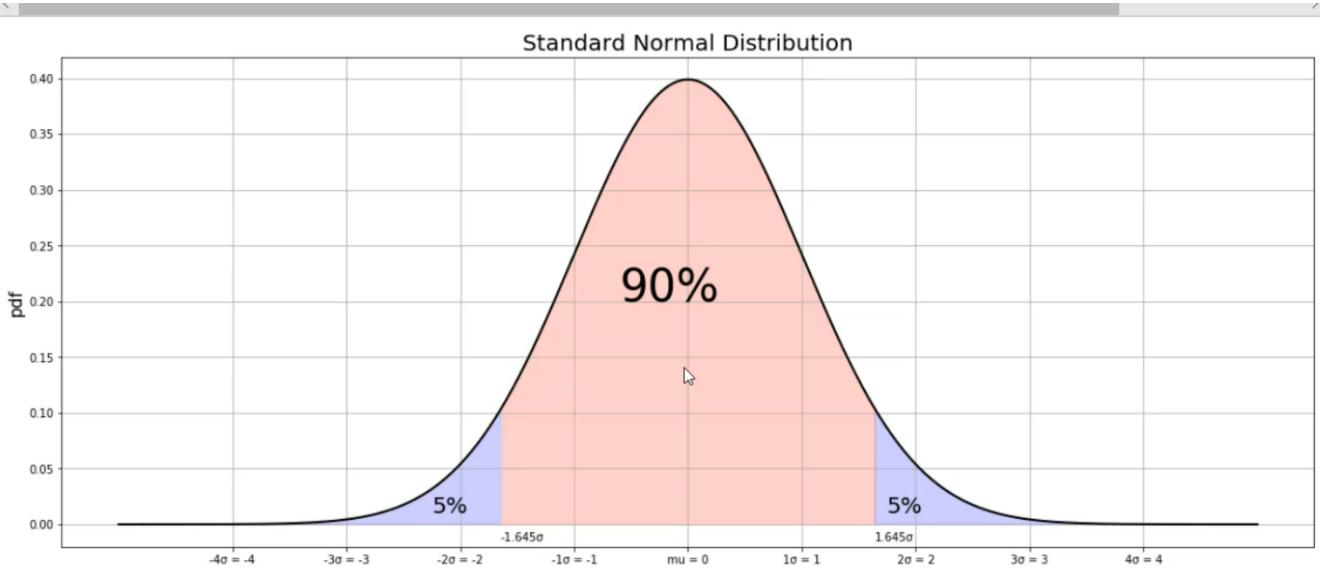
```
In [254...]: x = np.linspace(-5, 5, 1000)
```

```
In [255...]: y = stats.norm.pdf(x)
```

```

: plt.figure(figsize = (20, 8))
plt.plot(x, y, color = "black", linewidth = 2)
plt.fill_between(x, y, where = ((x > right) | (x < left)), color = "blue", alpha = 0.2)
plt.fill_between(x, y, where = ((x < right) & (x > left)), color = "red", alpha = 0.2)
plt.grid()
plt.annotate("5%", xy = (1.75, 0.01), fontsize = 20)
plt.annotate("5%", xy = (-2.25, 0.01), fontsize = 20)
plt.annotate("90%", xy = (-0.6, 0.2), fontsize = 40)
plt.annotate("-1.645 $\sigma$ ", xy = (-1.645, -0.015), fontsize = 10)
plt.annotate("1.645 $\sigma$ ", xy = (1.645, -0.015), fontsize = 10)
plt.xticks(np.arange(-4, 5, 1),
           labels = ["-4 $\sigma$  = -4", "-3 $\sigma$  = -3", "-2 $\sigma$  = -2", "-1 $\sigma$  = -1", "mu = 0", "1 $\sigma$  = 1", "2 $\sigma$  = 2", "3 $\sigma$  = 3", "4 $\sigma$  = 4"], fontsize = 10)
plt.title("Standard Normal Distribution", fontsize = 20)
plt.ylabel("pdf", fontsize = 15)
plt.show()

```



In []:

Defining User Defined Functions

In [1]:

```
pv = 100
r = 0.04
n = 5
```

In [4]:

```
# Note the multiplication sign is needed outside of parenthesis it is not like a calculator
fv = pv * (1 + r) ** n
```

In [9]:

```
# Run the actual formula.
fv = pv * (1 + r) ** n
fv
```

Out[9]:

```
121.66529024000002
```

Calculate the future value via a formula

Any numbers can be passed in by positional or defined arguments.

In [21]:

```
def future_value(pv, rate, nper):
    fv = pv * (1 + r) ** n
    return fv
```

In [22]:

```
# Can work with a keyword argument.
future_value(pv = 100, rate = 0.004, nper = 5)
```

Out[22]:

```
121.66529024000002
```

In [23]:

```
# Can work with positional arguments.
future_value(100, 0.004, 5)
```

Out[23]:

```
121.66529024000002
```

A combination of keyword and positional arguments can be used.

Once the first keyword argument is used the rest has to be keyword arguments.

In [24]:

```
# Combination no positional arguments after the key word.
future_value (100, rate = 0.04, 5)
```

Input In [24]

```
future_value (100, rate = 0.04, 5)
```

^

SyntaxError: positional argument follows keyword argument

In [25]:

```
# Combination - no positional arguments after the keyword argument.
future_value(100, rate = 0.04, nper = 5)
```

```
Out[25]: 121.66529024000002
```

Default values

Default values are values that are to be utilized in case the user does not specify a value.

Improved Function

An improved function can take into consideration that most Future value scenarios compound once per year meaning the number of compound periods within a year is one. However there are cases where there is quarterly compoundings etc.

```
In [27]: def future_value2(pv, rate, nyears, m):  
    fv = pv * (1 + rate/m) ** (nyears * m)  
    return fv
```

```
In [29]: # Annual compounding.  
future_value2(100, 0.04, 5, 1)
```

```
Out[29]: 121.66529024000002
```

```
In [30]: # Quarterly compounding. Four compound periods a year.  
future_value2(100, 0.04, 5, 4)
```

```
Out[30]: 122.0190039947967
```

```
In [31]: # Monthly compounding 12 compound periods within a year. Tracked for 5 years.  
future_value2(100, 0.04, 5, 12)
```

```
Out[31]: 122.09965939421214
```

For a default value define the value when the function is created.

```
In [34]: # Assumes Annual Compounding  
def future_value2(pv, rate, nyears, m = 1):  
    fv = pv * (1 + rate/m) ** (nyears * m)  
    return fv
```

```
In [35]: # Code that does not define m.  
# Still works because it is pre-determined upon its creation.  
future_value2(100, 0.04, 5, 4)
```

```
Out[35]: 122.0190039947967
```

Default Arguments must be defined last or it will get an error.

```
In [36]: # Assumes Annual Compounding  
def future_value2(pv, rate, m = 1, nyears):
```

```
fv = pv * (1 + rate/m) ** (nyears * m)
return fv
```

```
Input In [36]
def future_value2(pv, rate, m = 1 nyyears):
^
SyntaxError: invalid syntax
```

The Default argument None

Basically it says if m is not defined that the default value should be m = 1.

The only way to make m real is to give it a default value of nothing is to assign it a value.

```
In [55]: def future_value5(pv, rate, nyyears, m = None):
    # If m is a none object or is nothing pass in a value of m = 1
    if not m:
        m = 1

    fv = pv * (1 + rate/m) ** (nyyears * m)
    return fv
```

None has an output of nothing.

It returns a none type.

```
In [50]: a = None
```

```
In [51]: type(a)
```

```
Out[51]: NoneType
```

```
In [52]: # A boolean gives a false, because it is basically nothing.
bool(a)
```

```
Out[52]: False
```

```
In [53]: not a
```

```
Out[53]: True
```

```
In [54]: future_value5(100, 0.04, 5)
```

```
Out[54]: 121.66529024000002
```

```
In [58]: # Here the default value of m = 1 that comes from the if statement replaces the none type
# explicitly stating which
future_value5(100, 0.04, 5, 12)
```

```
Out[58]: 122.09965939421214
```

The none type is the best way to define an optional value. Hardcoding code

will make it remain the same based on what the variable (defined outside the function) was at the time of its creation.

The variable would have to be updated. Then the function would have to run again to update.

Unpacking Tuples

In [59]:

```
tup = (1, 2, 3, 4)
```

Tuple is basically a list that cannot be changed. Variables can be assigned to its individual constituents individually, respectively.

In [66]:

```
a, b, c, d = tup
```

Functions receive customized inputs via a tuple.

Look back at function 2.

In []:

```
def future_value2(pv, rate, nyears, m):  
    fv = pv * (1 + rate/m) ** (nyears * m)  
    return fv
```

In [67]:

```
future_value2(100, 0.04, 5, 12)
```

Out[67]:

```
122.09965939421214
```

In [68]:

```
tup = (100, 0.04, 5, 12)
```

In [69]:

```
tup
```

Out[69]:

```
(100, 0.04, 5, 12)
```

var_name,(asteric*) var_name2 = (x, y, z)

this allows for a tuple to be unpacked and stored as a list into one variable.

Note there must be several variables unpacking before the star variable can be used.

In []:

```
tup = (1, 2, 3, 4)
```

In [70]:

```
# Look at a vs *b  
a, *b = tup
```

In [71]:

```
a
```

Out[71]:

```
100
```

In [72]:

```
b
```

```
Out[72]: [0.04, 5, 12]
```

```
In [78]: tup2 = (7, 39, 68, 73)
```

```
In [83]: c, d, *e = tup2
```

```
In [84]: c
```

```
Out[84]: 7
```

```
In [85]: d
```

```
Out[85]: 39
```

```
In [86]: e
```

```
Out[86]: [68, 73]
```

Sequences as arguments and *args

```
In [87]: cf = [-200, 20, 50, 70, 100, 50]
```

```
In [88]: r = 0.06
```

```
In [89]: NPV = 0
for i in range(len(cf)):
    NPV += cf[i] / (1 + r) ** (i)
print(NPV)
```

```
38.71337130837991
```

```
In [92]: def npv(rate, values):
    NPV = 0
    for i in range(len(values)):
        NPV += values[i] / (1 + rate) ** (i)
    return NPV
```

```
In [93]: npv(rate = r, values = cf)
```

```
Out[93]: 38.71337130837991
```

For the positional argument to work we need to have a list for one of the arguments.

The rate does not change so that will be one individual entry. The values need to be put in a list.

```
In [99]: npv(r, -200, 20, 50, 70, 100, 50)
```

```
-----  
TypeError                                     Traceback (most recent call last)  
Input In [99], in <cell line: 1>()  
----> 1 npv(r, -200, 20, 50, 70, 100, 50)  
  
TypeError: npv() takes 2 positional arguments but 7 were given
```

Must pass the values as a list, that excludes the r values. So we need a list for cf.

```
In [100...]: # Note the r rate has already been defined.  
npv(r, [-200, 20, 50, 70, 100, 50])
```

```
Out[100...]: 38.71337130837991
```

Can use rate and args instead. The * shows all or the rest,

With this method the first value is recognized as the rate. Everything that comes after that is known to be the args.

```
In [ ]: def npv(rate, *args):  
    NPV = 0  
    for i in range(len(values)):  
        NPV += values[i] / (1 + rate) ** (i)  
    return NPV
```

```
In [102...]: # Example of what does not work. It does not work because as is there are way too many args  
npv(r, -200, 20, 50, 70, 100, 100, 50)
```

```
-----  
TypeError                                     Traceback (most recent call last)  
Input In [102], in <cell line: 2>()  
1 # Example of what does not work. It does not work because as is there are way too many arguments.  
----> 2 npv(r, -200, 20, 50, 70, 100, 100, 50)
```

```
TypeError: npv() takes 2 positional arguments but 8 were given
```

```
In [109...]: # Must use rate and *args before passing in the values.  
# It could be named anything but this is the standard convention.  
# The * helps it pick up the rest of the values.  
rate, *args = (r, -200, 20, 50, 70, 100, 50)
```

```
In [104...]: rate
```

```
Out[104...]: 0.06
```

```
In [110...]: args
```

```
Out[110...]: [-200, 20, 50, 70, 100, 50]
```

Returning several results using functions

In [116]:

```
def npv_irr(rate, values, guess = 0.05):  
  
    NPV = 0  
    for i in range(len(values)):  
        NPV += values[i] / (1 + rate) ** (i)  
  
    step = 0.0000001  
    target_npv = 0  
    tolerance = 0.001  
  
    while True:  
        f = 1 + guess  
        npv = 0  
        for i in range(len(values)):  
            npv += values[i] / f** (i)  
            diff = npv - target_npv  
  
        if abs(diff) > tolerance:  
            if diff < 0:  
                guess -= step  
            elif diff > 0:  
                guess += step  
        else:  
            break  
    return NPV, guess
```

The function returns two values.

The code above throws an infinite loop but that is what it is supposed to do, return two values.

In []:

```
npv_irr(rate = r, values = cf, guess = 0.06)
```

In []:

```
npv, irr = npv_irr(rate = r, values = cf, guess = 0.06)
```

In []:

```
npv
```

In []:

```
irr
```

Functions make local variables not global ones.

In order to make a function global make a global declaration.

The same variables can be used inside of a function call due to their scopes being different.

```
In [17]: a = 10  
b = 20
```

```
In [18]: def my_func(a, b):  
    a += 5  
    b += 5  
    return a, b
```

```
In [19]: my_func(a = a, b = b)
```

```
Out[19]: (15, 25)
```

If a variable is referenced within a function and it does not exist the eponymous variable inside of the global scope will be used by last resort.

```
In [118]: a = 10
```

```
In [119]: def addition(b):  
    #a = 100  
    add = a +b  
    return add
```

```
In [120]: # Note only b has been defined by a user input. a has not been defined except globally.  
# Set b = 20  
addition(20)
```

```
Out[120]: 30
```

```
In [ ]: a = 10
```

```
In [121]: # Priority will be given to the locally defined variable.  
def addition(b):  
    a = 100  
    add = a +b  
    return add
```

```
In [122]: addition(20)
```

```
Out[122]: 120
```

Functions can be used within other functions, as long as they are defined first.

Un-nested function.

```
In [5]: def irr1(values):
    guess = 0.06
    step = 0.0000001
    target_npv = 0
    tolerance = 0.001

    while True:
        f = 1 + guess
        NPV = 0
        for i in range(len(values)):
            NPV += values[i] / f**(i)
        diff = NPV - target_npv

        if abs(diff) > tolerance:
            if diff < 0:
                guess -= step
            elif diff > 0:
                guess += step
            else:
                break
    return guess
```

Nested function

Calls a function instead of hardcoding values.

```
In [ ]: def irr2(values):
    guess = 0.06
    step = 0.0000001
    target_npv = 0
    tolerance = 0.001

    while True:
        f = 1 + guess
        NPV = npv(values, guess)
        diff = NPV - target_npv

        if abs(diff) > tolerance:
            if diff < 0:
                guess -= step
            elif diff > 0:
                guess += step
            else:
                break
    return guess
```

Functions can be passed into other functions as keyword

arguments or positional ones.

In []:

In []:

In []:

Importing and Merging many files

```
In [1]: import pandas as pd
```

Dataset info

There needed to be five occurrences of the name in order to get it on the list. Also there are no headers. The first row becomes the header automatically.

```
In [26]: pd.read_csv("yob1880.txt").head()
```

```
Out[26]:
```

	Mary	F	7065
0	Anna	F	2604
1	Emma	F	2003
2	Elizabeth	F	1939
3	Minnie	F	1746
4	Margaret	F	1578

```
In [39]: df_1880 = pd.read_csv("yob1880.txt", header = None, names = ["Name", "Gender", "Count"])
```

```
In [40]: df_1880
```

```
Out[40]:
```

	Name	Gender	Count
0	Mary	F	7065
1	Anna	F	2604
2	Emma	F	2003
3	Elizabeth	F	1939
4	Minnie	F	1746
...
1995	Woodie	M	5
1996	Worthy	M	5
1997	Wright	M	5
1998	York	M	5
1999	Zachariah	M	5

2000 rows × 3 columns

```
In [41]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
```

```
Data columns (total 3 columns):  
 #  Column  Non-Null Count Dtype  
 ---  -----  -----  -----  
 0  Name    5 non-null    object  
 1  Gender   5 non-null    object  
 2  Count    5 non-null    int64  
 dtypes: int64(1), object(2)  
 memory usage: 248.0+ bytes
```

```
In [42]: df_1881 = pd.read_csv("yob1881.txt", header = None, names = ["Name", "Gender", "Count"])  
df_1881
```

```
Out[42]:
```

	Name	Gender	Count
0	Mary	F	6919
1	Anna	F	2698
2	Emma	F	2034
3	Elizabeth	F	1852
4	Margaret	F	1658
...
1930	Wiliam	M	5
1931	Wilton	M	5
1932	Wing	M	5
1933	Wood	M	5
1934	Wright	M	5

1935 rows × 3 columns

Simple Concatination

Flawed Method

Use pd.concat to combine dataframes

To connect the dataframes vertically use this. The axis must be zero to combine vertically. Because 0 refers to the index. 1 would refer to combining columns side by side.

pd.concat(objs = [dataframe1, dataframe2], axis = 0)

Always make sure that the names are constant when concatinating.

See why it is flawed below.

```
In [43]: pd.concat(objs = [df_1880, df_1881], axis = 0)
```

```
Out[43]:
```

	Name	Gender	Count
0	Mary	F	7065

	Name	Gender	Count
1	Anna	F	2604
2	Emma	F	2003
3	Elizabeth	F	1939
4	Minnie	F	1746
...
1930	Wiliam	M	5
1931	Wilton	M	5
1932	Wing	M	5
1933	Wood	M	5
1934	Wright	M	5

3935 rows × 3 columns

Issues with this method

The indexes reset

Notice how there is 3935 rows. However the last index is at 1934. This is because the counter reset between the two sets.

It is hard to tell which data set the info is from.

There is no clear indication of when one year ends and the other one starts.

The Good Method

```
pd.concat(objs = [df_1, df_2], axis = 0, keys = [year1, year2])
```

```
pd.concat(objs = [df_1, df_2], axis = 0, keys = [year1, year2])
```

In [53]:

```
pd.concat(objs = [df_1880, df_1881], axis = 0, keys = [1880, 1881], names = ["Year"])
```

Out[53]:

	Name	Gender	Count
Year			
1880	0	Mary	F
	1	Anna	F
	2	Emma	F
	3	Elizabeth	F
	4	Minnie	F
...
1881	1930	Wiliam	M
1931		Wilton	M

	Name	Gender	Count
--	------	--------	-------

Year

1932	Wing	M	5
1933	Wood	M	5
1934	Wright	M	5

3935 rows × 3 columns

An improvement with no index.

In [58]:

```
pd.concat(objs = [df_1880, df_1881], axis = 0, keys = [1880, 1881], names = ["Year"]).drop
```

Out[58]:

	Name	Gender	Count
--	------	--------	-------

Year

1880	Mary	F	7065
1880	Anna	F	2604
1880	Emma	F	2003
1880	Elizabeth	F	1939
1880	Minnie	F	1746
...
1881	Wiliam	M	5
1881	Wilton	M	5
1881	Wing	M	5
1881	Wood	M	5
1881	Wright	M	5

3935 rows × 3 columns

Drop the Index inside of the table to make the separation part of the table.

The droplevel makes the year part of the table. The reset index brings back the index numbers.

In [57]:

```
pd.concat(objs = [df_1880, df_1881], axis = 0, keys = [1880, 1881], names = ["Year"]).drop
```

Out[57]:

	Year	Name	Gender	Count
--	------	------	--------	-------

0	1880	Mary	F	7065
1	1880	Anna	F	2604
2	1880	Emma	F	2003
3	1880	Elizabeth	F	1939
4	1880	Minnie	F	1746

Year	Name	Gender	Count
...
3930	1881	Wiliam	M 5
3931	1881	Wilton	M 5
3932	1881	Wing	M 5
3933	1881	Wood	M 5
3934	1881	Wright	M 5

3935 rows × 4 columns

Importing Many Datasets using a for loop

In [59]:

```
pd.read_csv("yob{}.txt".format(1880), header = None, names = ["Names", "Gender", "Count"])
```

Out[59]:

	Names	Gender	Count
0	Mary	F 7065	
1	Anna	F 2604	
2	Emma	F 2003	
3	Elizabeth	F 1939	
4	Minnie	F 1746	
...
1995	Woodie	M 5	
1996	Worthy	M 5	
1997	Wright	M 5	
1998	York	M 5	
1999	Zachariah	M 5	

2000 rows × 3 columns

In [78]:

```
years = list(range(1880, 2019))
years[0:20]
```

Out[78]:

```
[1880,
 1881,
 1882,
 1883,
 1884,
 1885,
 1886,
 1887,
 1888,
 1889,
 1890,
 1891,
 1892,
 1893,
```

```
1894,  
1895,  
1896,  
1897,  
1898,  
1899]
```

In [61]:

```
dataframes = []  
for year in years:  
    data = pd.read_csv("yob{}.txt".format(year), header = None,  
                       names = ["Name", "Gender", "Count"])  
    dataframes.append(data)
```

In [79]:

```
# See Dataframes at the end of the document  
# dataframes
```

In [63]:

```
len(dataframes)
```

Out[63]:

```
139
```

In [64]:

```
df = pd.concat(dataframes, axis = 0, keys = years, names = ["Year"]).droplevel(-1).reset_index()
```

In [65]:

```
df
```

Out[65]:

	Year	Name	Gender	Count
0	1880	Mary	F	7065
1	1880	Anna	F	2604
2	1880	Emma	F	2003
3	1880	Elizabeth	F	1939
4	1880	Minnie	F	1746
...
1957041	2018	Zylas	M	5
1957042	2018	Zyran	M	5
1957043	2018	Zyrie	M	5
1957044	2018	Zyron	M	5
1957045	2018	Zzyzx	M	5

1957046 rows × 4 columns

In [68]:

```
# Shows how many mb the data takes up  
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1957046 entries, 0 to 1957045  
Data columns (total 4 columns):  
 #   Column  Dtype  
---  --  --  
 0   Year    int64
```

```
1  Name    object
2  Gender   object
3  Count    int64
dtypes: int64(2), object(2)
memory usage: 59.7+ MB
```

Final concatenation with almost 2 million rows

```
In [69]: df
```

```
Out[69]:
```

	Year	Name	Gender	Count
0	1880	Mary	F	7065
1	1880	Anna	F	2604
2	1880	Emma	F	2003
3	1880	Elizabeth	F	1939
4	1880	Minnie	F	1746
...
1957041	2018	Zylas	M	5
1957042	2018	Zyran	M	5
1957043	2018	Zyrie	M	5
1957044	2018	Zyron	M	5
1957045	2018	Zzyzx	M	5

1957046 rows × 4 columns

Final Steps

Convert Document to a CSV and reimport

```
In [70]: df.to_csv("us_baby_names.csv", index = False)
```

```
In [71]: pd.read_csv("us_baby_names.csv")
```

```
Out[71]:
```

	Year	Name	Gender	Count
0	1880	Mary	F	7065
1	1880	Anna	F	2604
2	1880	Emma	F	2003
3	1880	Elizabeth	F	1939
4	1880	Minnie	F	1746
...
1957041	2018	Zylas	M	5
1957042	2018	Zyran	M	5

Year	Name	Gender	Count
1957043	Zyrie	M	5
1957044	Zyron	M	5
1957045	Zzyzx	M	5

1957046 rows × 4 columns

In [80]:

```
#Dataframes Screenshot  
dataframes
```

Out[77]:

	Name	Gender	Count
0	Mary	F	7065
1	Anna	F	2604
2	Emma	F	2003
3	Elizabeth	F	1939
4	Minnie	F	1746
...
1995	Woodie	M	5
1996	Worthy	M	5
1997	Wright	M	5
1998	York	M	5
1999	Zachariah	M	5

[2000 rows × 3 columns],

	Name	Gender	Count
0	Mary	F	6919
1	Anna	F	2698
2	Emma	F	2034
3	Elizabeth	F	1852
4	Margaret	F	1658
...
1930	Wiliam	M	5
1931	Wilton	M	5
1932	Wing	M	5
1933	Wood	M	5
1934	Wright	M	5

[1935 rows × 3 columns],

In []:

Project 6: Importing & Merging many files (Baby Names Dataset) - Part 2

Project Brief for Self-Coders

Getting the Files from the Web

1. Go to <https://catalog.data.gov/dataset/baby-names-from-social-security-card-applications-data-by-state-and-district-of-> and **download** and **unzip** the file.

Use glob to tap into the root directory.

<https://docs.python.org/3/library/glob.html#module-glob>

In [18]:

```
# Already downloaded.
import pandas as pd
import numpy as np
```

Importing one File & Understanding the Data Structure

1. Load the file "AK.txt" into Pandas and inspect.

In [17]:

```
pd.read_csv("AK.txt", header = None, names = ["State", "Gender", "Year", "Name", "Count"])
```

Out[17]:

	State	Gender	Year	Name	Count
0	AK	F	1910	Mary	14
1	AK	F	1910	Annie	12
2	AK	F	1910	Anna	10
3	AK	F	1910	Margaret	8
4	AK	F	1910	Helen	7
...
28523	AK	M	2018	Theo	5
28524	AK	M	2018	Thorin	5
28525	AK	M	2018	Trenton	5
28526	AK	M	2018	Victor	5
28527	AK	M	2018	Zion	5

28528 rows × 5 columns

The glob module

1. From glob import glob. ### Glob recognizes patterns.

```
In [19]: from glob import glob
```

"*.TXT" finds all of the files from the working directory.

1. **Find** all filenames with the structure "**A?.txt**" in your current directory (? is a single character wildcard).

```
In [21]: glob("*.TXT")
```

```
Out[21]: ['AK.TXT',  
          'AL.TXT',  
          'AR.TXT',  
          'AZ.TXT',  
          'CA.TXT',  
          'CO.TXT',  
          'CT.TXT',  
          'DC.TXT',  
          'DE.TXT',  
          'FL.TXT',  
          'GA.TXT',  
          'HI.TXT',  
          'IA.TXT',  
          'ID.TXT',  
          'IL.TXT',  
          'IN.TXT',  
          'KS.TXT',  
          'KY.TXT',  
          'LA.TXT',  
          'MA.TXT',  
          'MD.TXT',  
          'ME.TXT',  
          'MI.TXT',  
          'MN.TXT',  
          'MO.TXT',  
          'MS.TXT',  
          'MT.TXT',  
          'NC.TXT',  
          'ND.TXT',  
          'NE.TXT',  
          'NH.TXT',  
          'NJ.TXT',  
          'NM.TXT',  
          'NV.TXT',  
          'NY.TXT',  
          'OH.TXT',  
          'OK.TXT',  
          'OR.TXT',  
          'PA.TXT',  
          'RI.TXT',  
          'SC.TXT',  
          'SD.TXT',  
          'TN.TXT',  
          'TX.TXT',  
          'UT.TXT',  
          'VA.TXT',  
          'VT.TXT',  
          'WA.TXT',  
          'WI.TXT',  
          'WV.TXT',  
          'WY.TXT']
```

1. **Find** all filenames with the following structure in your current directory and save the resulting list in a

variable:

There is a file called "subdir" that can be found by putting the name of the file in the following code.

```
In [24]: glob("subdir\\*.TXT") # (* is a wildcard for zero or many characters)
```

```
Out[24]: ['subdir\\yob1880.txt', 'subdir\\yob1881.txt']
```

```
In [ ]: glob("file_name\\*.TXT")
```

Importing & merging many Files (complex case)

1. **Load** all files (*.txt) and **merge/concatenate** all files into one Pandas DataFrame.

1. Create a **RangeIndex** and **save** the DataFrame (with columns "State", "Gender", "Year", "Name", "Count") in a new csv-file.

```
In [25]: filenames = glob("*.TXT")
```

```
In [26]: filenames
```

```
Out[26]: ['AK.TXT',  
          'AL.TXT',  
          'AR.TXT',  
          'AZ.TXT',  
          'CA.TXT',  
          'CO.TXT',  
          'CT.TXT',  
          'DC.TXT',  
          'DE.TXT',  
          'FL.TXT',  
          'GA.TXT',  
          'HI.TXT',  
          'IA.TXT',  
          'ID.TXT',  
          'IL.TXT',  
          'IN.TXT',  
          'KS.TXT',  
          'KY.TXT',  
          'LA.TXT',  
          'MA.TXT',  
          'MD.TXT',  
          'ME.TXT',  
          'MI.TXT',  
          'MN.TXT',  
          'MO.TXT',  
          'MS.TXT',  
          'MT.TXT',  
          'NC.TXT',  
          'ND.TXT',  
          'NE.TXT',  
          'NH.TXT',  
          'NJ.TXT',  
          'NM.TXT',  
          'NV.TXT',  
          'NY.TXT',  
          'OH.TXT',  
          'OK.TXT',  
          'OR.TXT',  
          'PA.TXT',  
          'RI.TXT',  
          'SD.TXT',  
          'UT.TXT',  
          'VA.TXT',  
          'VI.TXT',  
          'WA.TXT',  
          'WV.TXT',  
          'WY.TXT']
```

```
'NV.TXT',
'NY.TXT',
'OH.TXT',
'OK.TXT',
'OR.TXT',
'PA.TXT',
'RI.TXT',
'SC.TXT',
'SD.TXT',
'TN.TXT',
'TX.TXT',
'UT.TXT',
'VA.TXT',
'VT.TXT',
'WA.TXT',
'WI.TXT',
'WV.TXT',
'WY.TXT']
```

```
In [29]: len(filenames)
```

```
Out[29]: 51
```

```
In [37]: dataframes = []
for name in filenames:
    df = pd.read_csv(name, header = None, names = ["State", "Gender", "Year", "Name", "Count"])
    dataframes.append(df)
```

```
In [39]: # Ignore index creates a range from start to finish.
df = pd.concat(dataframes, ignore_index = True)
```

```
df
```

```
In [40]: df
```

```
Out[40]:
```

	State	Gender	Year	Name	Count
0	AK	F	1910	Mary	14
1	AK	F	1910	Annie	12
2	AK	F	1910	Anna	10
3	AK	F	1910	Margaret	8
4	AK	F	1910	Helen	7
...
6028146	WY	M	2018	Peyton	5
6028147	WY	M	2018	Richard	5
6028148	WY	M	2018	Titus	5
6028149	WY	M	2018	Tristan	5
6028150	WY	M	2018	Zander	5

```
6028151 rows × 5 columns
```

```
In [35]:
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28020 entries, 0 to 28019
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
--- 
 0   State    28020 non-null   object  
 1   Gender   28020 non-null   object  
 2   Year     28020 non-null   int64  
 3   Name     28020 non-null   object  
 4   Count    28020 non-null   int64  
dtypes: int64(2), object(3)
memory usage: 1.1+ MB
```

```
In [42]: len(dataframes)
```

```
Out[42]: 51
```

```
In [43]: df.to_csv("baby_names_state.csv", index = False)
```

```
In [45]: pd.read_csv("baby_names_state.csv")
```

```
Out[45]:
```

	State	Gender	Year	Name	Count
0	AK	F	1910	Mary	14
1	AK	F	1910	Annie	12
2	AK	F	1910	Anna	10
3	AK	F	1910	Margaret	8
4	AK	F	1910	Helen	7
...
6028146	WY	M	2018	Peyton	5
6028147	WY	M	2018	Richard	5
6028148	WY	M	2018	Titus	5
6028149	WY	M	2018	Tristan	5
6028150	WY	M	2018	Zander	5

```
6028151 rows × 5 columns
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

Project 8: Data Preprocessing & Feature Engineering for Machine Learning (Housing Dataset)

Project Brief for Self-Coders

Here you'll have the opportunity to code major parts of Project 8 on your own. If you need any help or inspiration, have a look at the Videos or the Jupyter Notebook with the full code.

Keep in mind that it's all about **getting the right results/conclusions**. It's not about finding the identical code. Things can be coded in many different ways. Even if you come to the same conclusions, it's very unlikely that we have the very same code.

Data Import and first Inspection

1. **Import** the housing dataset (housing.csv) and **inspect**!

In [60]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

In [2]:

```
df = pd.read_csv("housing.csv")
```

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   longitude        20640 non-null   float64
 1   latitude         20640 non-null   float64
 2   housing_median_age 20640 non-null   float64
 3   total_rooms      20640 non-null   float64
 4   total_bedrooms   20433 non-null   float64
 5   population       20640 non-null   float64
 6   households       20640 non-null   float64
 7   median_income    20640 non-null   float64
 8   median_house_value 20640 non-null   float64
 9   ocean_proximity  20640 non-null   object  
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

In [4]:

```
df
```

Out[4]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_inco
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.31
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.31
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.21

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_inco
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8
...
20635	-121.09	39.48	25.0	1665.0	374.0	845.0	330.0	1.5
20636	-121.21	39.49	18.0	697.0	150.0	356.0	114.0	2.5
20637	-121.22	39.43	17.0	2254.0	485.0	1007.0	433.0	1.7
20638	-121.32	39.43	18.0	1860.0	409.0	741.0	349.0	1.8
20639	-121.24	39.37	16.0	2785.0	616.0	1387.0	530.0	2.3

20640 rows × 10 columns

Features:

- **longitude:** geographic coordinate (district's east-west position)
- **latitude:** geographic coordinate (district's north-south position)
- **housing_median_age:** median age of houses in district
- **total_rooms** Sum of all rooms in district
- **total_bedrooms** Sum of all bedrooms in district
- **population:** total population in district
- **households:** total households in district
- **median_income:** median household income in district
- **median_house_value:** median house value in district
- **ocean_proximity:** District's proximity to the ocean

In []:

Data Cleaning and Creating additional Features

1. **Drop** all rows with (at least one) missing value(s).
1. **Add** the additional Feature "**rooms_per_household**" (should be self-explanatory)
1. **Add** the additional Feature "**population_per_household**" (should be self-explanatory)
1. **Add** the additional Feature "**bedrooms_per_household**" (should be self-explanatory)

In [7]:

```
# Drop rows with missing values.
df.dropna(axis = 0, how = "any")
```

Out[7]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_inco
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_inco
3	-122.25	37.85		52.0	1274.0	235.0	558.0	219.0
4	-122.25	37.85		52.0	1627.0	280.0	565.0	259.0
...
20635	-121.09	39.48		25.0	1665.0	374.0	845.0	330.0
20636	-121.21	39.49		18.0	697.0	150.0	356.0	114.0
20637	-121.22	39.43		17.0	2254.0	485.0	1007.0	433.0
20638	-121.32	39.43		18.0	1860.0	409.0	741.0	349.0
20639	-121.24	39.37		16.0	2785.0	616.0	1387.0	530.0

20433 rows × 10 columns

In [15]:

```
# rooms_per_households = total_rooms / households
df["rooms_per_household"] = df["total_rooms"] / df["households"]
df["rooms_per_household"]
```

Out[15]:

```
0      6.984127
1      6.238137
2      8.288136
3      5.817352
4      6.281853
      ...
20635    5.045455
20636    6.114035
20637    5.205543
20638    5.329513
20639    5.254717
Name: rooms_per_household, Length: 20640, dtype: float64
```

In [16]:

```
# population_per_household = population / households
df["population_per_household"] = df["population"] / df["households"]
df["population_per_household"]
```

Out[16]:

```
0      2.555556
1      2.109842
2      2.802260
3      2.547945
4      2.181467
      ...
20635    2.560606
20636    3.122807
20637    2.325635
20638    2.123209
20639    2.616981
Name: population_per_household, Length: 20640, dtype: float64
```

In [20]:

```
# Bedrooms per households
df["bedrooms_per_household"] = df.total_bedrooms / df.households
df.bedrooms_per_household
```

Out[20]:

```
0      1.023810
1      0.971880
2      1.073446
3      1.073059
```

```
4      1.081081
...
20635  1.133333
20636  1.315789
20637  1.120092
20638  1.171920
20639  1.162264
Name: bedrooms_per_household, Length: 20640, dtype: float64
```

Which Factors influence House Prices?

1. Calculate the Correlation between "median_house_value" and all features. Which factors seems to influence house prices/values?

```
In [124]: df.corr().loc[:, "median_house_value"].sort_values(ascending = False)
```

```
Out[124]: median_house_value      1.000000
median_income          0.688075
rooms_per_household   0.151948
total_rooms            0.134153
housing_median_age    0.105623
households             0.065843
total_bedrooms         0.049686
population_per_household -0.023737
population             -0.024650
longitude              -0.045967
bedrooms_per_household -0.046739
latitude               -0.144160
Name: median_house_value, dtype: float64
```

```
In [125]: df.corr().loc["median_house_value"].sort_values(ascending = False)
```

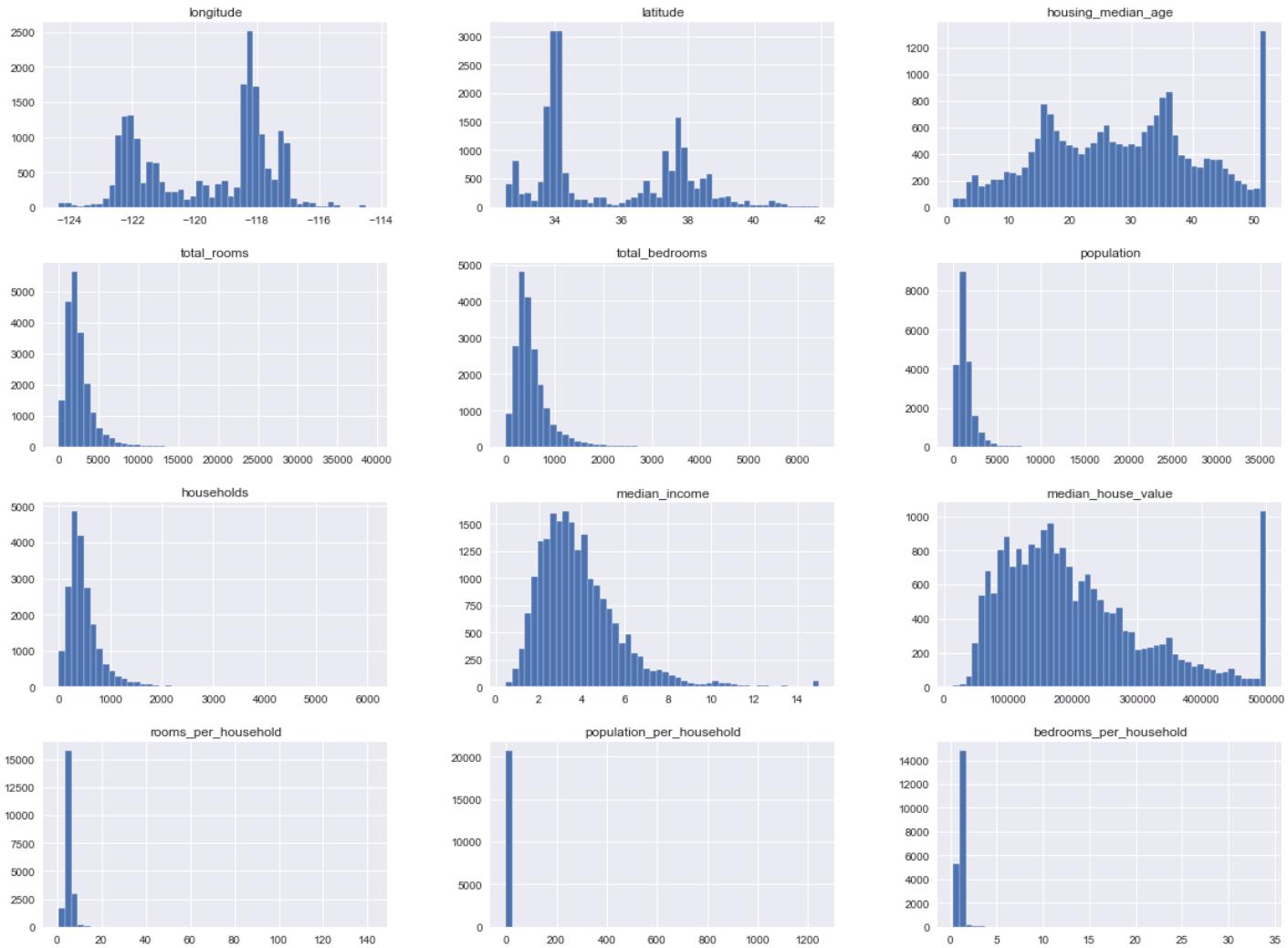
```
Out[125]: median_house_value      1.000000
median_income          0.688075
rooms_per_household   0.151948
total_rooms            0.134153
housing_median_age    0.105623
households             0.065843
total_bedrooms         0.049686
population_per_household -0.023737
population             -0.024650
longitude              -0.045967
bedrooms_per_household -0.046739
latitude               -0.144160
Name: median_house_value, dtype: float64
```

```
In [35]: df.columns
```

```
Out[35]: Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity', 'rooms_per_household',
       'population_per_household', 'bedrooms_per_household'],
      dtype='object')
```

A histogram for everything

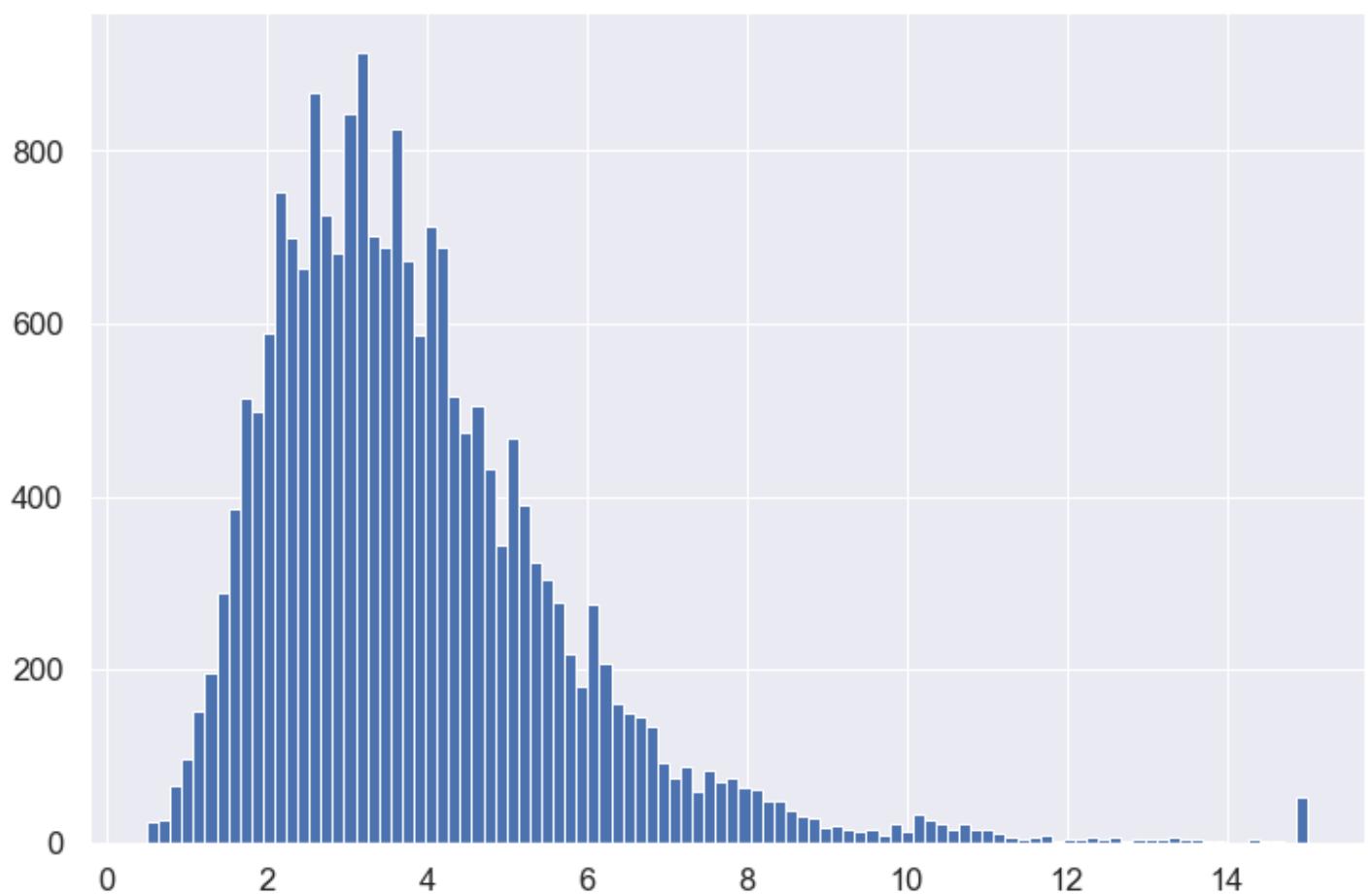
```
In [142]: # Bins is the number of intervals
df.hist(bins = 50, figsize = (20, 15))
plt.show()
```



Use df.column.hist(bins, figsize)

This will make a histogram.

```
In [126]: df.median_income.hist(bins = 100, figsize = (12,8))
plt.show()
```

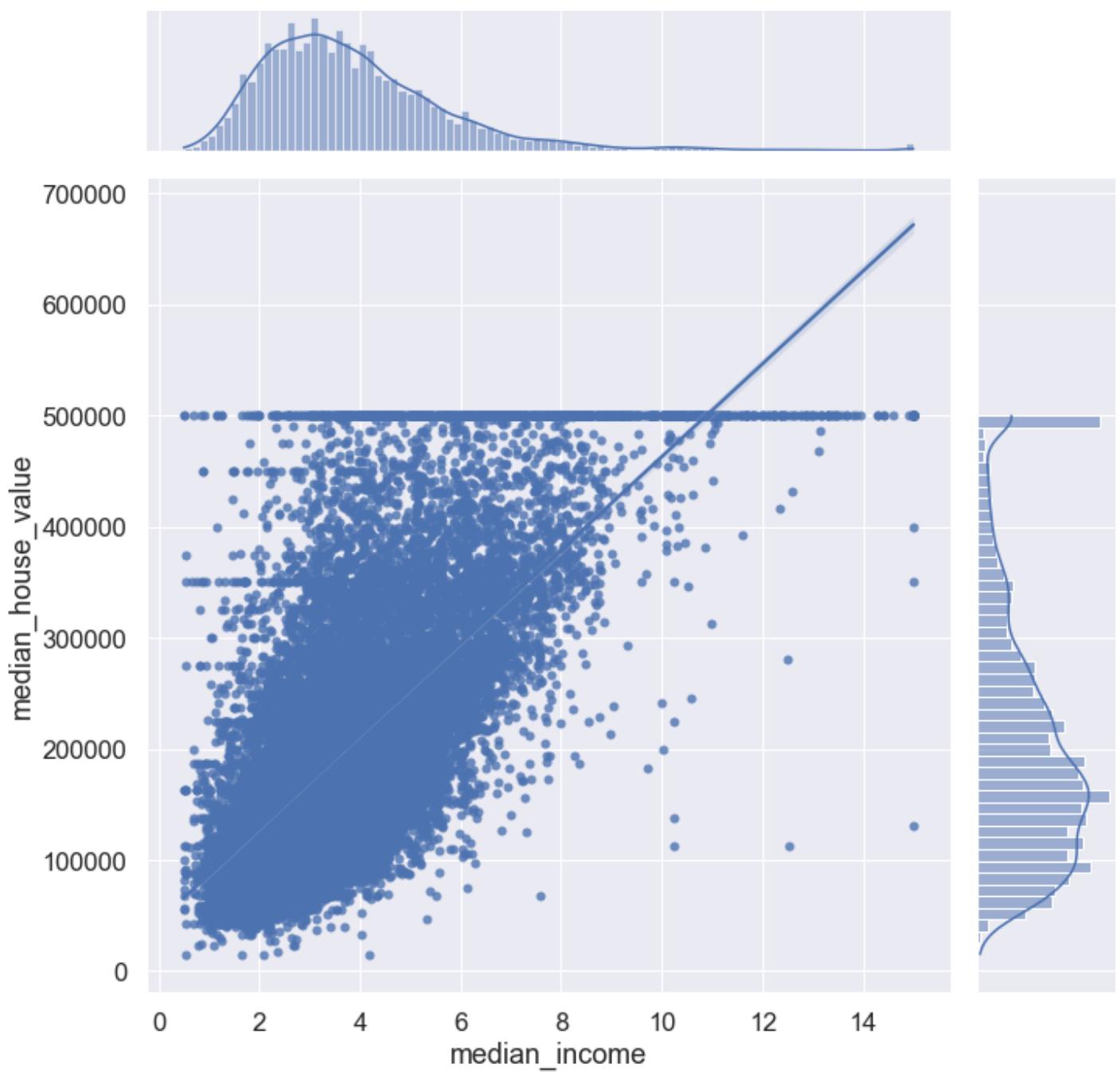


1. Create a Seaborn Regression plot (`jointplot`) with income on the x-axis and house value on the y-axis.

Joint plots kind = "reg" for regression and kind = "kde" for kernel density estimator

In [177]:

```
sns.set(font_scale = 1.5)
sns.jointplot(data = df, x = "median_income", y = "median_house_value",
              kind = "reg", height = 10)
plt.show()
```



In []:

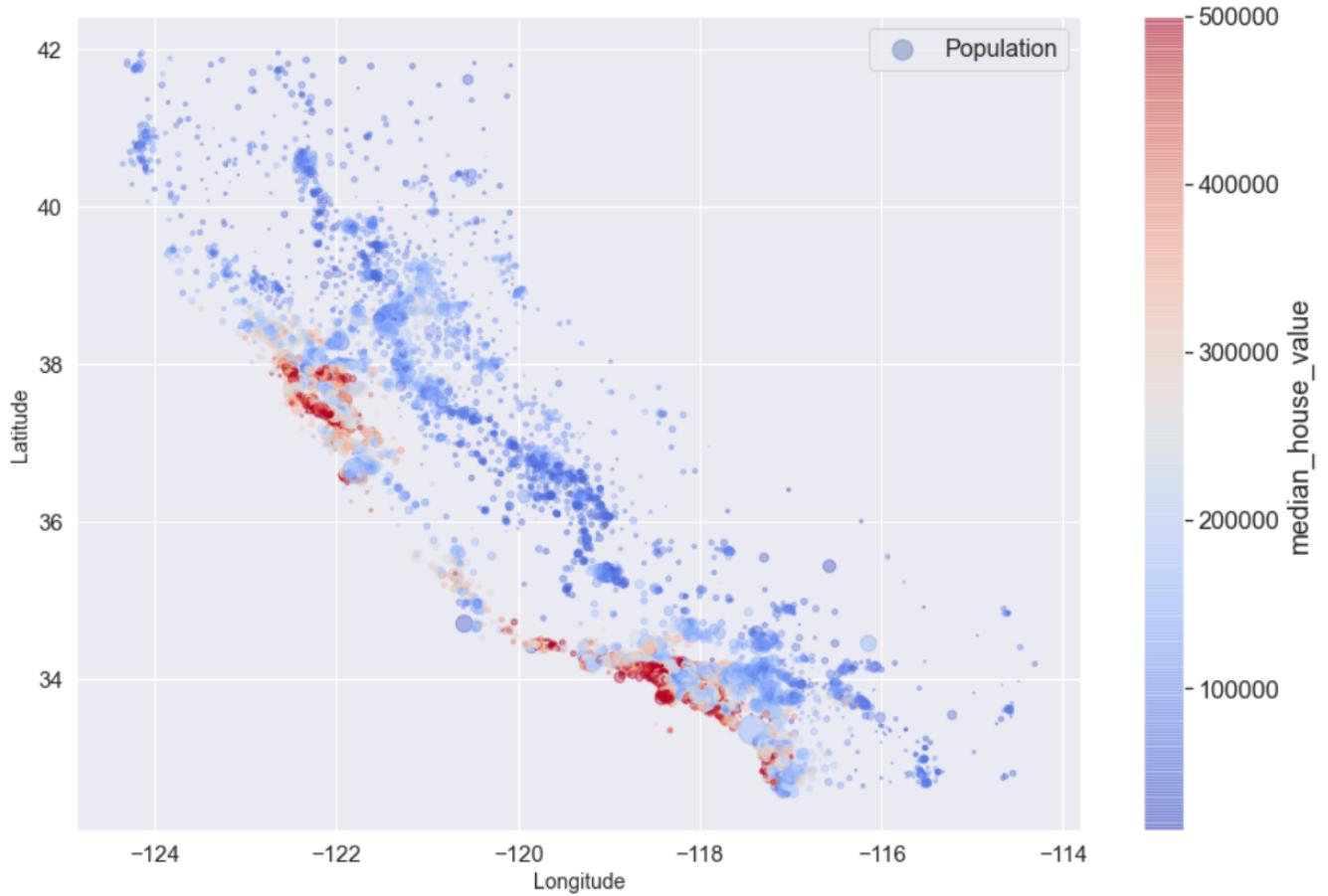
```

sns.set(font_scale = 1.5)
sns.jointplot(data = df, x = "median_income", y = "median_house_value", kind = "kde",
               height = 10)
plt.show()

```

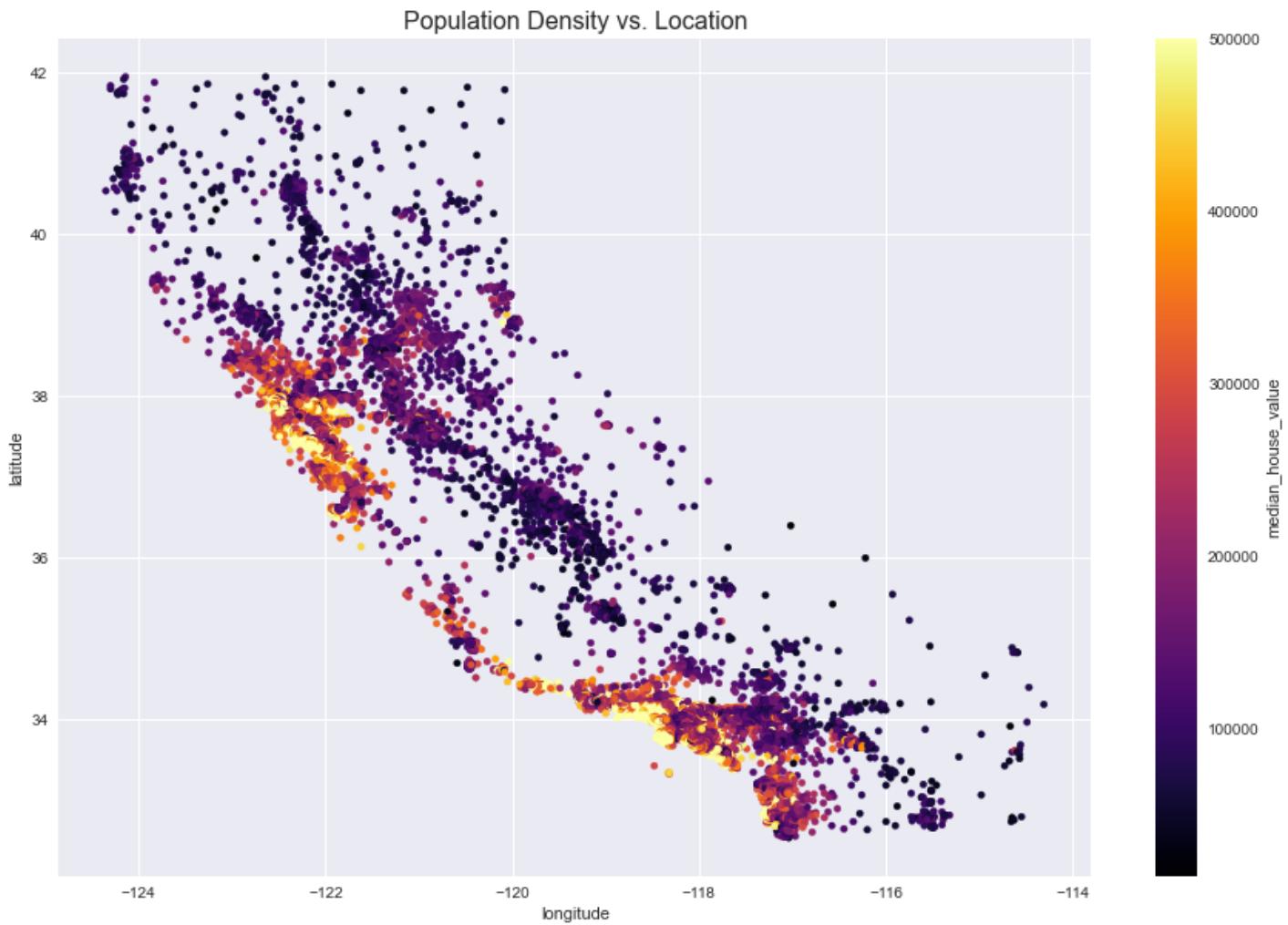
1. Create the following **scatterplot** (df.plot(kind = "scatter")) with

- longitude on x-axis
- latitude on y-axis
- size (s) of data points determined by population
- color (c) of data points determined by median_house_value

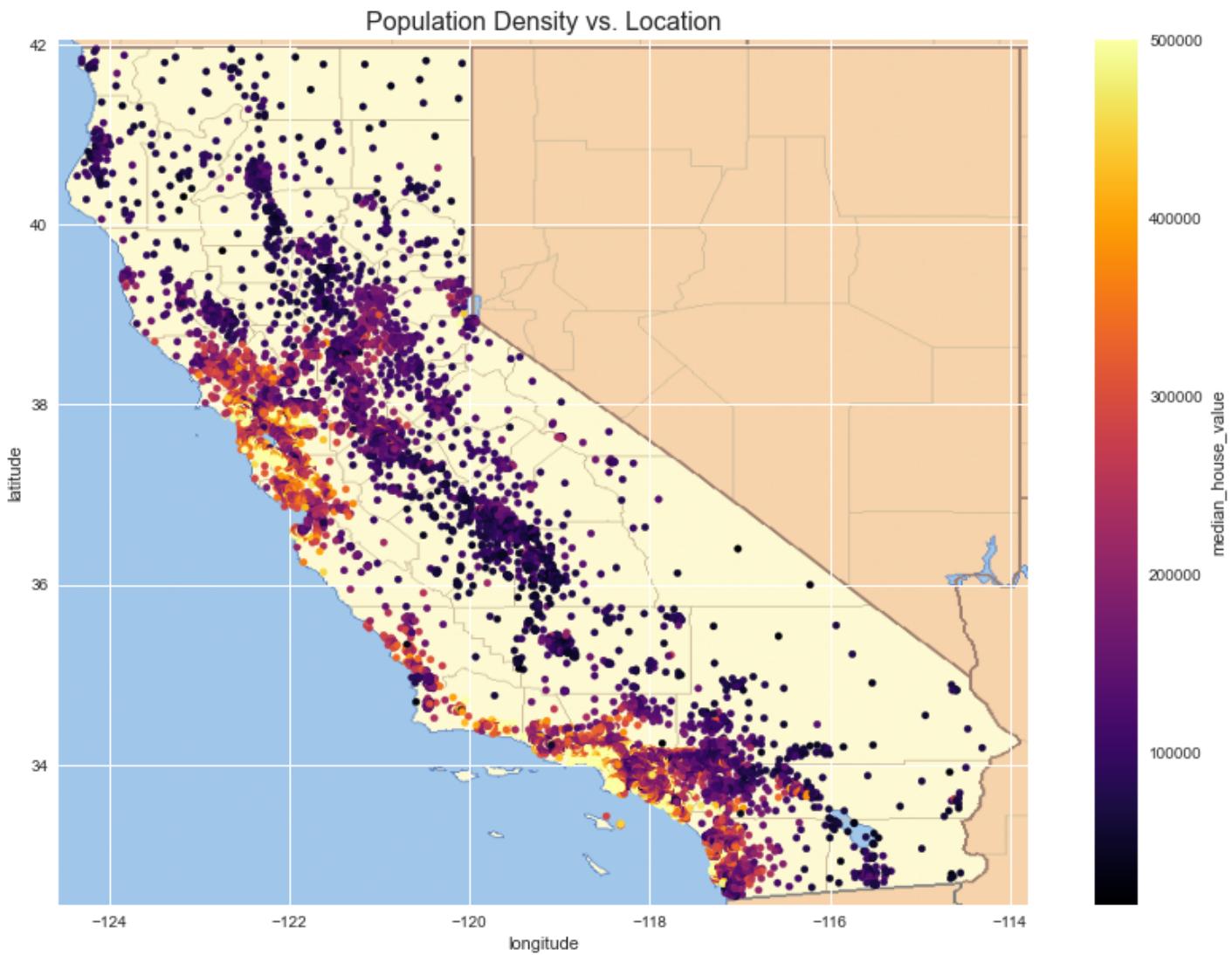


```
In [168...]:  
plt.style.use("seaborn")  
df.plot(figsize = (15, 10), kind = "scatter", x = "longitude",  
        y = "latitude", c = "median_house_value",  
        sharex = False, colormap = "inferno")  
  
plt.title("Population Density vs. Location", fontsize = 16)  
  
# import matplotlib.image as mpimg  
# california_img = mpimg.imread("california.png")  
# plt.imshow(california_img, extent = [-124.55, -113.80, 32.45, 42.05])  
  
plt.plot()
```

```
Out[168...]: []
```



```
In [167...]:  
plt.style.use("seaborn")  
df.plot(figsize = (15, 10), kind = "scatter", x = "longitude", y = "latitude",  
       c = "median_house_value", sharex = False, colormap = "inferno")  
plt.title("Population Density vs. Location", fontsize = 16)  
  
import matplotlib.image as mpimg  
california_img = mpimg.imread("california.png")  
plt.imshow(california_img, extent = [-124.55, -113.80, 32.45, 42.05])  
  
plt.plot()  
[  
Out[167...]:
```



```
In [135... prox = df.ocean_proximity.unique()
prox
```

```
Out[135... array(['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND'],
      dtype=object)
```

```
In [136... df_loc = df[df.ocean_proximity == prox[3]].copy()
```

```
In [138... df_loc2 = df[df.ocean_proximity == prox[2]].copy()
```

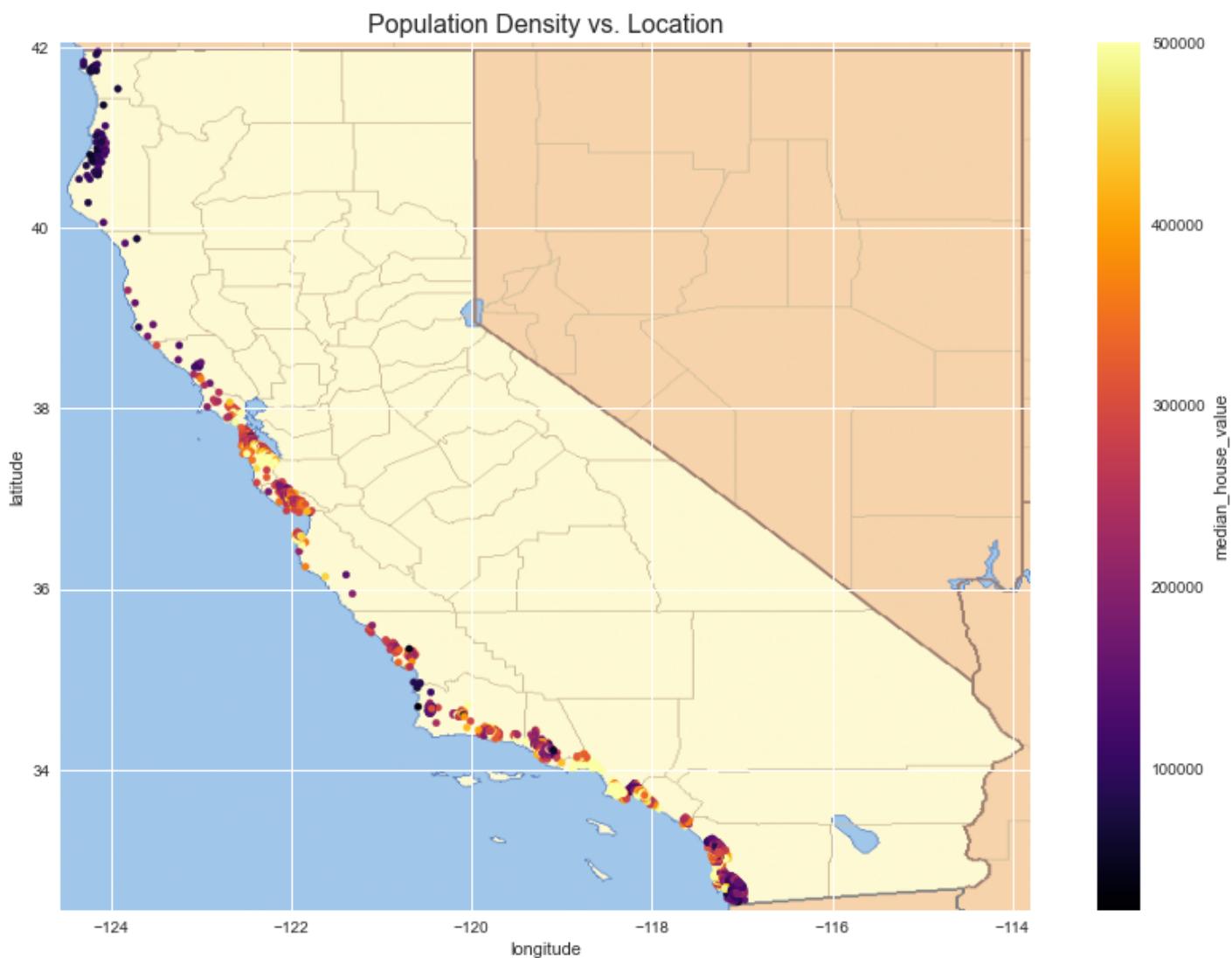
Plot for near ocean district using a custom filter

```
In [166... plt.style.use("seaborn")
# Df loc shows the ocean location
df_loc.plot(figsize = (15, 10), kind = "scatter", x = "longitude", y = "latitude",
            c = "median_house_value", sharex = False, colormap = "inferno")
plt.title("Population Density vs. Location", fontsize = 16)

import matplotlib.image as mpimg
california_img = mpimg.imread("california.png")
plt.imshow(california_img, extent = [-124.55, -113.80, 32.45, 42.05])

plt.plot()
```

Out[166...]

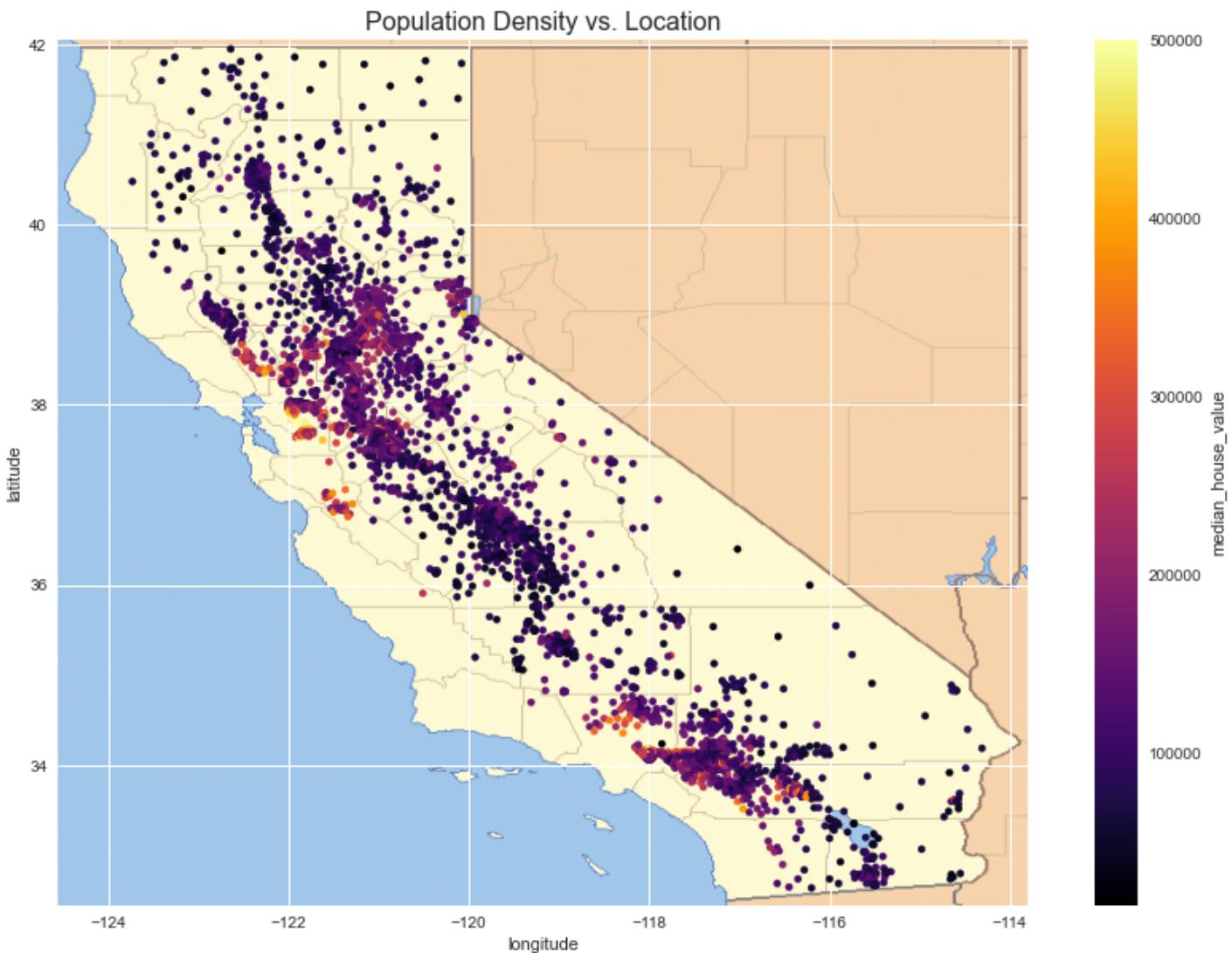


In [165...]

```
plt.style.use("seaborn")
# Df loc shows the ocean location
df_loc2.plot(figsize = (15, 10), kind = "scatter", x = "longitude", y = "latitude",
             c = "median_house_value", sharex = False, colormap = "inferno")
plt.title("Population Density vs. Location", fontsize = 16)

import matplotlib.image as mpimg
california_img = mpimg.imread("california.png")
plt.imshow(california_img, extent = [-124.55, -113.80, 32.45, 42.05])
```

Out[165...]



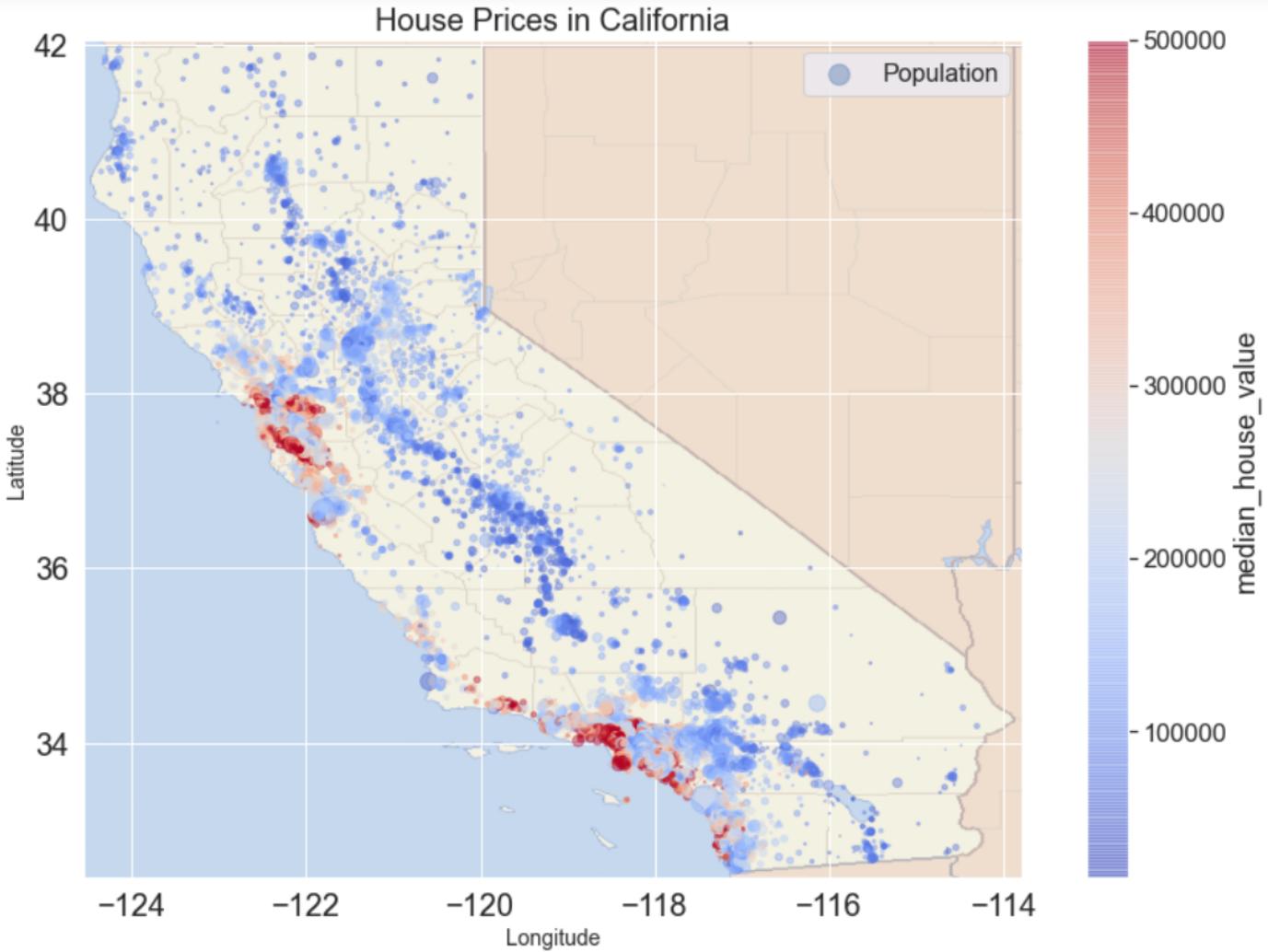
1. Does this look familiar to you? It's California. Let's **add the map** of California saved in **california.png**.

Hint: You can load and display the image **california.png** with the right latitude/longitude as follows:

In []:

```
import matplotlib.image as mpimg
california_img = mpimg.imread("california.png")
plt.figure(figsize = (15, 10))
plt.imshow(california_img, extent=[-124.55, -113.80, 32.45, 42.05])
plt.show()
```

The final plot should look like this:



In []:

Advanced Explanatory Data Analysis with Seaborn

1. Add the additional column "income_cat" with the following income categories:

- lowest 25% -> "Low"
- 25th to 50th percentile -> "Below_Average"
- 50th to 75th percentile -> "Above_Average"
- 75th to 95th percentile -> "High"
- Above 95th percentile -> "Very High"

Very important

Must use qcut to separate into bins actually use pd to start

```
pd.qcut(df["column"], q = num_cuts, labels = ["list_of_quantifiers"])
```

income_cat

The creation of a new column df.income_cat

In [141...]

```
df["income_cat"] = pd.qcut(df.median_income, q = 5, labels = ["Low", "Below Average", "Abc
```

```
In [100]: df["income_cat"]
```

```
Out[100]: 0           Very High
1           Very High
2           Very High
3           Very High
4           Above Average
...
20635      Low
20636      Below Average
20637      Low
20638      Low
20639      Below Average
Name: income_cat, Length: 20640, dtype: category
Categories (5, object): ['Low' < 'Below Average' < 'Above Average' < 'High' < 'Very High']
```

Very Important

Normalize Categories

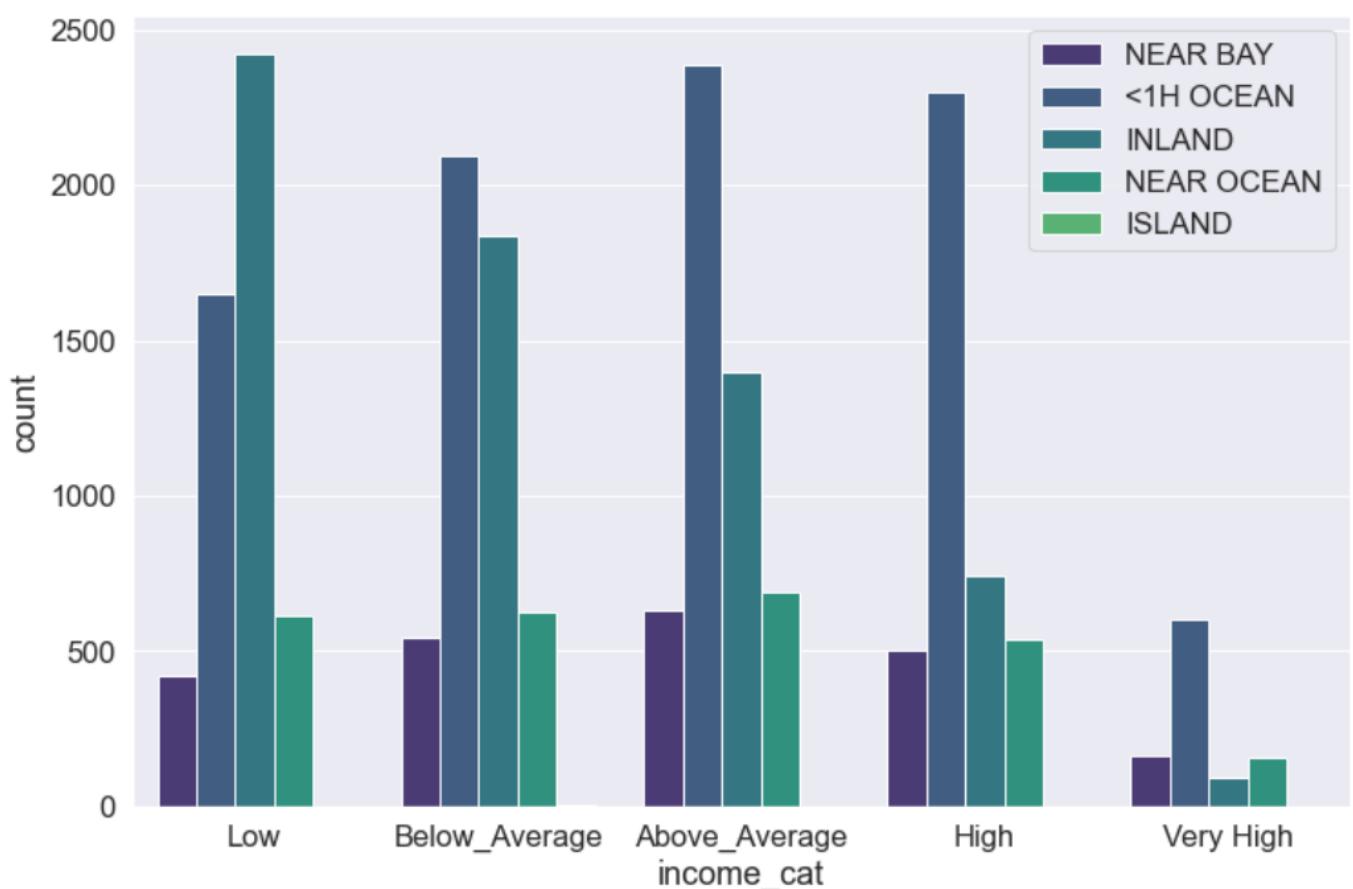
```
df.column.value_counts(normalize = True)
```

```
df.column.value_counts(normalize = True)
```

```
In [143]: df.income_cat.value_counts(normalize = True)
```

```
Out[143]: Below Average    0.200145
Low           0.200097
High          0.200000
Very High     0.200000
Above Average 0.199758
Name: income_cat, dtype: float64
```

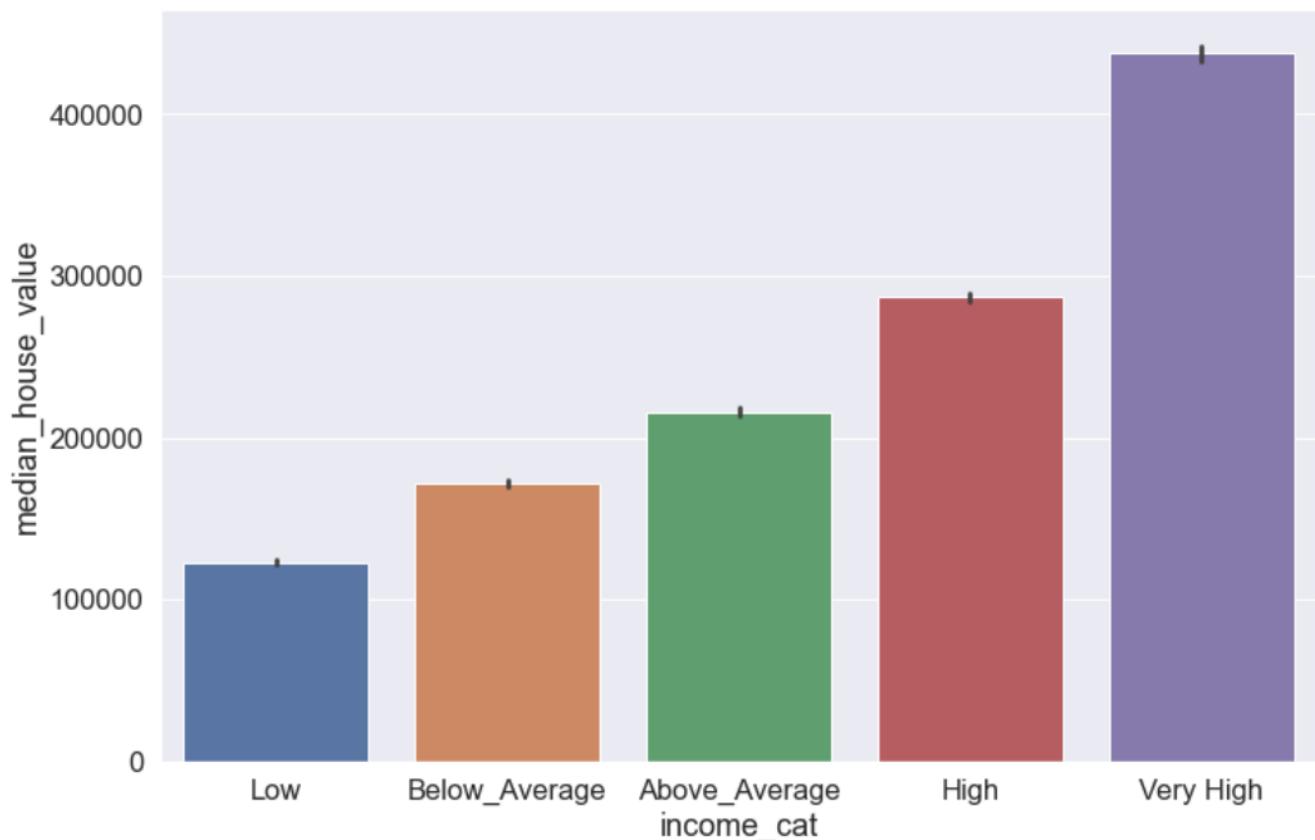
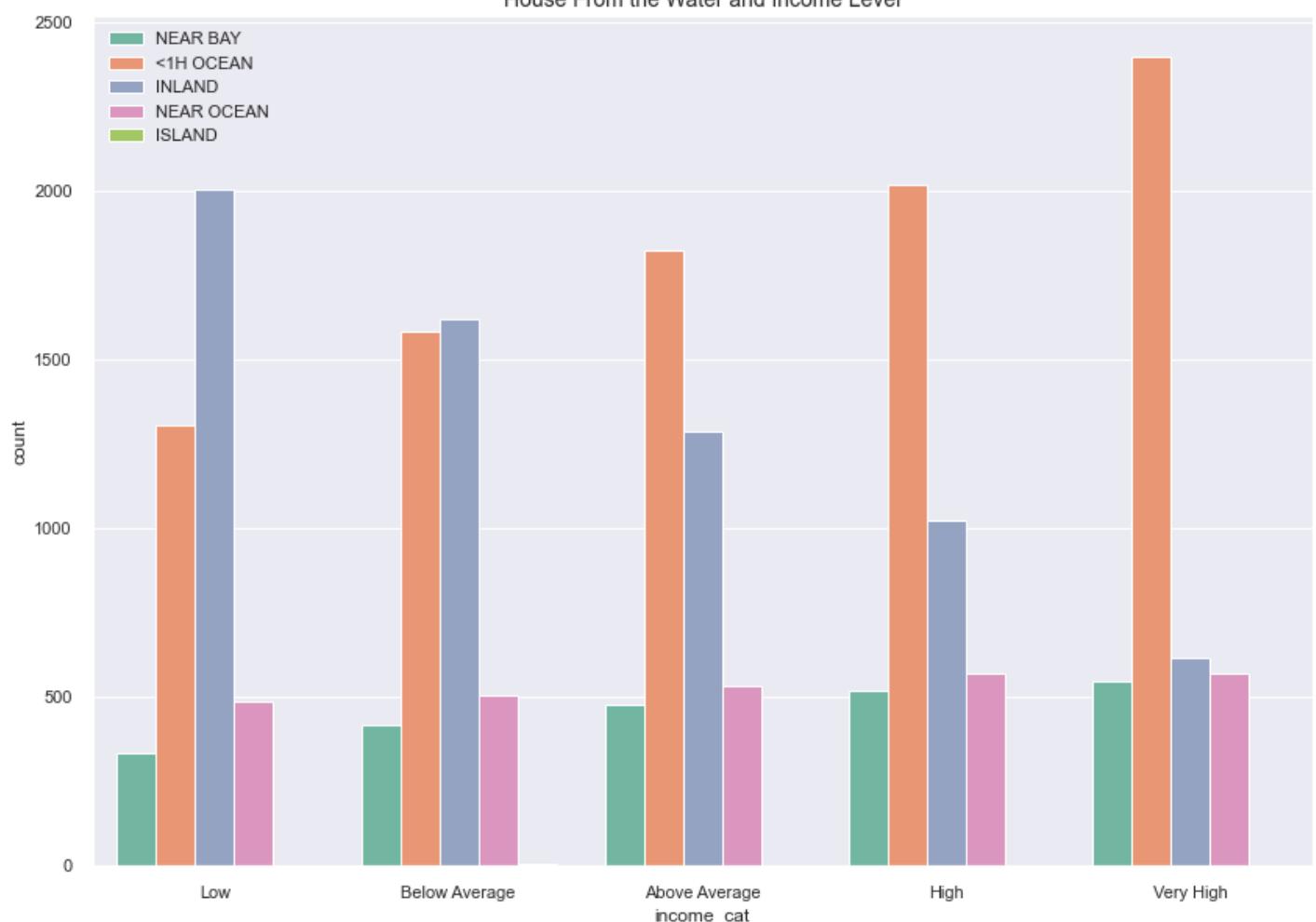
1. **Create** (and interpret) the following Seaborn **Countplots**:



In [158]:

```
plt.figure(figsize = (14, 10))
sns.set(font_scale = 1, palette = "Set2")
sns.countplot(data = df, x = "income_cat", hue = "ocean_proximity")
plt.title("House From the Water and Income Level", fontsize = 14)
plt.legend(loc = 2)
plt.show()
```

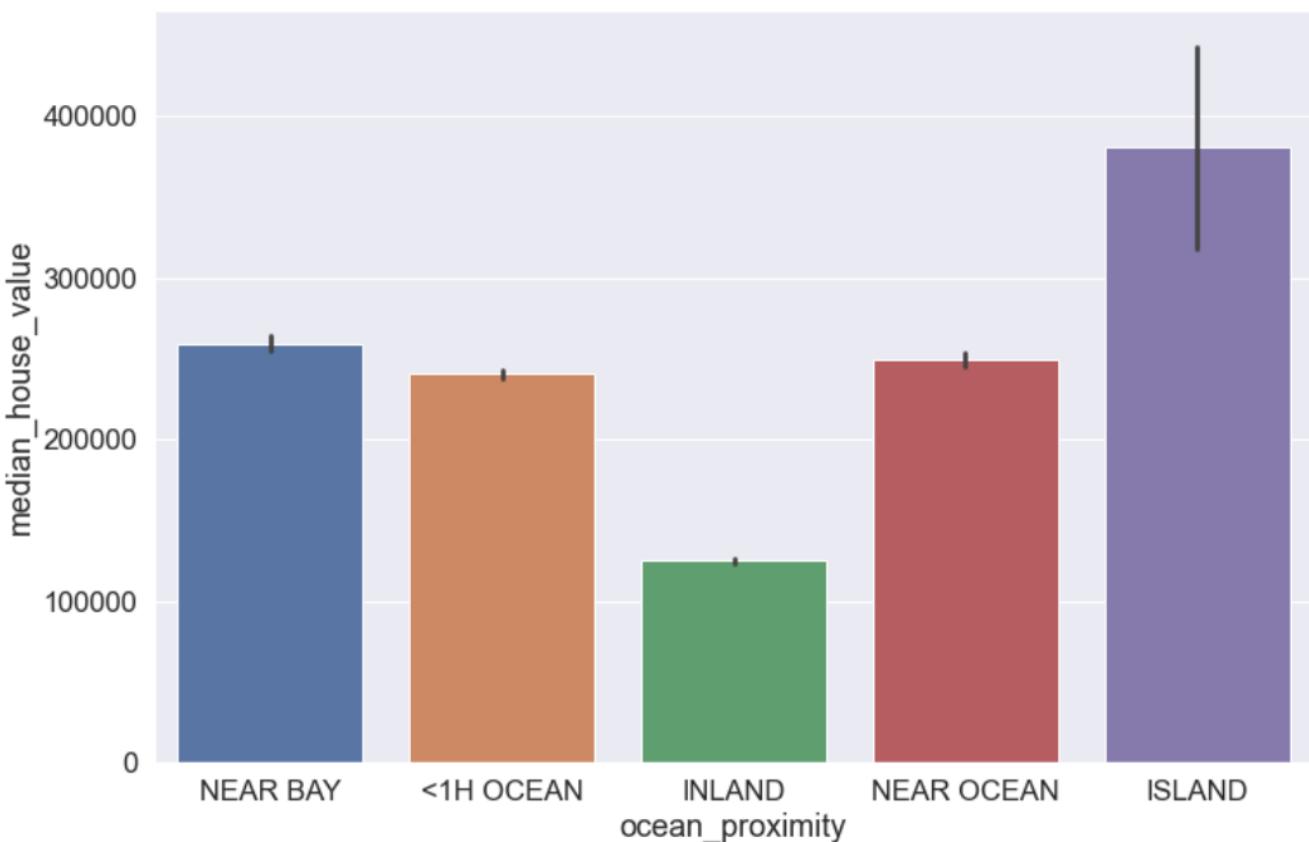
House From the Water and Income Level



In [103]:

```
plt.figure(figsize = (12, 8))
plt.title("Median House Value vs. Income")
sns.set(font_scale = 1.5)
```

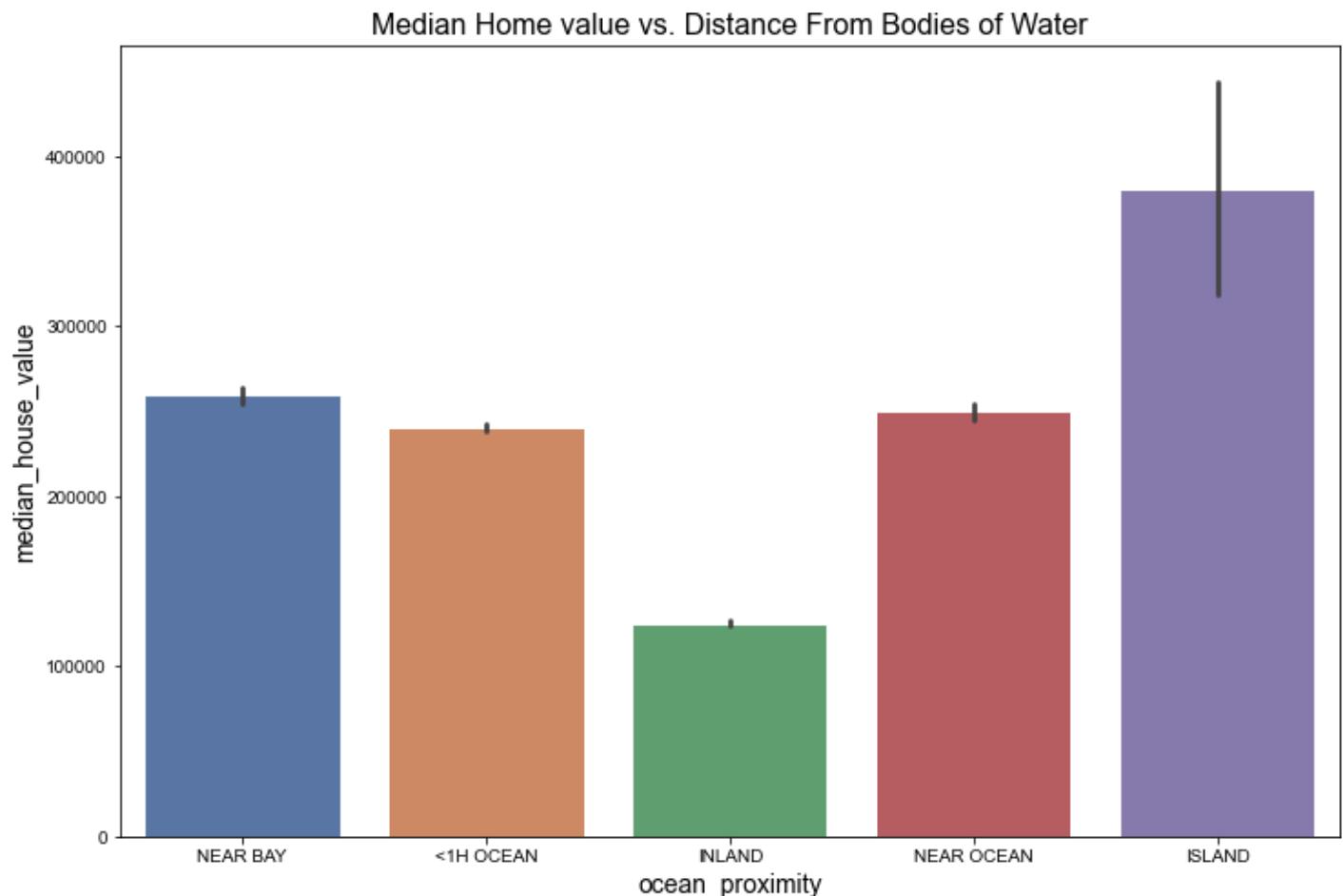
```
sns.barplot(data = df, x = df.income_cat, y = df.median_house_value)
plt.show()
```



In []:

```
In [95]: plt.rcParams.update(plt.rcParamsDefault)
%matplotlib inline
```

```
plt.figure(figsize = (12,8))
plt.title("Median Home value vs. Distance From Bodies of Water", fontsize = 16)
plt.xlabel("Ocean Proximity", fontsize = 14)
plt.ylabel("Median House Value", fontsize = 14)
plt.style.use("seaborn")
sns.set(font_scale = 1.5)
sns.barplot(data = df, x = "ocean_proximity", y = "median_house_value")
plt.show()
```



The black lines show the 95% confidence interval.

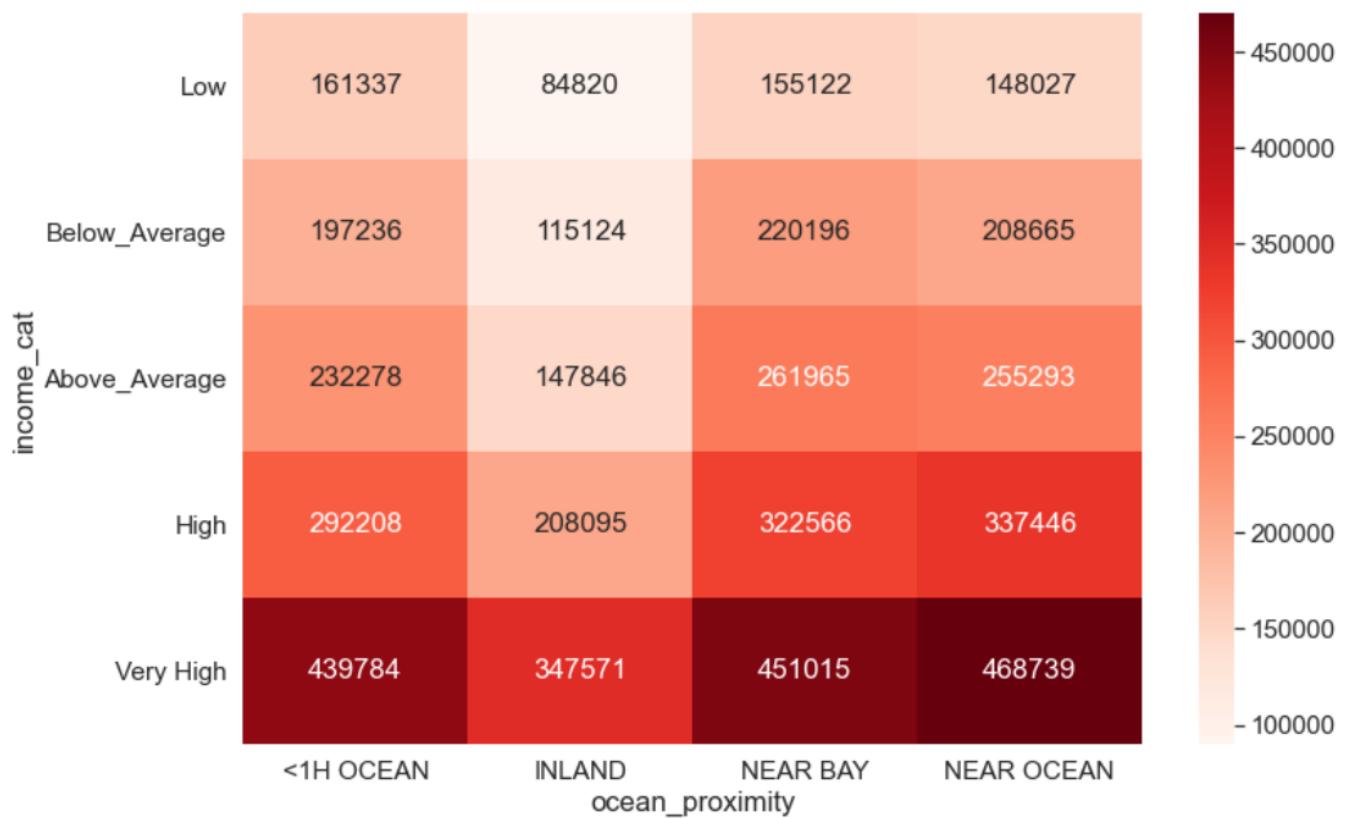
"We" are 95% confident that the real value falls somewhere on the bar or on the black line.

```
In [68]: plt.style.available
```

```
Out[68]: ['Solarize_Light2',
'_classic_test_patch',
'_mpl-gallery',
'_mpl-gallery-nogrid',
'bmh',
'classic',
'dark_background',
'fast',
'fivethirtyeight',
'ggplot',
'grayscale',
'seaborn',
'seaborn-bright',
```

```
'seaborn-colorblind',
'seaborn-dark',
'seaborn-dark-palette',
'seaborn-darkgrid',
'seaborn-deep',
'seaborn-muted',
'seaborn-notebook',
'seaborn-paper',
'seaborn-pastel',
'seaborn-poster',
'seaborn-talk',
'seaborn-ticks',
'seaborn-white',
'seaborn-whitegrid',
'tableau-colorblind10']
```

1. **Create** (and interpret) the following Seaborn **Heatmap** with mean house values for all combinations of `income_cat` & `ocean_proximity`:



Create a Matrix

Then can create a heatmap

In [159...]

```
# Island was dropped because there were only five values
matrix = df.groupby(["income_cat", "ocean_proximity"]).median_house_value.mean().unstack()
# CODE CONTINUED
# .median_house_value.mean().unstack().drop(columns = ["ISLAND"])
```

In [162...]

```
matrix.astype("int")
```

Out[162...]

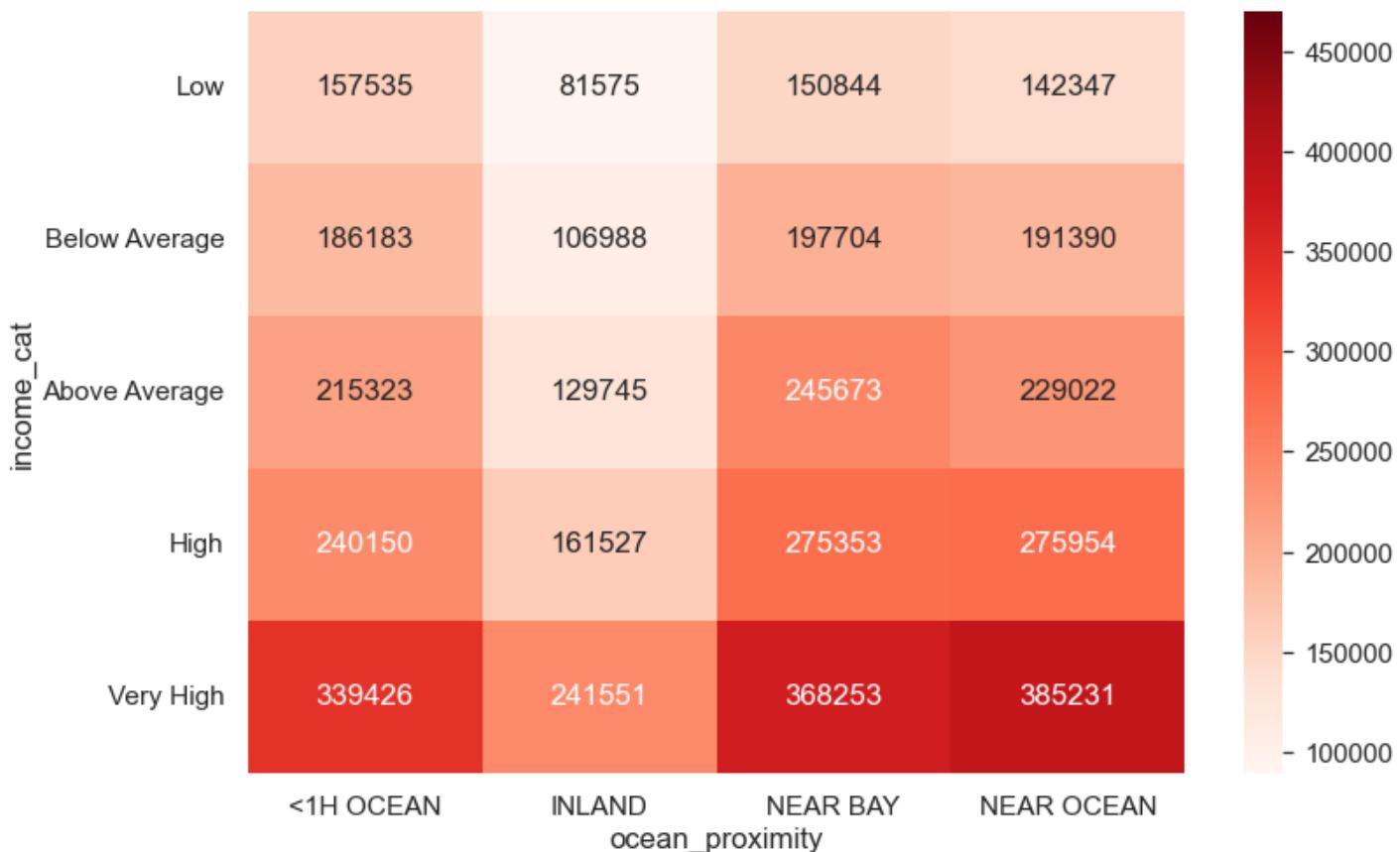
income_cat	<1H OCEAN	INLAND	NEAR BAY	NEAR OCEAN
Low	161337	84820	155122	148027
Below_Average	197236	115124	220196	208665
Above_Average	232278	147846	261965	255293
High	292208	208095	322566	337446
Very High	439784	347571	451015	468739

ocean_proximity	<1H OCEAN	INLAND	NEAR BAY	NEAR OCEAN
income_cat				
Low	157535	81575	150844	142347
Below Average	186183	106988	197704	191390
Above Average	215323	129745	245673	229022
High	240150	161527	275353	275954
Very High	339426	241551	368253	385231

```
matrix = df.groupby(["income_cat", "ocean_proximity"]).median_house_value.mean().unstack().drop(columns = ["ISLAND"])
```

In [164]:

```
plt.figure(figsize = (12, 8))
sns.set(font_scale = 1.4)
sns.heatmap(matrix.astype("int"), cmap = "Reds", annot = True,
            fmt = "d", vmin = 90000, vmax = 470000)
plt.show()
```



In []:

++++++ See some Hints below
++++++

In []:

In []:

In []:

In []:

++++++
Hints++++++

In []:

Hints for 10.

Use pd.qcut()

In []:

Web scraping - the Dow Jones Constituents

In [19]:

```
import pandas as pd
```

Use pd.read_html() to read websites

In [20]:

```
const = pd.read_html("https://en.wikipedia.org/wiki/Dow_Jones_Industrial_Average") [1]
const
```

Out[20]:

	Company	Exchange	Symbol	Industry	Date added	Notes	Index weighting
0	3M	NYSE	MMM	Conglomerate	1976-08-09	As Minnesota Mining and Manufacturing	3.02%
1	American Express	NYSE	AXP	Financial services	1982-08-30		3.60%
2	Amgen	NASDAQ	AMGN	Biopharmaceutical	2020-08-31		4.48%
3	Apple	NASDAQ	AAPL	Information technology	2015-03-19		3.25%
4	Boeing	NYSE	BA	Aerospace and defense	1987-03-12		3.96%
5	Caterpillar	NYSE	CAT	Construction and Mining	1991-05-06		3.74%
6	Chevron	NYSE	CVX	Petroleum industry	2008-02-19	Also 1930-07-18 to 1999-11-01	2.53%
7	Cisco	NASDAQ	CSCO	Information technology	2009-06-08		1.03%
8	Coca-Cola	NYSE	KO	Drink industry	1987-03-12	Also 1932-05-26 to 1935-11-20	1.15%
9	Disney	NYSE	DIS	Broadcasting and entertainment	1991-05-06		2.65%
10	Dow	NYSE	DOW	Chemical industry	2019-04-02		1.13%
11	Goldman Sachs	NYSE	GS	Financial services	2013-09-20		6.88%
12	Home Depot	NYSE	HD	Home Improvement	1999-11-01		6.71%
13	Honeywell	NASDAQ	HON	Conglomerate	2020-08-31	Also 1925-12-07 to 2008-02-19 under various na...	3.61%
14	IBM	NYSE	IBM	Information technology	1979-06-29	Also 1932-05-26 to 1939-03-04	2.55%
15	Intel	NASDAQ	INTC	Semiconductor industry	1999-11-01		0.91%
16	Johnson & Johnson	NYSE	JNJ	Pharmaceutical industry	1997-03-17		3.19%

	Company	Exchange	Symbol	Industry	Date added	Notes	Index weighting
17	JPMorgan Chase	NYSE	JPM	Financial services	1991-05-06	NaN	2.90%
18	McDonald's	NYSE	MCD	Food industry	1985-10-30	NaN	4.83%
19	Merck	NYSE	MRK	Pharmaceutical industry	1979-06-29	NaN	1.43%
20	Microsoft	NASDAQ	MSFT	Information technology	1999-11-01	NaN	5.66%
21	Nike	NYSE	NKE	Clothing industry	2013-09-20	NaN	2.67%
22	Procter & Gamble	NYSE	PG	Fast-moving consumer goods	1932-05-26	NaN	2.97%
23	Salesforce	NYSE	CRM	Information technology	2020-08-31	NaN	4.04%
24	Travelers	NYSE	TRV	Insurance	2009-06-08	NaN	3.20%
25	UnitedHealth	NYSE	UNH	Managed health care	2012-09-24	NaN	9.17%
26	Verizon	NYSE	VZ	Telecommunications industry	2004-04-08	NaN	0.98%
27	Visa	NYSE	V	Financial services	2013-09-20	NaN	4.23%
28	Walgreens Boots Alliance	NASDAQ	WBA	Retailing	2018-06-26	NaN	0.92%
29	Walmart	NYSE	WMT	Retailing	1997-03-17	NaN	2.56%

The code below gets all of the indexes but only the first six columns 0 to 5. Note that the numbers count as a column.

In [21]:

```
const = const.iloc[:, :5].copy()
const
```

Out[21]:

	Company	Exchange	Symbol	Industry	Date added
0	3M	NYSE	MMM	Conglomerate	1976-08-09
1	American Express	NYSE	AXP	Financial services	1982-08-30
2	Amgen	NASDAQ	AMGN	Biopharmaceutical	2020-08-31
3	Apple	NASDAQ	AAPL	Information technology	2015-03-19
4	Boeing	NYSE	BA	Aerospace and defense	1987-03-12
5	Caterpillar	NYSE	CAT	Construction and Mining	1991-05-06
6	Chevron	NYSE	CVX	Petroleum industry	2008-02-19
7	Cisco	NASDAQ	CSCO	Information technology	2009-06-08

	Company	Exchange	Symbol	Industry	Date added
8	Coca-Cola	NYSE	KO	Drink industry	1987-03-12
9	Disney	NYSE	DIS	Broadcasting and entertainment	1991-05-06
10	Dow	NYSE	DOW	Chemical industry	2019-04-02
11	Goldman Sachs	NYSE	GS	Financial services	2013-09-20
12	Home Depot	NYSE	HD	Home Improvement	1999-11-01
13	Honeywell	NASDAQ	HON	Conglomerate	2020-08-31
14	IBM	NYSE	IBM	Information technology	1979-06-29
15	Intel	NASDAQ	INTC	Semiconductor industry	1999-11-01
16	Johnson & Johnson	NYSE	JNJ	Pharmaceutical industry	1997-03-17
17	JPMorgan Chase	NYSE	JPM	Financial services	1991-05-06
18	McDonald's	NYSE	MCD	Food industry	1985-10-30
19	Merck	NYSE	MRK	Pharmaceutical industry	1979-06-29
20	Microsoft	NASDAQ	MSFT	Information technology	1999-11-01
21	Nike	NYSE	NKE	Clothing industry	2013-09-20
22	Procter & Gamble	NYSE	PG	Fast-moving consumer goods	1932-05-26
23	Salesforce	NYSE	CRM	Information technology	2020-08-31
24	Travelers	NYSE	TRV	Insurance	2009-06-08
25	UnitedHealth	NYSE	UNH	Managed health care	2012-09-24
26	Verizon	NYSE	VZ	Telecommunications industry	2004-04-08
27	Visa	NYSE	V	Financial services	2013-09-20
28	Walgreens Boots Alliance	NASDAQ	WBA	Retailing	2018-06-26
29	Walmart	NYSE	WMT	Retailing	1997-03-17

Must rename columns with spaces with:

```
dataframe.rename(columns = {"old_name": "new_name"}, inplace = True)
```

In [22]:

```
const.rename(columns = {"Date added": "Date_Added"}, inplace = True)
```

In [23]:

```
const.columns
```

Out[23]:

```
Index(['Company', 'Exchange', 'Symbol', 'Industry', 'Date_Added'], dtype='object')
```

Convert a string date to a datetime using: pd.to_datetime(df.column)

```
pd.to_datetime(df.column)
```

In [25]:

```
const.Date_Added = pd.to_datetime(const.Date_Added)
```

In [26]:

```
const.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype  
---  --  
 0   Company     30 non-null    object  
 1   Exchange    30 non-null    object  
 2   Symbol      30 non-null    object  
 3   Industry    30 non-null    object  
 4   Date_Added  30 non-null    datetime64[ns]
dtypes: datetime64[ns](1), object(4)
memory usage: 1.3+ KB
```

```
In [29]: const.Symbol[5]
```

```
Out[29]: 'CAT'
```

Importing Data With yFinance

```
In [30]: import pandas as pd
import yfinance as yf
```

Indexes tend to start with a "^" symbol.

From here you can download all of the historical data from the nineties.

```
In [34]: yf.download("^DJI")
```

```
[*****100%*****] 1 of 1 completed
```

```
Out[34]:
```

	Open	High	Low	Close	Adj Close	Volume
Date						
1992-01-02	3152.100098	3172.629883	3139.310059	3172.399902	3172.399902	23550000
1992-01-03	3172.399902	3210.639893	3165.919922	3201.500000	3201.500000	23620000
1992-01-06	3201.500000	3213.330078	3191.860107	3200.100098	3200.100098	27280000
1992-01-07	3200.100098	3210.199951	3184.479980	3204.800049	3204.800049	25510000
1992-01-08	3204.800049	3229.199951	3185.820068	3203.899902	3203.899902	29040000
...
2022-03-25	34702.390625	34942.699219	34631.519531	34861.238281	34861.238281	285440000
2022-03-28	34833.031250	34957.929688	34552.230469	34955.890625	34955.890625	299790000
2022-03-29	35114.351562	35372.261719	35030.070312	35294.191406	35294.191406	355050000
2022-03-30	35273.628906	35361.359375	35058.578125	35228.808594	35228.808594	317320000
2022-03-31	35201.519531	35201.519531	34677.988281	34678.351562	34678.351562	438355525

7620 rows × 6 columns

Use `yf.download("^DJI", start = "YYYY-MM-DD", end = YYYY-`

MM-DD")

This can be used to isolate a certain date range.

```
In [36]: dji = yf.download("^DJI", start = "2007-01-01", end = "2020-03-31")  
[*****100%*****] 1 of 1 completed
```

```
In [37]: dji
```

```
Out[37]:
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2007-01-03	12459.540039	12580.349609	12404.820312	12474.519531	12474.519531	327200000
2007-01-04	12473.160156	12510.410156	12403.860352	12480.690430	12480.690430	259060000
2007-01-05	12480.049805	12480.129883	12365.410156	12398.009766	12398.009766	235220000
2007-01-08	12392.009766	12445.919922	12337.370117	12423.490234	12423.490234	223500000
2007-01-09	12424.769531	12466.429688	12369.169922	12416.599609	12416.599609	225190000
...
2020-03-24	19722.189453	20737.699219	19649.250000	20704.910156	20704.910156	799340000
2020-03-25	21050.339844	22019.929688	20538.339844	21200.550781	21200.550781	796320000
2020-03-26	21468.380859	22595.060547	21427.099609	22552.169922	22552.169922	705180000
2020-03-27	21898.470703	22327.570312	21469.269531	21636.779297	21636.779297	588830000
2020-03-30	21678.220703	22378.089844	21522.080078	22327.480469	22327.480469	545540000

3333 rows × 6 columns

```
In [38]: dji.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
DatetimeIndex: 3333 entries, 2007-01-03 to 2020-03-30  
Data columns (total 6 columns):  
 #  Column      Non-Null Count  Dtype     
---  --          --          --  
 0   Open        3333 non-null   float64  
 1   High        3333 non-null   float64  
 2   Low         3333 non-null   float64  
 3   Close       3333 non-null   float64  
 4   Adj Close   3333 non-null   float64  
 5   Volume      3333 non-null   int64  
dtypes: float64(5), int64(1)  
memory usage: 182.3 KB
```

The Index can be saved in a CSV file

```
In [46]: dji.to_csv("diji.csv")
```

```
In [42]: ticker_list = const.Symbol.to_list()
```

```
In [47]: prices = yf.download(ticker_list, start = "2007-01-01", end = "2020-03-31")
```

```
[*****100%*****] 30 of 30 completed
```

```
In [48]: prices
```

```
Out[48]:
```

	AAPL	AMGN	AXP	BA	CAT	CRM	CSCO	CVX	DIS	Adj Close
Date										
2007-01-03	2.562707	52.442799	47.476803	64.405731	40.306820	9.017500	20.191025	39.532825	28.317097	
2007-01-04	2.619588	54.689247	47.130733	64.665741	40.201378	9.470000	20.722557	39.148460	28.540648	
2007-01-05	2.600933	54.819580	46.509335	64.391281	39.687321	9.880000	20.729841	39.298874	28.308815	
2007-01-08	2.613778	54.382553	46.949810	64.239601	39.733456	9.982500	20.846331	39.800198	28.565495	
2007-01-09	2.830904	54.643246	46.650921	63.560658	39.950935	9.990000	20.729841	39.343433	28.524092	
...
2020-03-24	60.883400	190.709641	81.702164	127.680000	96.570740	153.639999	36.307018	59.915062	98.120003	25.71
2020-03-25	60.547997	182.038452	87.651199	158.729996	99.744034	147.059998	35.432262	62.363873	100.730003	27.42
2020-03-26	63.734230	187.080948	90.684052	180.550003	105.299660	154.729996	38.169399	68.765007	105.360001	26.81
2020-03-27	61.095474	186.873596	86.251427	162.000000	100.477783	146.000000	36.513954	61.922729	96.400002	25.79
2020-03-30	62.839016	196.496704	87.748405	152.279999	106.452721	149.850006	37.924847	64.776672	99.800003	25.88

3333 rows × 180 columns

```
In [49]: prices.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3333 entries, 2007-01-03 to 2020-03-30
Columns: 180 entries, ('Adj Close', 'AAPL') to ('Volume', 'WMT')
dtypes: float64(152), int64(28)
memory usage: 4.6 MB
```

Got the fresh set of data from importing by making yf.download(tickers = ticker_list)

```
In [51]: prices = prices.loc[:, "Close"].copy()
```

```
In [52]: prices
```

Out[52]:

AAPL AMGN AXP BA CAT CRM CSCO CVX DIS

Date	AAPL	AMGN	AXP	BA	CAT	CRM	CSCO	CVX	DIS
2007-01-03	2.992857	68.400002	60.360001	89.169998	61.160000	9.017500	27.730000	70.970001	33.738300
2007-01-04	3.059286	71.330002	59.919998	89.529999	61.000000	9.470000	28.459999	70.279999	34.004654
2007-01-05	3.037500	71.500000	59.130001	89.150002	60.220001	9.880000	28.469999	70.550003	33.728436
2007-01-08	3.052500	70.930000	59.689999	88.940002	60.290001	9.982500	28.629999	71.449997	34.034248
2007-01-09	3.306071	71.269997	59.310001	88.000000	60.619999	9.990000	28.469999	70.629997	33.984924
...
2020-03-24	61.720001	202.339996	84.050003	127.680000	101.339996	153.639999	38.599998	66.550003	98.120003
2020-03-25	61.380001	193.139999	90.169998	158.729996	104.669998	147.059998	37.669998	69.269997	100.730003
2020-03-26	64.610001	198.490005	93.290001	180.550003	110.500000	154.729996	40.580002	76.379997	105.360001
2020-03-27	61.935001	198.270004	88.730003	162.000000	105.440002	146.000000	38.820000	68.779999	96.400002
2020-03-30	63.702499	208.479996	90.269997	152.279999	111.709999	149.850006	40.320000	71.949997	99.800003

3333 rows × 30 columns

In [53]:

prices.info()

```

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3333 entries, 2007-01-03 to 2020-03-30
Data columns (total 30 columns):
 #   Column  Non-Null Count  Dtype  
--- 
 0   AAPL    3333 non-null   float64
 1   AMGN    3333 non-null   float64
 2   AXP     3333 non-null   float64
 3   BA     3333 non-null   float64
 4   CAT    3333 non-null   float64
 5   CRM    3333 non-null   float64
 6   CSCO   3333 non-null   float64
 7   CVX    3333 non-null   float64
 8   DIS    3333 non-null   float64
 9   DOW    260  non-null   float64
 10  GS     3333 non-null   float64
 11  HD     3333 non-null   float64
 12  HON    3333 non-null   float64
 13  IBM    3333 non-null   float64
 14  INTC   3333 non-null   float64
 15  JNJ    3333 non-null   float64
 16  JPM    3333 non-null   float64
 17  KO     3333 non-null   float64
 18  MCD    3333 non-null   float64
 19  MMM    3333 non-null   float64
 20  MRK    3333 non-null   float64

```

```
21  MSFT      3333 non-null    float64
22  NKE       3333 non-null    float64
23  PG        3333 non-null    float64
24  TRV       3333 non-null    float64
25  UNH       3333 non-null    float64
26  V         3029 non-null    float64
27  VZ        3333 non-null    float64
28  WBA       3333 non-null    float64
29  WMT       3333 non-null    float64
dtypes: float64(30)
memory usage: 807.2 KB
```

```
In [54]: prices.to_csv("const_prices.csv")
```

```
In [ ]:
```