

PYTHON FOR DATA SCIENCE AND MACHINE LEARNING BOOTCAMP

NUMPY

--> Numpy arrays come in two essential flavors--Matrices and Vectors--> Vectors are strictly 1-d Array and Matrices are 2-D arrays (but note that matrices can still have only one row and only one column)--> It is also a type of array--> Vectors are called as 1-D array and Matrices are called as 2-D array

```
In [2]: #Numpy is the module for numpy array
import numpy as np
```

```
In [1]: #Creating a list
my_list=[1,2,3]
```

```
In [2]: #Printing the list
my_list
```

```
Out[2]: [1, 2, 3]
```

```
In [7]: #Creating an array of lists with numpy
arr=np.array(my_list)
```

```
In [9]: arr          # One-dimensional Array
```

```
Out[9]: array([1, 2, 3])
```

```
In [10]: #Creating a dual lists
my_list1=[[1,2,3],[3,4,5],[7,8,9]]
```

```
In [11]: #Printing the dual Lists
my_list1
```

```
Out[11]: [[1, 2, 3], [3, 4, 5], [7, 8, 9]]
```

```
In [12]: #Creating a Two-Dimensional Lists of arrays
arr1=np.array(my_list1)
```

```
In [14]: arr1          #Printing the two dimensional array
```

```
Out[14]: array([[1, 2, 3],
[3, 4, 5],
[7, 8, 9]])
```

--> Numpy arrays are faster and simpler to Create

```
In [19]: #Creating a numpy array
arr2=np.arange(0,10)
#Parameters arange(start,stop,,stepsize,dtype)
```

```
In [20]: arr2
```

```
Out[20]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [7]: #Creating a numpy array
```

```
#Stepping over into two for even values
arr3=np.arange(0,10,2)
#Parameters arange(start,stop,,stepsize,dtype)
```

In [8]: # Printing the stepsize values
arr3

Out[8]: array([0, 2, 4, 6, 8])

In [9]: #Creating zeros with numpy arrays
arr4=np.zeros(3)

In [11]: #Printing the zeros values
arr4

Out[11]: array([0., 0., 0.])

In [3]: # In this (rows,columns)
arr5=np.zeros((5,4))

In [4]: #Printing the array
arr5

Out[4]: array([[0., 0., 0., 0.],
 [0., 0., 0., 0.],
 [0., 0., 0., 0.],
 [0., 0., 0., 0.],
 [0., 0., 0., 0.]])

In [7]: # In this (rows,columns)
arr6=np.ones((4,5))
#It is two dimensional array

In [8]: #Printing the array
arr6

Out[8]: array([[1., 1., 1., 1., 1.],
 [1., 1., 1., 1., 1.],
 [1., 1., 1., 1., 1.],
 [1., 1., 1., 1., 1.]])

In [10]: #One-Dimensional Array
arr7=np.ones(5)
#Printing the array
arr7

Out[10]: array([1., 1., 1., 1., 1.])

In [11]: #It shows one dimensional vector with 5 evenly distributed spaced points
arr8=np.linspace(0,10,5)

In [12]: #Printing the array
arr8

Out[12]: array([0. , 2.5, 5. , 7.5, 10.])

In [17]: #It is one dimensional arry as it has only one set of squared brackets

```
arr9=np.linspace(0,10,15) #It evenly splits up the numbers
```

In [18]: *#Printing the array*
arr9

Out[18]: array([0. , 0.71428571, 1.42857143, 2.14285714, 2.85714286, 3.57142857, 4.28571429, 5. , 5.71428571, 6.42857143, 7.14285714, 7.85714286, 8.57142857, 9.28571429, 10.])

In [21]: *#Creating identity martices*
arr10=np.eye(4)
#Printing the array
arr10
#Here the diagonal line has number 1 and all other is 0

Out[21]: array([[1., 0., 0., 0.],
[0., 1., 0., 0.],
[0., 0., 1., 0.],
[0., 0., 0., 1.]])

In [22]: *#Creating random number in numpy arrays*
#It is one dimensional arrays
arr11=np.random.rand(5)
#Printing the array
arr11

Out[22]: array([0.53924685, 0.76509223, 0.95346931, 0.53751853, 0.57042947])

In [23]: *#Creating an random numbers with two dimensional arrays in numpy*
arr12=np.random.rand(5,4)
#Printing the array
arr12

Out[23]: array([[0.89832883, 0.53849176, 0.44245585, 0.89450312],
[0.34751088, 0.38682936, 0.5769906 , 0.03787814],
[0.23416593, 0.95103604, 0.8719166 , 0.16417138],
[0.33565192, 0.82983238, 0.05070193, 0.743674],
[0.68730645, 0.54324987, 0.37666903, 0.17548291]])

In [24]: *#Creating an random numbers with randn from random module*
arr13=np.random.randn(2)

In [25]: *#Printing the array*
arr13

Out[25]: array([-0.27401975, 1.35886071])

In [26]: *#Returning random numbers from gaussian random distribution or standard normal distribut*
arr14=np.random.randn(3)
#Printing the array
arr14

Out[26]: array([0.01286826, -0.73428979, 0.32992708])

In [29]: *#Printing random integers with randint in numpy*
arr15=np.random.randint(1,100)
#Printing the array
arr15

Out[29]: 72

In [40]: *#Printing random integers with randint with step over of given numbers in numpy
#It prints 10 different random numbers*
`arr16=np.random.randint(0,50,10)`
#Printing the array
`arr16`

Out[40]: `array([39, 17, 21, 44, 41, 40, 29, 2, 11, 28])`

In [42]: `arr17=np.arange(25)`
#Printing the array
`arr17`

Out[42]: `array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
 17, 18, 19, 20, 21, 22, 23, 24])`

In [43]: *#Respaing the array according to given rows and columns*
`arr17.reshape(5,5)`

Out[43]: `array([[0, 1, 2, 3, 4],
 [5, 6, 7, 8, 9],
 [10, 11, 12, 13, 14],
 [15, 16, 17, 18, 19],
 [20, 21, 22, 23, 24]])`

In [44]: *#Getting maximum value*
`arr17.max()`

Out[44]: 24

In [45]: *#Getting minimum value*
`arr17.min()`

Out[45]: 0

In [48]: *#Returing the index of the max value*
`arr17.argmax()`

Out[48]: 24

In [50]: *#Returning the index of the min value*
`arr17.argmin()`

Out[50]: 0

In [53]: *#Printing the shape of the given array
#here the given array is one-Dimensional array*
`arr17.shape`

Out[53]: (25,)

In [57]: *#Reshaping it into 5,5*
`arr18=arr17.reshape(5,5)`

In [60]: *#Viewing the reshape*
`arr18.shape`

```
Out[60]: (5, 5)
```

```
In [61]: #printing the data type of the array
arr18.dtype
```

```
Out[61]: dtype('int32')
```

```
In [62]: #Method of importing only randint
from numpy.random import randint
```

```
In [63]: #Printing the random number
randint(2,10)
```

```
Out[63]: 3
```

NUMPY INDEXING AND SELECTION

```
In [64]: #Getting a range of numbers
arr19=np.arange(0,11)
```

```
In [65]: #Printing the numbers
arr19
```

```
Out[65]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
In [67]: #Get the value of the given index of the array
arr19[4]
```

```
Out[67]: 4
```

```
In [69]: #Index slicing with [start:stop:stepover]
arr19[1:5]
```

```
Out[69]: array([1, 2, 3, 4])
```

```
In [70]: #Indexing with stepover
arr19[1:11:2]
```

```
Out[70]: array([1, 3, 5, 7, 9])
```

```
In [71]: #printing the number untill 6
arr19[:6]
```

```
Out[71]: array([0, 1, 2, 3, 4, 5])
```

```
In [73]: #Printing the number from 3
arr19[3:]
```

```
Out[73]: array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

```
In [74]: #Broad casting the array i,e:changing the number to 100 from start index to stop indexe
arr19[0:5]=100
```

```
In [75]: #Printing the array
```

arr19

Out[75]: array([100, 100, 100, 100, 100, 5, 6, 7, 8, 9, 10])

In [76]: #Creating the number from 0 to 9
arr20=np.arange(10)

In [77]: arr20

Out[77]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [80]: #Slicing everything from the array i,e:Getting all the numbers
arr20[:]

Out[80]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [83]: #copy function is used to copy all the details from arr20 to arr21
arr21=arr20.copy()In [84]: #Printing the array
arr21

Out[84]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [87]: #Creating an 2d-array which is like matrix
arr21_in_2d=np.array([[5,10,15],[20,25,30],[35,40,45]])In [88]: #Printing the array
arr21_in_2dOut[88]: array([[5, 10, 15],
[20, 25, 30],
[35, 40, 45]])

Two-Dimensional array indexing

In [90]: #Getting the value from the given indexes
#Method 1:
arr21_in_2d[0][0]

Out[90]: 5

In [93]: #Getting the value from the given indexes
arr21_in_2d[1][2]

Out[93]: 30

In [94]: #Getting the value from the given indexes
#Method 2: This is called as slice notations
arr21_in_2d[1,2]

Out[94]: 30

In [101...]: #Grabbing the 1 st row to 2 nd row and first column to second column
#array[row_start:row_stop,column_start:column_stop]
arr21_in_2d[:,2,1:]

```
Out[101... array([[10, 15],
 [25, 30]])
```

```
In [102... arr22=np.arange(1,11)
```

```
In [103... #Printing the array
arr22
```

```
Out[103... array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
In [108... # Comparision with boolean and with the array
#It shows the data in Boolean(True or False) by comparing the logics
boolean_array22=arr22 >=5
```

```
In [112... #Printing the values
boolean_array22
```

```
Out[112... array([False, False, False, False, True, True, True, True, True,
 True])
```

```
In [111... #We can view the values the condition values are true
#Method 1:
arr22[boolean_array22]
```

```
Out[111... array([ 5,  6,  7,  8,  9, 10])
```

```
In [114... #We can view the values the condition values are true
#Method 2:
arr22[arr22>=5]
```

```
Out[114... array([ 5,  6,  7,  8,  9, 10])
```

```
In [116... #Printing the array number less than 4
arr22[arr22<4]
```

```
Out[116... array([1, 2, 3])
```

```
In [118... #Creating the numbers with range of 0 to 49 and arranged into the shape of 5 rows and 10 columns
arr_2d=np.arange(50).reshape(5,10)
```

```
In [119... #Printing the array
arr_2d
```

```
Out[119... array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
 [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
 [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
 [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
 [40, 41, 42, 43, 44, 45, 46, 47, 48, 49]])
```

```
In [130... #Indexing the array into row_start:row_stop,column_start:column_stop
arr_2d[1:3,3:5]
```

```
Out[130... array([[13, 14],
 [23, 24]])
```

NUMPY OPERATIONS

```
In [131... #Creating a range of the numbers
```

```
arr23=np.arange(11)
```

In [132...]: #Printing the array
arr23

Out[132...]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

In [135...]: arr24=arr23.copy()

In [136...]: #Printing the array
arr24

Out[136...]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

In [137...]: # Adding two arrays
arr24+arr23

Out[137...]: array([0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20])

In [138...]: #Subtracting two arrays
arr24-arr23

Out[138...]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

In [139...]: #Multiplying two arrays
arr24*arr23

Out[139...]: array([0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100])

In [140...]: #Dividing two arrays
#We get nan when we divide 0/0
arr24/arr23

<ipython-input-140-4b258d170573>:2: RuntimeWarning: invalid value encountered in true_divide
arr24/arr23

Out[140...]: array([nan, 1., 1., 1., 1., 1., 1., 1., 1., 1.])

In [142...]: #Adding arrays with 100
arr24+100

Out[142...]: array([100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110])

In [144...]: #We get infinity(inf) if we divide 1/arr
1/arr24

<ipython-input-144-82b937e203f0>:2: RuntimeWarning: divide by zero encountered in true_divide
1/arr24

Out[144...]: array([inf, 1. , 0.5 , 0.33333333, 0.25 ,
 0.2 , 0.16666667, 0.14285714, 0.125 , 0.11111111,
 0.1])

In [145...]: #Getting the squares values
arr24 ** 2

```
Out[145... array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81, 100], dtype=int32)
```

```
In [147... #Getting the squareroot of th enumbers in the arrays  
np.sqrt(arr24)
```

```
Out[147... array([0. , 1. , 1.41421356, 1.73205081, 2. ,  
2.23606798, 2.44948974, 2.64575131, 2.82842712, 3. ,  
3.16227766])
```

```
In [149... #Getting the exponent value of the given array  
np.exp(arr24)
```

```
Out[149... array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,  
5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,  
2.98095799e+03, 8.10308393e+03, 2.20264658e+04])
```

```
In [152... #Getting the maximum value of the array  
np.max(arr24)
```

```
Out[152... 10
```

```
In [154... #Getting the minimum value of the array  
np.min(arr24)
```

```
Out[154... 0
```

```
In [156... #Getting the sin value  
np.sin(arr24)
```

```
Out[156... array([ 0. , 0.84147098, 0.90929743, 0.14112001, -0.7568025 ,  
-0.95892427, -0.2794155 , 0.6569866 , 0.98935825, 0.41211849,  
-0.54402111])
```

```
In [157... #Getting cos value  
np.cos(arr24)
```

```
Out[157... array([ 1. , 0.54030231, -0.41614684, -0.9899925 , -0.65364362,  
0.28366219, 0.96017029, 0.75390225, -0.14550003, -0.91113026,  
-0.83907153])
```

```
In [160... #Getting Log value  
np.log(arr24)
```

```
<ipython-input-160-f68a81cd2e7e>:2: RuntimeWarning: divide by zero encountered in log  
np.log(arr24)
```

```
Out[160... array([-inf, 0. , 0.69314718, 1.09861229, 1.38629436,  
1.60943791, 1.79175947, 1.94591015, 2.07944154, 2.19722458,  
2.30258509])
```

```
In [161... #Getting the standard deviation value  
np.std(arr24)
```

```
Out[161... 3.1622776601683795
```

```
In [162... #Getting the sum of th earary  
np.sum(arr24)
```

```
Out[162... 55
```

PANDAS

```
In [ ]: -->Pandas is an open source library built on top of numpy
-->It allows for fast analysis and data cleaning and
    preparation
-->It has built-in visualization features
-->It can work with data from a wide variety of sources
```

```
In [50]: #Importing required modules
import numpy as np
import pandas as pd
import random
```

```
In [51]: #Creating a lists named as Labels
labels=['a','b','b']
my_data=[10,20,30]
#Crating an array with the list data
arr25=np.array(my_data)
#Creating an dictionary
d={'a':10,'b':20,'c':30}
```

Pandas->Series

```
In [167...]: #Series is used to view the datas in different types
pd.Series(data=my_data)
```

```
Out[167...]: 0    10
1    20
2    30
dtype: int64
```

```
In [170...]: #Displaying the series with the given indexes
arr26=pd.Series(data=my_data,index=labels)
#Therefore with this Labels we can call the values
```

```
In [172...]: #Displaying the array
arr26
```

```
Out[172...]: a    10
b    20
b    30
dtype: int64
```

```
In [174...]: #Displaying the series with the given indexes
pd.Series(my_data,labels)
```

```
Out[174...]: a    10
b    20
b    30
dtype: int64
```

```
In [177...]: #Executing the Series with the numpy array
#Here the data type is int 32
pd.Series(arr25,labels)
```

```
Out[177...]: a    10
b    20
b    30
dtype: int32
```

```
In [182...]: #Printing the dictionary  
d
```

```
Out[182...]: {'a': 10, 'b': 20, 'c': 30}
```

```
In [181...]: #Passing a dictionary in Series  
pd.Series(d)
```

```
Out[181...]: a    10  
b    20  
c    30  
dtype: int64
```

```
In [52...]: #Printing the labels  
labels
```

```
Out[52...]: ['a', 'b', 'b']
```

```
In [184...]: #We can also pass the string in the pandas Series  
pd.Series(labels)
```

```
Out[184...]: 0    a  
1    b  
2    b  
dtype: object
```

```
In [186...]: #Creating an series of given data  
ser1=pd.Series([1,2,3,4,5],["Usa","Germany","France","Japan","China"])
```

```
In [188...]: #Printing the series  
ser1
```

```
Out[188...]: Usa      1  
Germany  2  
France   3  
Japan    4  
China    5  
dtype: int64
```

```
In [203...]: #Creating an another series  
#Series([data],[labels])  
ser2=pd.Series([1,2,5,4],["Usa","Germany","Italy","Japan"])
```

```
In [192...]: #Printing the series  
ser2
```

```
Out[192...]: Usa      1  
Germany  2  
Italy    5  
Japan    4  
dtype: int64
```

```
In [194...]: #Indexing the series value  
ser1["Usa"]
```

```
Out[194...]: 1
```

```
In [197...]: #Creating a series and making the data as labels  
ser3=pd.Series(data=labels)
```

ser3

```
Out[197... 0    a
          1    b
          2    b
         dtype: object
```

```
In [199]: #Printing the index value
           ser3[0]
```

```
Out[199... 'a'
```

```
In [202]: #Adding up two series
           #It adds up if there is a match else it will show nan
           #Note that it does not print the value if the string is not there
           #in the series it prints nan
           #For Example china is in ser1 and not there in ser2
           #in ser1 china's value is 5,Even though it is 5 it prints nan
           ser1+ser2
           #Also it is converted into floats
```

```
Out[202...   China      NaN
             France     NaN
             Germany    4.0
             Italy       NaN
             Japan      8.0
             Usa        2.0
            dtype: float64
```

Pandas-->DataFrames

```
In [42]: #Importing all required modules
           import numpy as np
           import pandas as pd

           from numpy.random import randn
```

```
In [4]: #This is used to get order by order random numbers
           np.random.seed(101)
```

```
In [5]: #Creating an dataframe with random numbers
           #Here ABCDE are row indexex and WXYZ are columns indexes
           #The numbers printed are random numbers
           df=pd.DataFrame(randn(5,4),['A','B','C','D','E'],['W','X','Y','Z'])
```

```
In [211]: #Printing the df
           df
```

```
Out[211...      W         X         Y         Z
          A  2.706850  0.628133  0.907969  0.503826
          B  0.651118 -0.319318 -0.848077  0.605965
          C -2.018168  0.740122  0.528813 -0.589001
          D  0.188695 -0.758872 -0.933237  0.955057
          E  0.190794  1.978757  2.605967  0.683509
```

In [213...]: #Grabbing the column values
df['W']

Out[213...]: A 2.706850
B 0.651118
C -2.018168
D 0.188695
E 0.190794
Name: W, dtype: float64

In [216...]: #In dataframe indexing first is column and then row
df['W']['A']

Out[216...]: 2.706849839399938

In [219...]: #Getting the type
#So W is a Series
type(df['W'])

Out[219...]: pandas.core.series.Series

In [220...]: #Here df is the type of DataFrame
type(df)

Out[220...]: pandas.core.frame.DataFrame

In [222...]: #Method 2 of indexing it is normally used in sql
df.W

Out[222...]: A 2.706850
B 0.651118
C -2.018168
D 0.188695
E 0.190794
Name: W, dtype: float64

In [224...]: #Getting more than one columns we use two squared brackets to print
df[['W','Z']]

Out[224...]:

	W	Z
A	2.706850	0.503826
B	0.651118	0.605965
C	-2.018168	-0.589001
D	0.188695	0.955057
E	0.190794	0.683509

In [7]: #Creating a new column
df["A"] = df["W"] + df["Y"]

In [8]: #Printing the df
df

Out[8]:

	W	X	Y	Z	A
--	----------	----------	----------	----------	----------

	W	X	Y	Z	A
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	-2.018168	0.740122	0.528813	-0.589001	-1.489355
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542
E	0.190794	1.978757	2.605967	0.683509	2.796762

In [9]: `#Deleting the column in the dataframes
#Here putting axis is very important to specify
#it as column
df.drop("A",axis=1)`

Out[9]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

In [11]: `#Even though if we drop the "A" column we still get A
df`

Out[11]:

	W	X	Y	Z	A
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	-2.018168	0.740122	0.528813	-0.589001	-1.489355
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542
E	0.190794	1.978757	2.605967	0.683509	2.796762

In [12]: `#The solution is to use inplace as true
df.drop("A",axis=1,inplace=True)`

In [14]: `#now we see that it disappears
df`

Out[14]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

```
In [15]: #Dropping the rows in the dataframe df
df.drop("E",axis=0,inplace=True)
```

```
In [16]: #removing the "E" row
df
```

```
Out[16]:
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057

Remember that axis=0 means rows axis=1 means columns

```
In [20]: df.shape
#(0 axis,1 axis)-->(row,columns)
```

```
Out[20]: (4, 4)
```

```
In [22]: #Getting only the required and given columns
df[["Z","X"]]
```

```
Out[22]:
```

	Z	X
A	0.503826	0.628133
B	0.605965	-0.319318
C	-0.589001	0.740122
D	0.955057	-0.758872

```
In [24]: #There are two methods to get the rows in data frame
#Method1:
#Here in this method we use loc if we have the string in row
df.loc ["A"]
```

```
Out[24]: W    2.706850
         X    0.628133
         Y    0.907969
         Z    0.503826
Name: A, dtype: float64
```

```
In [28]: #Method 2:
#We use this method to get data using the index method
df.iloc[2]
```

```
Out[28]: W    -2.018168
         X    0.740122
         Y    0.528813
         Z    -0.589001
Name: C, dtype: float64
```

```
In [29]: #index slicing in the dataframe
#getting the value of "B"th row and "Y" columns
```

```
df.loc["B","Y"]
```

Out[29]: -0.8480769834036315

In [32]: #getting the value with slicing of [[row_start, row_stop],[column_start, column_stop]]

```
df.loc[[ "A", "B"], [ "W", "Y"]]
```

Out[32]:

	W	Y
A	2.706850	0.907969
B	0.651118	-0.848077

In [33]: #Printing the df

```
df
```

Out[33]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057

In [37]: #To get a boolean result in the dataframe

```
bool_df=df>0
```

In [38]: #Prinitng the result in boolean result

```
bool_df
```

Out[38]:

	W	X	Y	Z
A	True	True	True	True
B	True	False	False	True
C	False	True	True	False
D	True	False	False	True

In [40]: #Here we get the values if it is True and it prints Nan if it is False

```
df[bool_df]
```

Out[40]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	NaN	NaN	0.605965
C	NaN	0.740122	0.528813	NaN
D	0.188695	NaN	NaN	0.955057

In [44]: #getting an boolean value with particular given row

```
df[ "W"]>0
```

```
Out[44]: A      True
          B      True
          C     False
          D      True
Name: W, dtype: bool
```

In [45]: *#It prints only the true values and does not print false values*
`df[df["W"]>0]`

```
Out[45]:    W      X      Y      Z
          A  2.706850  0.628133  0.907969  0.503826
          B  0.651118 -0.319318 -0.848077  0.605965
          D  0.188695 -0.758872 -0.933237  0.955057
```

In [46]: *#It will return the values of the (C) rows and all columns*
`df[df["Z"]<0]`

```
Out[46]:    W      X      Y      Z
          C -2.018168  0.740122  0.528813 -0.589001
```

In [48]: *#Printintng the datas greater than 0*
`result_df=df[df["W"]>0]`

In [49]: *#Printintng the result*
`result_df`

```
Out[49]:    W      X      Y      Z
          A  2.706850  0.628133  0.907969  0.503826
          B  0.651118 -0.319318 -0.848077  0.605965
          D  0.188695 -0.758872 -0.933237  0.955057
```

In [50]: *#Printintng teh result column*
`result_df["X"]`

```
Out[50]: A      0.628133
          B     -0.319318
          D     -0.758872
Name: X, dtype: float64
```

In [13]: *#REmoving "W" greater than 0 and getiing "X" indexes*
`df[df["W"]>0][["X","Y"]]`

```
Out[13]:    X      Y
          A  0.628133  0.907969
          B -0.319318 -0.848077
          D -0.758872 -0.933237
          E  1.978757  2.605967
```

```
In [12]: boolser=df["W"]>0
result=df[boolser]
my_columns=["Y","X"]
result[my_columns]
```

Out[12]:

	Y	X
A	0.907969	0.628133
B	-0.848077	-0.319318
D	-0.933237	-0.758872
E	2.605967	1.978757

```
In [10]: #printing the result
result[my_columns]
```

Out[10]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

```
In [19]: #In data frame we use & symbol instead of and symbol
df[(df["W"]>0) & df["Y"]>0]
```

Out[19]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

```
In [20]: #In data frame we use | symbol instead of or symbol
df[(df["W"]>0) | df["Y"]>0]
```

Out[20]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

```
In [23]: #Here we get the index values ABCDE in index columns
df.reset_index()
```

Out[23]:

Index		W	X	Y	Z
0	A	2.706850	0.628133	0.907969	0.503826
1	B	0.651118	-0.319318	-0.848077	0.605965
2	C	-2.018168	0.740122	0.528813	-0.589001
3	D	0.188695	-0.758872	-0.933237	0.955057
4	E	0.190794	1.978757	2.605967	0.683509

```
In [27]: #Creating a new index
new_index="CA NY WY OR CO".split()
```

```
In [25]: #Splitting the words into lists with split
new_in="CA NY UE UEJ UHJ".split()
```

```
In [26]: #Printing the result
new_in
```

```
Out[26]: ['CA', 'NY', 'UE', 'UEJ', 'UHJ']
```

```
In [31]: #Adding new column states into the data frame
df["States"]=new_index
```

```
In [30]: #printing the result
df
```

```
Out[30]:
```

	W	X	Y	Z	States
A	2.706850	0.628133	0.907969	0.503826	CA
B	0.651118	-0.319318	-0.848077	0.605965	NY
C	-2.018168	0.740122	0.528813	-0.589001	WY
D	0.188695	-0.758872	-0.933237	0.955057	OR
E	0.190794	1.978757	2.605967	0.683509	CO

```
In [34]: #TO make the States in the rows as index
df.set_index("States")
```

```
Out[34]:
```

States	W	X	Y	Z
CA	2.706850	0.628133	0.907969	0.503826
NY	0.651118	-0.319318	-0.848077	0.605965
WY	-2.018168	0.740122	0.528813	-0.589001
OR	0.188695	-0.758872	-0.933237	0.955057
CO	0.190794	1.978757	2.605967	0.683509

DataFrame-Multi index and index Key

```
In [37]: #Index Levels
outside=[ "G1", "G1", "G1", "G2", "G2", "G2"]
inside=[1,2,3,1,2,3]
#Zipping means combining two list
hier_index=list(zip(outside,inside))
#Function in pandas for indexing in order
hier_index=pd.MultiIndex.from_tuples(hier_index)
```

```
In [38]: hier_index
```

```
Out[38]: MultiIndex([( 'G1', 1),
                      ( 'G1', 2),
                      ( 'G1', 3),
                      ( 'G2', 1),
                      ( 'G2', 2),
                      ( 'G2', 3)],
                     )
```

```
In [46]: #Creating an DataFrame
df1=pd.DataFrame(randn(6,2),hier_index,[ "A", "B"])
```

```
In [48]: #Here we get two levels of index
df1
```

```
Out[48]:
```

	A	B
G1 1	-0.497104	-0.754070
2	-0.943406	0.484752
3	-0.116773	1.901755
G2 1	0.238127	1.996652
2	-0.993263	0.196800
3	-1.136645	0.000366

```
In [52]: #Locating the values
df1.loc[ "G1"]
```

```
Out[52]:
```

	A	B
1	-0.497104	-0.754070
2	-0.943406	0.484752
3	-0.116773	1.901755

```
In [53]: #Locating the values
df1.loc[ "G1"].loc[1]
```

```
Out[53]: A    -0.497104
B    -0.754070
Name: 1, dtype: float64
```

```
In [54]: #It is used to find the index names in the data frame
df1.index.names
```

```
FrozenList([None, None])
```

Out[54]:

```
In [59]: #Adding names in index of columns
df1.index.names=["Groups","Num"]
```

```
In [58]: #printing the result
df1
```

Out[58]:

	A	B
Groups	Num	
G1	1 -0.497104 -0.754070	
	2 -0.943406 0.484752	
	3 -0.116773 1.901755	
G2	1 0.238127 1.996652	
	2 -0.993263 0.196800	
	3 -1.136645 0.000366	

```
In [62]: df1.loc["G2"].loc[2]["A"]
```

Out[62]: -0.993263499973366

```
In [66]: #Crossaction of rows and columns
#This is used to get the details of whole G1
df1.xs("G1")
```

```
Out[66]:
```

	A	B
Num		
1	-0.497104	-0.754070
2	-0.943406	0.484752
3	-0.116773	1.901755

```
In [67]: #getting the details of both G1 and G2 of number 1
df1.xs(1,level="Num")
```

```
Out[67]:
```

	A	B
Groups		
G1	-0.497104	-0.754070
G2	0.238127	1.996652

MISSING DATAS-->SOLVING WITH PANDAS

-->Normally in pandas if the place or index is not filled then it takes the place as Nan or No

```
In [70]: #Creating an dictionary
#np.nan is used as nan value in the dataframe
d={"A":[1,np.nan,np.nan],"B":[3,np.nan,np.nan],"C":[1,4,2]}
```

```
In [71]: df=pd.DataFrame(d)
```

```
In [72]: df
```

```
Out[72]:
```

	A	B	C
0	1.0	3.0	1
1	NaN	NaN	4
2	NaN	NaN	2

```
In [73]: #Dropna is used to drop all nan values in the dataframe  
#Normally it removes all rows  
df.dropna()
```

```
Out[73]:
```

	A	B	C
0	1.0	3.0	1

```
In [76]: #Axis 1 is taken to drop the columns  
df.dropna(axis=1)
```

```
Out[76]:
```

	C
0	1
1	4
2	2

```
In [81]: #Thresh is used to remove all the nan rows in the dataframe  
df.dropna(thresh=2)
```

```
Out[81]:
```

	A	B	C
0	1.0	3.0	1

```
In [82]: df
```

```
Out[82]:
```

	A	B	C
0	1.0	3.0	1
1	NaN	NaN	4
2	NaN	NaN	2

```
In [83]: #fillna is used fill the nan values into the given value  
#Here we are filling the nan value into 3  
df.fillna(3)
```

```
Out[83]:
```

	A	B	C
0	1.0	3.0	1

	A	B	C
1	3.0	3.0	4
2	3.0	3.0	2

In [85]: *#We can also fill the details with string*
`df.fillna("Filling")`

Out[85]:

	A	B	C
0	1	3	1
1	Filling	Filling	4
2	Filling	Filling	2

In [86]: *#filling the nan value with mean in a*
`df["A"].fillna(value=df["A"].mean())`

Out[86]:

0	1.0
1	1.0
2	1.0

Name: A, dtype: float64

PANDAS-GROUP BY

In [1]:

```
import pandas as pd
# Create dataframe
data = {'Company': ['GOOG', 'GOOG', 'MSFT', 'MSFT', 'FB', 'FB'],
        'Person': ['Sam', 'Charlie', 'Amy', 'Vanessa', 'Carl', 'Sarah'],
        'Sales': [200, 120, 340, 124, 243, 350]}
```

In [2]:

```
df=pd.DataFrame(data)
```

In [3]:

```
df
```

Out[3]:

	Company	Person	Sales
0	GOOG	Sam	200
1	GOOG	Charlie	120
2	MSFT	Amy	340
3	MSFT	Vanessa	124
4	FB	Carl	243
5	FB	Sarah	350

In [6]: *#Grouping up all the companies into single one and storing it in by_company*
`by_company=df.groupby("Company")`

In [8]:

```
by_company
```

Out[8]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001F218E1C400>
```

In [10]: #Finding mean of the company
by_company.mean()

Out[10]: Sales

Company

FB 296.5

GOOG 160.0

MSFT 232.0

In [12]: #Finding maximum values
by_company.max()

Out[12]: Person Sales

Company

FB Sarah 350

GOOG Sam 200

MSFT Vanessa 340

In [13]: #Finding sum of companies
by_company.sum()

Out[13]: Sales

Company

FB 593

GOOG 320

MSFT 464

In [14]: #Finding the sum of the comany and seeing the details of the FB with its location
by_company.sum().loc["FB"]

Out[14]: Sales 593
Name: FB, dtype: int64

NOTE THAT TO FIND ALL THE ABBERRIVATIONS WE USE . OPERATOR TO DO NEXT AND NEXT

In [17]: df.groupby("Company").count()

Out[17]: Person Sales

Company

FB 2 2

GOOG 2 2

```
Person  Sales
```

Company

MSFT	2	2
------	---	---

In [21]: *#This describe function is used to gett all the details of the dataframe*
`df.groupby("Company").describe()`

Out[21]:

Sales								
	count	mean	std	min	25%	50%	75%	max
Company								
FB	2.0	296.5	75.660426	243.0	269.75	296.5	323.25	350.0
GOOG	2.0	160.0	56.568542	120.0	140.00	160.0	180.00	200.0
MSFT	2.0	232.0	152.735065	124.0	178.00	232.0	286.00	340.0

In [22]: *#This describe function is used to gett all the details of the dataframe*
#Transpose is used to make the details in vertical order
`df.groupby("Company").describe().transpose()`

Out[22]:

	Company	FB	GOOG	MSFT
Sales	count	2.000000	2.000000	2.000000
	mean	296.500000	160.000000	232.000000
	std	75.660426	56.568542	152.735065
	min	243.000000	120.000000	124.000000
	25%	269.750000	140.000000	178.000000
	50%	296.500000	160.000000	232.000000
	75%	323.250000	180.000000	286.000000
	max	350.000000	200.000000	340.000000

PANDAS-->MERGING JOINING AND CONCATENATION OF THE DATAFRAMES

In [23]: `df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
 'B': ['B0', 'B1', 'B2', 'B3'],
 'C': ['C0', 'C1', 'C2', 'C3'],
 'D': ['D0', 'D1', 'D2', 'D3']},
 index=[0, 1, 2, 3])`

In [24]: `df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
 'B': ['B4', 'B5', 'B6', 'B7'],
 'C': ['C4', 'C5', 'C6', 'C7'],
 'D': ['D4', 'D5', 'D6', 'D7']},
 index=[4, 5, 6, 7])`

In [25]: `df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
 'B': ['B8', 'B9', 'B10', 'B11'],
 'C': ['C8', 'C9', 'C10', 'C11'],
 'D': ['D8', 'D9', 'D10', 'D11']},
 index=[8, 9, 10, 11])`

```
'B': ['B8', 'B9', 'B10', 'B11'],
'C': ['C8', 'C9', 'C10', 'C11'],
'D': ['D8', 'D9', 'D10', 'D11']},
index=[8, 9, 10, 11])
```

In [27]: df1

Out[27]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

In [28]: df2

Out[28]:

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

In [29]: df3

Out[29]:

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

CONCATENATION

Concatenation basically glues together DataFrames. Keep in mind that dimensions should match along the axis you are concatenating on. You can use pd.concat and pass in a list of DataFrames to concatenate together

In [31]:

```
#Concatenation is combining more than one dataframes
#In default axis=0 so it concatenates with rows
pd.concat([df1,df2,df3])
```

Out[31]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4

	A	B	C	D
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

```
In [33]: #When the axis is given as 1 it combined with columns
#We get NaN for missing values
pd.concat([df1,df2,df3],axis=1)
```

Out[33]:

	A	B	C	D	A	B	C	D	A	B	C	D
0	A0	B0	C0	D0	NaN							
1	A1	B1	C1	D1	NaN							
2	A2	B2	C2	D2	NaN							
3	A3	B3	C3	D3	NaN							
4	NaN	NaN	NaN	NaN	A4	B4	C4	D4	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	A5	B5	C5	D5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	A6	B6	C6	D6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	A7	B7	C7	D7	NaN	NaN	NaN	NaN
8	NaN	A8	B8	C8	D8							
9	NaN	A9	B9	C9	D9							
10	NaN	A10	B10	C10	D10							
11	NaN	A11	B11	C11	D11							

```
In [34]: left = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                           'A': ['A0', 'A1', 'A2', 'A3'],
                           'B': ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                      'C': ['C0', 'C1', 'C2', 'C3'],
                      'D': ['D0', 'D1', 'D2', 'D3']})
```

PANDAS-->MERGING

The **merge** function allows you to merge DataFrames together using a similar logic as merging SQL Tables together. For example:

```
In [35]: #Displaying left Dataframe
left
```

Out[35]:

	key	A	B
0	K0	A0	B0
1	K1	A1	B1
2	K2	A2	B2
3	K3	A3	B3

In [37]:

```
#Displaying right dataframe
right
```

Out[37]:

	key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K2	C2	D2
3	K3	C3	D3

In [40]:

```
#Here we use on as key column which acts common on both dataframe in left and right
pd.merge(left,right,how="inner",on="key")
```

Out[40]:

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K1	A1	B1	C1	D1
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3

In [41]:

```
#Creating a Left Dataframe
left = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
                     'key2': ['K0', 'K1', 'K0', 'K1'],
                     'A': ['A0', 'A1', 'A2', 'A3'],
                     'B': ['B0', 'B1', 'B2', 'B3']})
```

```
#Creating a Dataframe called right
```

```
right = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
                      'key2': ['K0', 'K0', 'K0', 'K0'],
                      'C': ['C0', 'C1', 'C2', 'C3'],
                      'D': ['D0', 'D1', 'D2', 'D3']})
```

In [44]:

```
#Merging both left and right dataframes which has common key1 and key2
pd.merge(left,right,how="outer",on=["key1","key2"])
```

Out[44]:

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	C1	D1
3	K1	K0	A2	B2	C2	D2

	key1	key2	A	B	C	D
4	K2	K1	A3	B3	NaN	NaN
5	K2	K0	NaN	NaN	C3	D3

```
In [45]: pd.merge(left,right,how="right",on=["key1","key2"])
```

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2
3	K2	K0	NaN	NaN	C3	D3

PANDAS-->JOINING

Joining is a convenient method for combining the columns of two potentially differently-indexed DataFrames into a single result DataFrame.

```
In [46]: left = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                           'B': ['B0', 'B1', 'B2']},
                           index=['K0', 'K1', 'K2'])

right = pd.DataFrame({'C': ['C0', 'C2', 'C3'],
                      'D': ['D0', 'D2', 'D3']},
                      index=['K0', 'K2', 'K3'])
```

```
In [47]: left.join(right)
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2

```
In [48]: left.join(right,how="outer")
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2
K3	NaN	NaN	C3	D3

PANDAS-->OPERATIONS

```
In [1]: import pandas as pd
df = pd.DataFrame({'col1':[1,2,3,4],'col2':[444,555,666,444],'col3':['abc','def','ghi',
```

```
df.head()
```

```
Out[1]:   col1  col2  col3
0      1    444    abc
1      2    555    def
2      3    666    ghi
3      4    444    xyz
```

```
In [5]: #Finding unique values in dataframe
df["col2"].unique()
```

```
Out[5]: array([444, 555, 666], dtype=int64)
```

```
In [6]: ## Finding length of the unique values in dataframe
df["col2"].nunique()
```

```
Out[6]: 3
```

```
In [8]: #counting the values of each row index datas
df["col2"].value_counts()
```

```
Out[8]: 444    2
555    1
666    1
Name: col2, dtype: int64
```

```
In [12]: #condition selection
df[df["col1"]>2]
```

```
Out[12]:   col1  col2  col3
2      3    666    ghi
3      4    444    xyz
```

```
In [16]: #Creating an userdefined funciton
def times2(x):
    return x*2
```

```
In [17]: #Dataframe has a ability to execute the data in userdefined function also
#For this we use apply function
df["col1"].apply(times2)
```

```
Out[17]: 0    2
1    4
2    6
3    8
Name: col1, dtype: int64
```

```
In [21]: df["col3"].apply(len)
```

```
Out[21]: 0    3
1    3
2    3
```

```
3      3
Name: col3, dtype: int64
```

```
In [22]: #using Lambda to multiply the values into 2
df["col2"].apply(lambda x :x**2)
```

```
Out[22]: 0    888
1   1110
2   1332
3    888
Name: col2, dtype: int64
```

```
In [23]: #This is used to return the columns in data frame
df.columns
```

```
Out[23]: Index(['col1', 'col2', 'col3'], dtype='object')
```

```
In [25]: #Viewing the range index
df.index
```

```
Out[25]: RangeIndex(start=0, stop=4, step=1)
```

```
In [27]: #sorting the datas with given rows or columns
df.sort_values("col2")
```

	col1	col2	col3
0	1	444	abc
3	4	444	xyz
1	2	555	def
2	3	666	ghi

```
In [29]: #Getting a boolean result whether any data is not given
df.isnull()
```

	col1	col2	col3
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False

Pandas-->Pivot-Table

```
In [30]: data = {'A':['foo','foo','foo','bar','bar','bar'],
             'B':['one','one','two','two','one','one'],
             'C':['x','y','x','y','x','y'],
             'D':[1,3,2,5,4,1]}

df = pd.DataFrame(data)
```

```
In [31]: df
```

Out[31]:

	A	B	C	D
0	foo	one	x	1
1	foo	one	y	3
2	foo	two	x	2
3	bar	two	y	5
4	bar	one	x	4
5	bar	one	y	1

In [33]:

```
#Creating an pivot tables
#Create a spreadsheet-style pivot table as a DataFrame.
#Here we are taking D values with index of A and B in column C
df.pivot_table(values="D",index=["A","B"],columns="C")
```

Out[33]:

	C	x	y
A	B		
bar	one	4.0	1.0
	two	NaN	5.0
foo	one	1.0	3.0
	two	2.0	NaN

Pandas-->Data Input and Data Output

In [35]:

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

In [38]:

```
#Reading csv files
df = pd.read_csv('example')
df
```

Out[38]:

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

In [39]:

```
#Writing into csv files
df.to_csv('example',index=False)
```

Excel

Pandas can read and write excel files, keep in mind, this only imports data. Not formulas or images, having images or macros may cause this read_excel method to crash.

```
In [43]: #reading excel files
pd.read_excel('Excel_Sample.xlsx')
```

```
Out[43]:   Unnamed: 0   a   b   c   d
0           0   0   1   2   3
1           1   4   5   6   7
2           2   8   9  10  11
3           3  12  13  14  15
```

```
In [44]: #Writing into excel file
df.to_excel('Excel_Sample.xlsx',sheet_name='Sheet1')
```

MATPLOTLIB

Matplotlib is most popular plotting library for python. It is designed similar as matLab graphical plotting. Website for matplotlib is: matplotlib.org

```
In [31]: #Importing matplotlib pyplot as plt
import matplotlib.pyplot as plt
```

```
In [32]: #This is used to run in jupyter notebook
%matplotlib inline
```

```
In [33]: #Importing module
import numpy as np
```

```
In [93]: #Creating the array
x=np.linspace(0,5,11)
y=x ** 2
```

```
In [35]: x
```

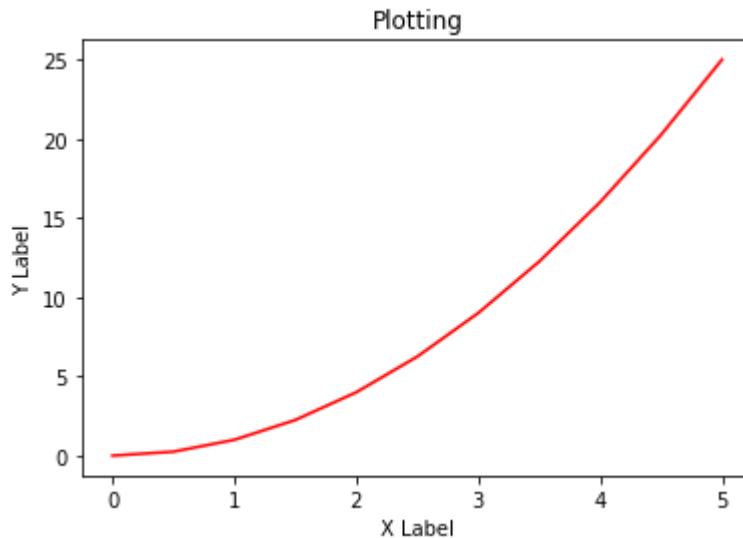
```
Out[35]: array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])
```

```
In [36]: y
```

```
Out[36]: array([ 0. ,  0.25,  1. ,  2.25,  4. ,  6.25,  9. , 12.25, 16. ,
 20.25, 25. ])
```

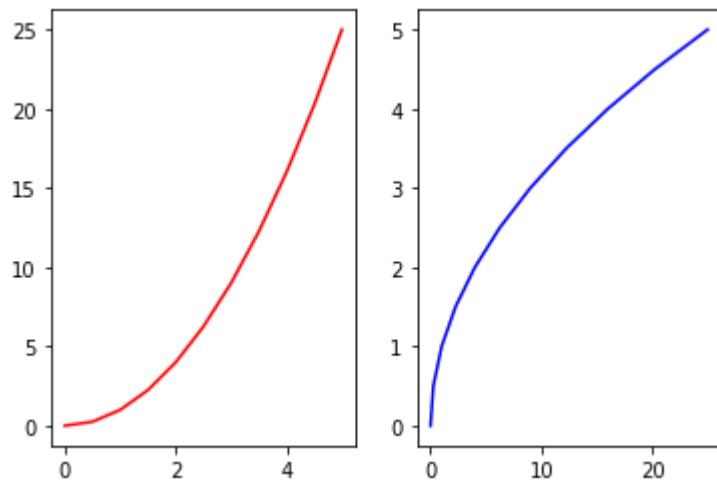
```
In [37]: #functional plotting
#Displaying the plot with the x and y data
plt.plot(x,y,"red")
#Printing the x axis label
plt.xlabel("X Label")
#Printing the y axis label
plt.ylabel("Y Label")
#Printing the title of the graph
plt.title("Plotting")
```

```
Out[37]: Text(0.5, 1.0, 'Plotting')
```



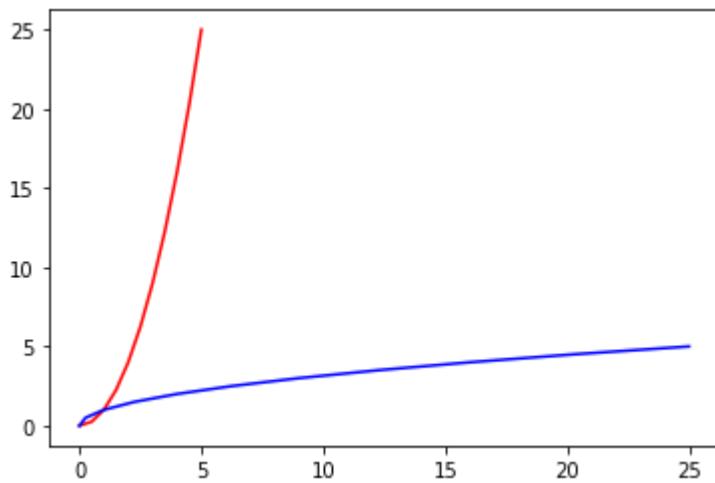
```
In [38]: #Creating two subplots next to next  
#Here subplot is used to place it in the position  
plt.subplot(1,2,1)  
plt.plot(x,y,"r")  
plt.subplot(1,2,2)  
plt.plot(y,x,"blue")
```

```
Out[38]: [
```



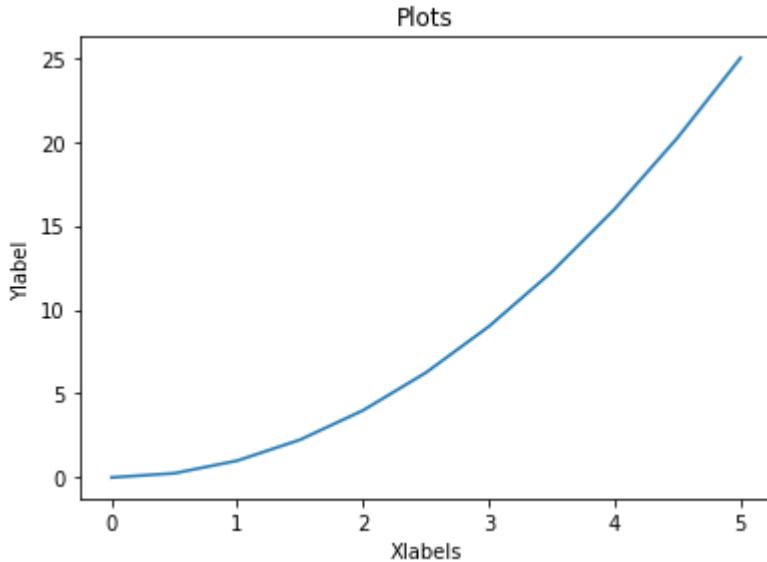
```
In [39]: #Two plots in same graph sheet  
plt.plot(x,y,"r")  
plt.plot(y,x,"blue")
```

```
Out[39]: [
```



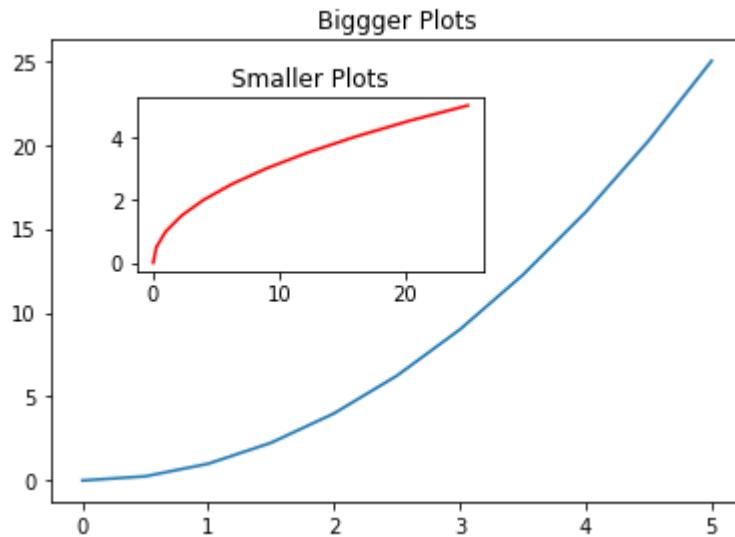
```
In [40]: # Object Oriented Method
#Creating an figure plot i,e: crating an blank graph
figure=plt.figure()
#Adding values in the x-axis and y-axis in the graph
axes=figure.add_axes([0.1,0.1,0.8,0.8])
#Adding plotted graph in the figure graph sheet
axes.plot(x,y)
#Labels in the graph
axes.set_xlabel("Xlabel")
axes.set_ylabel("Ylabel")
axes.set_title("Plots")
```

Out[40]: Text(0.5, 1.0, 'Plots')

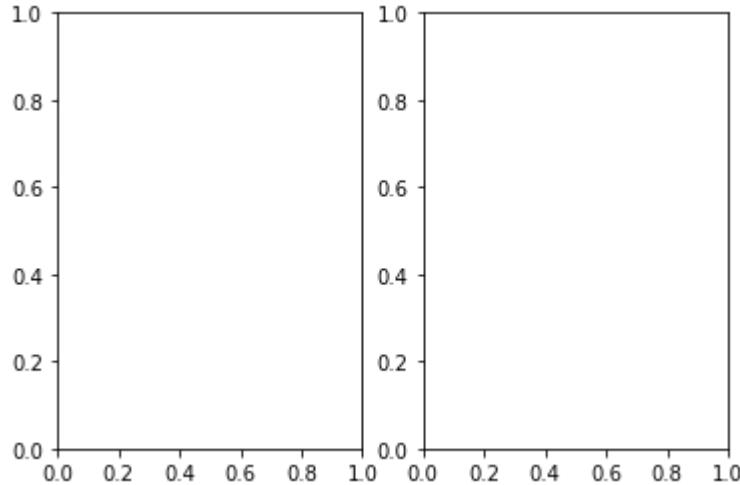


```
In [41]: fig=plt.figure()
#Crating two figures
axes1=fig.add_axes([0.1,0.1,0.8,0.8])
axes2=fig.add_axes([0.2,0.5,0.4,0.3])
#Adding the plot in the figure graph
axes1.plot(x,y)
axes2.plot(y,x,"red")
axes.set_title("Bigger in smaller plots")
axes1.set_title("Bigger Plots")
axes2.set_title("Smaller Plots")
```

Out[41]: Text(0.5, 1.0, 'Smaller Plots')



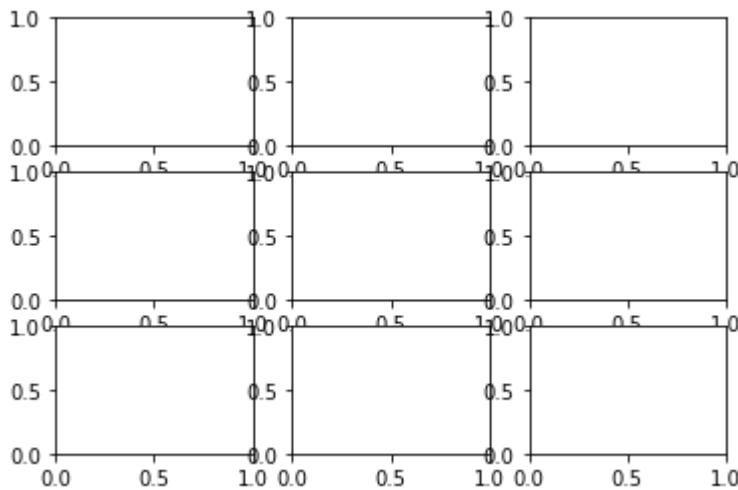
```
In [42]: #Creating two subplots graph  
fig,axes=plt.subplots(nrows=1,ncols=2)
```



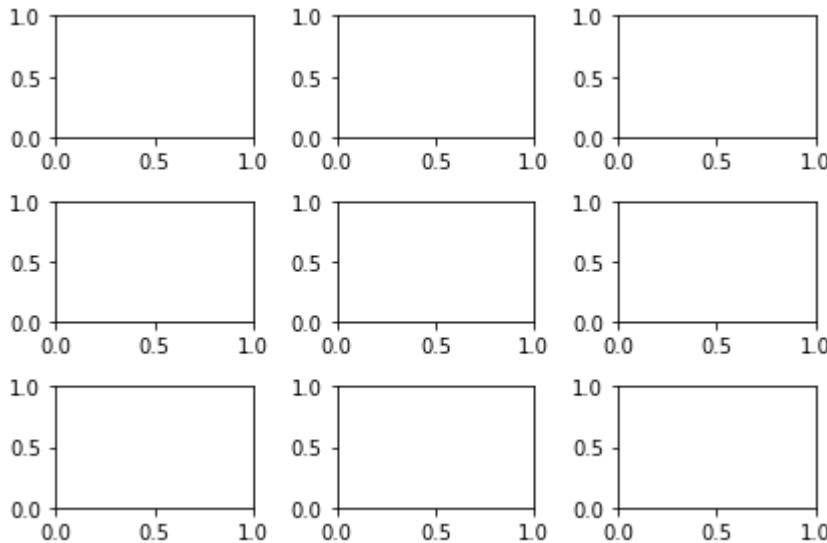
In [43]: axes

Out[43]: array([<AxesSubplot:>, <AxesSubplot:>], dtype=object)

```
In [44]: #Creating two subplots graph  
#It is like matrix  
fig,axes=plt.subplots(nrows=3,ncols=3)
```



```
In [45]: #Creating two subplots graph
#It is like matrix
fig,axes=plt.subplots(nrows=3,ncols=3)
#This tight_layout is used to space between each graph
plt.tight_layout()
```



```
In [46]: axes
```

```
Out[46]: array([[],[],[],[],[],[],[],[],[]], dtype=object)
```

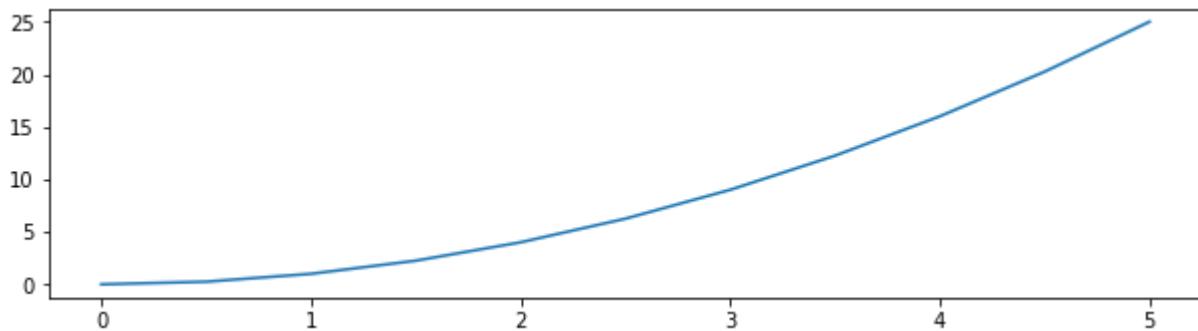
MATPLOTLIB-->FIGURE SIZE,ASPECT RATIO AND DPI

Figure size

```
In [59]: #Increasing the figure size i.e:increasing the graph size
fig=plt.figure(figsize=(8,2))
axes=fig.add_axes([0,0,1,1])
axes.plot(x,y)
```

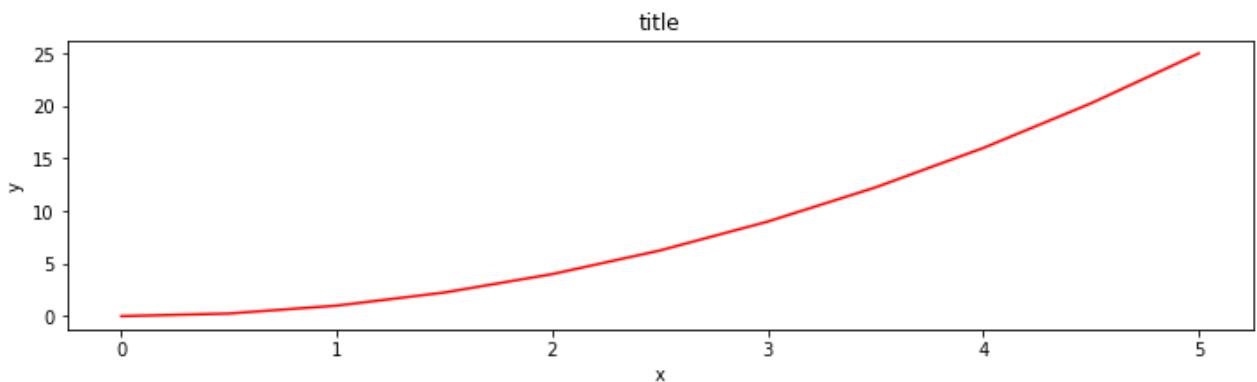
```
Out[59]: [

```



In [60]:

```
#in sub plots we need to put two variables to get the output
#Fig size is used to change the size of the graph
fig, axes = plt.subplots(figsize=(12,3))
#Plotting x and y graph
axes.plot(x, y, 'r')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');
```

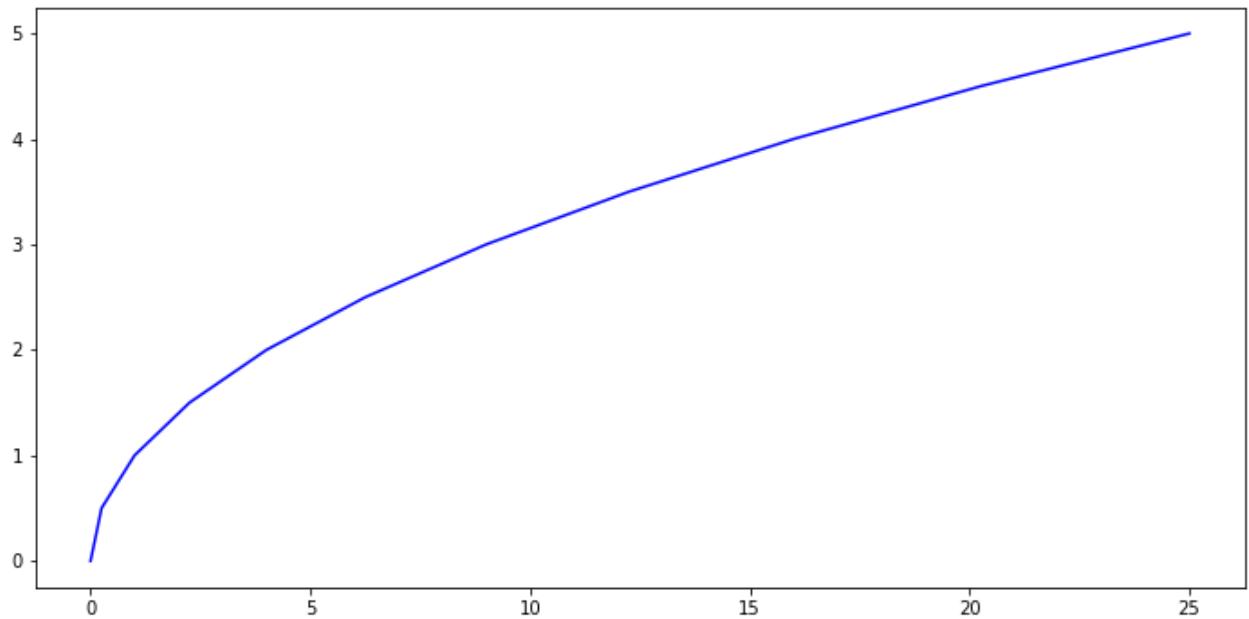
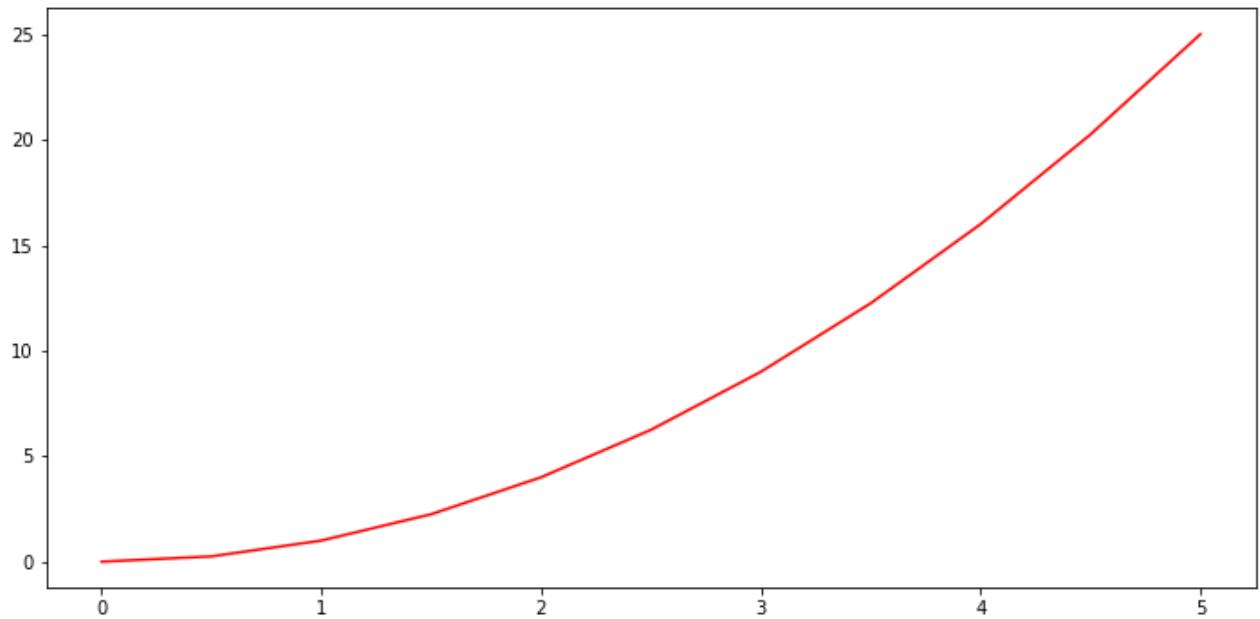


In [66]:

```
#Creating an subplots
figure,axes=plt.subplots(nrows=2,ncols=1,figsize=(12,13))
#Putting fist plot axes[0] as x and y graph
axes[0].plot(x,y,"red")
#Putting second plot as y and x graph
axes[1].plot(y,x,"blue")
```

Out[66]:

```
[<matplotlib.lines.Line2D at 0x1bc929fa520>]
```



COPYING THE FIGURE INTO THE FILE

```
In [70]: #Copying the high quality plots in pdf or jpeg or png formats  
#To save the figure in the file as jpg  
#Dpi is used for the resolution  
figure.savefig("My_image.jpg",dpi=500)
```

```
In [68]: #To save the figure in the file as png  
figure.savefig("My_image.png")
```

```
In [69]: #To save the figure in the file as pdf  
figure.savefig("My_image.pdf")
```

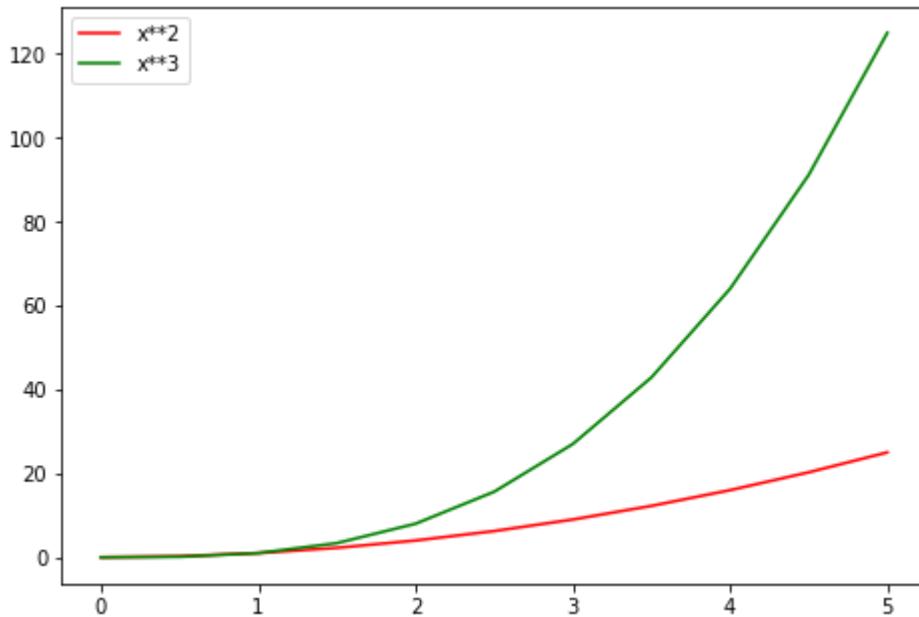
Legends, labels and titles

Legends

In [78]: #Method 1 of adding Label

```
figure=plt.figure()
ax=figure.add_axes([0,0,1,1])
ax.plot(x,x**2,"red")
ax.plot(x,x**3,"green")
labels=["x**2","x**3"]
ax.legend(labels)
```

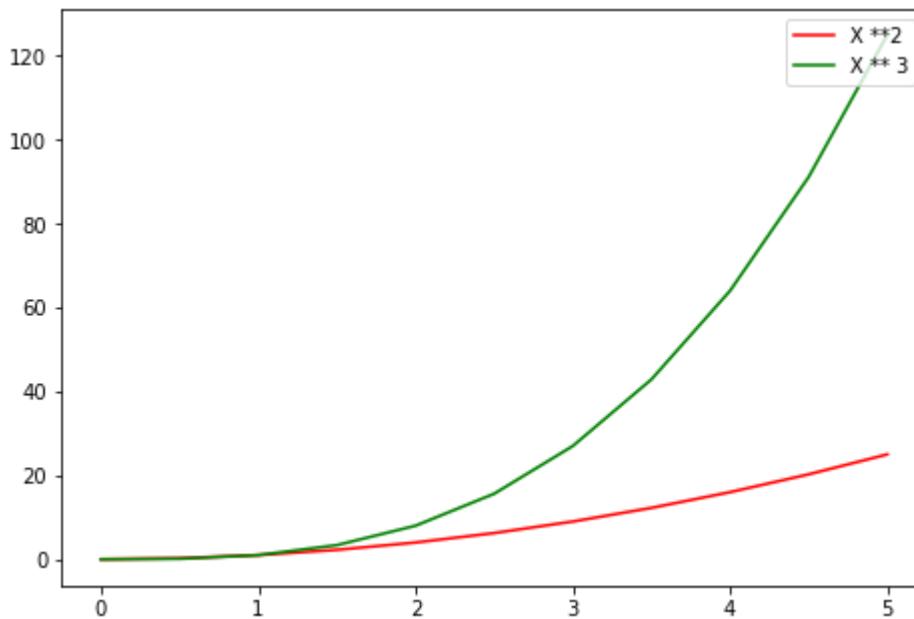
Out[78]: <matplotlib.legend.Legend at 0x1bc936e79d0>



In [83]:

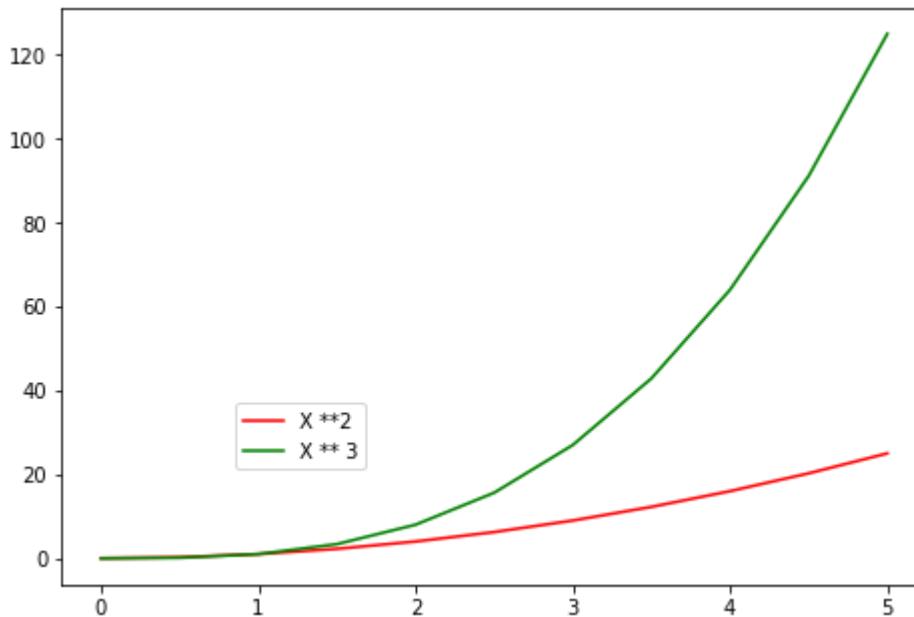
```
#Method 2 of adding Label
#For more information about Legend head over to matplotlib.org
figure=plt.figure()
ax=figure.add_axes([0,0,1,1])
ax.plot(x,x**2,"red",label="X **2")
ax.plot(x,x**3,"green",label="X ** 3")
labels=["x**2","x**3"]
#Adding location in the Legend parameters
ax.legend(loc=1)
```

Out[83]: <matplotlib.legend.Legend at 0x1bc8db3bcd0>



```
In [91]: #For more information about Legend head over to matplotlib.org
figure=plt.figure()
ax=figure.add_axes([0,0,1,1])
ax.plot(x,x**2,"red",label="X **2")
ax.plot(x,x**3,"green",label="X ** 3")
labels=["x**2","x**3"]
#Adding location in the Legend parameters
#Adding the location in x axis and y axis like in the graph
ax.legend(loc=[0.2,0.2])
```

Out[91]: <matplotlib.legend.Legend at 0x1bc931c4a60>

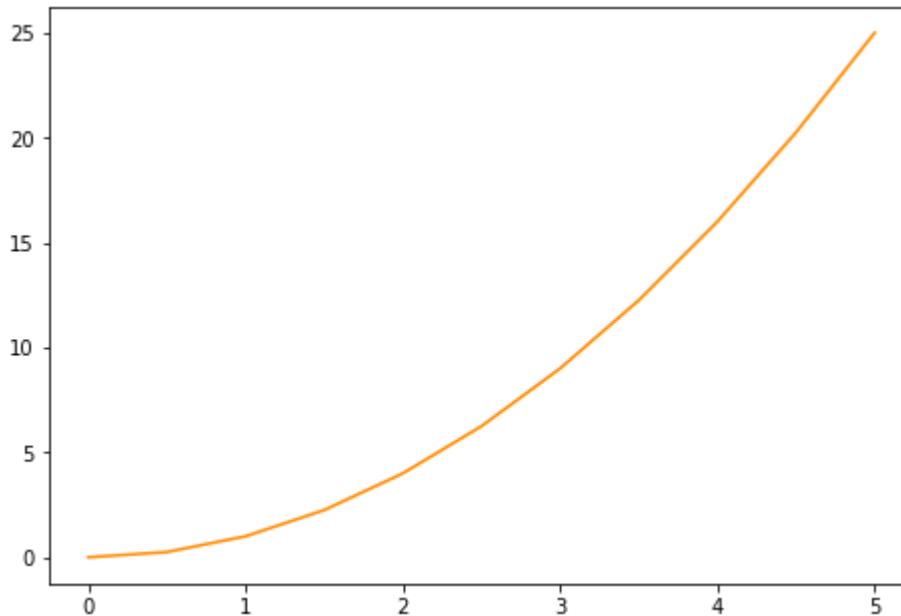


PLOT APPEARANCE

```
In [102...]: #Creating an graph sheet for plot
figure=plt.figure()
#Adding the axes of the plot that is size
axes=figure.add_axes([0,0,1,1])
```

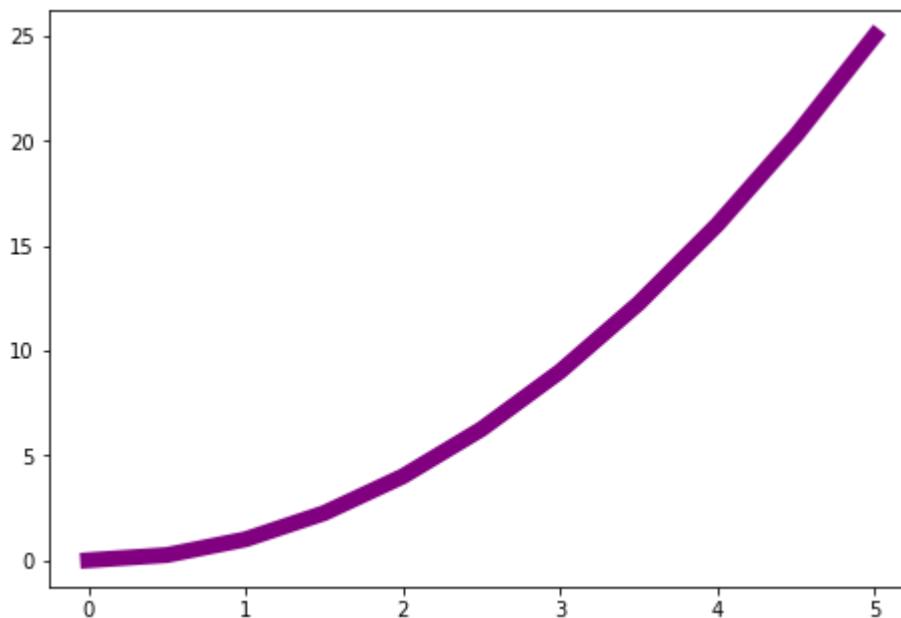
```
#Adding graph in the plot with color
axes.plot(x,y,color="#FF8C00") #RGB Heccodes available in internet
```

Out[102...]: <matplotlib.lines.Line2D at 0x1bc9f290610>



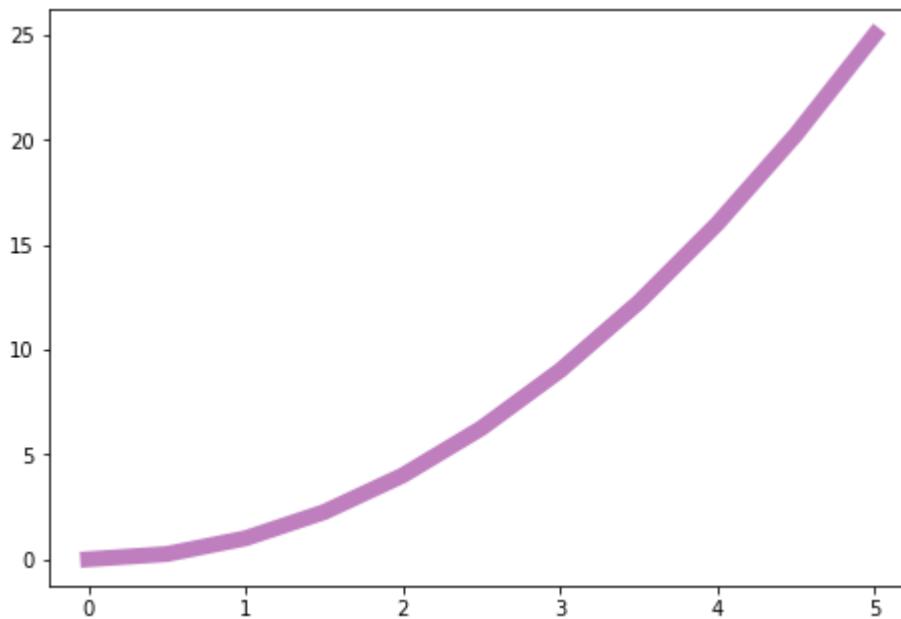
```
In [104...]: figure=plt.figure()
axes=figure.add_axes([0,0,1,1])
#Here the thickness of the line can be changed with linewidth
axes.plot(x,y,color="purple",linewidth=8)
```

Out[104...]: <matplotlib.lines.Line2D at 0x1bc9f33d160>



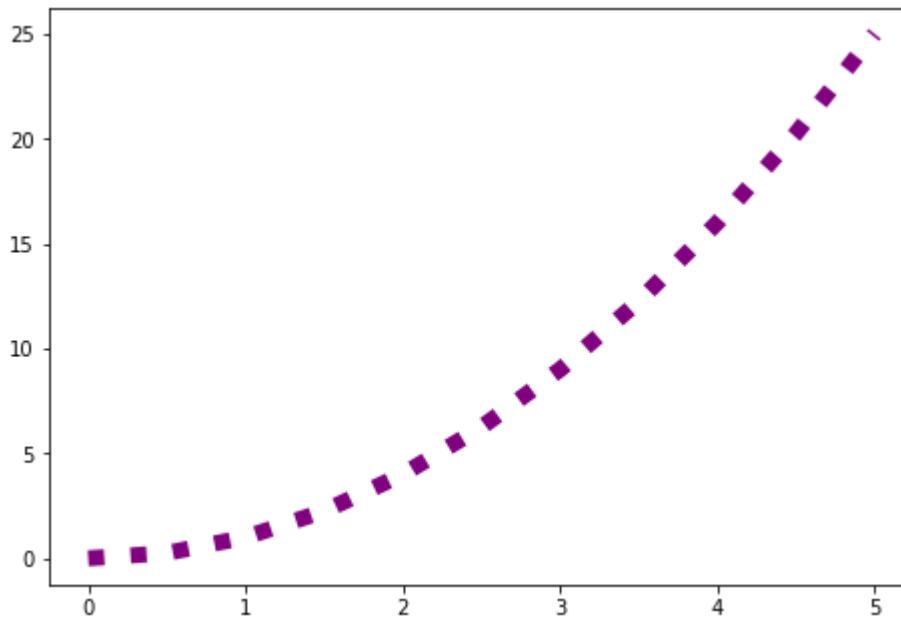
```
In [105...]: figure=plt.figure()
axes=figure.add_axes([0,0,1,1])
#Here the thickness of the line can be changed with linewidth
#To get the transparency of the Line we use alpha
axes.plot(x,y,color="purple",linewidth=8,alpha=0.5)
```

Out[105...]: <matplotlib.lines.Line2D at 0x1bc9f393040>



```
In [112]: figure=plt.figure()
axes=figure.add_axes([0,0,1,1])
#Here the thickness of the line can be changed with linewidth
#Line style is used to change it into different styles of the line
axes.plot(x,y,color="purple",linewidth=8,linestyle=":")
```

```
Out[112]: <matplotlib.lines.Line2D at 0x1bc92916bb0>
```

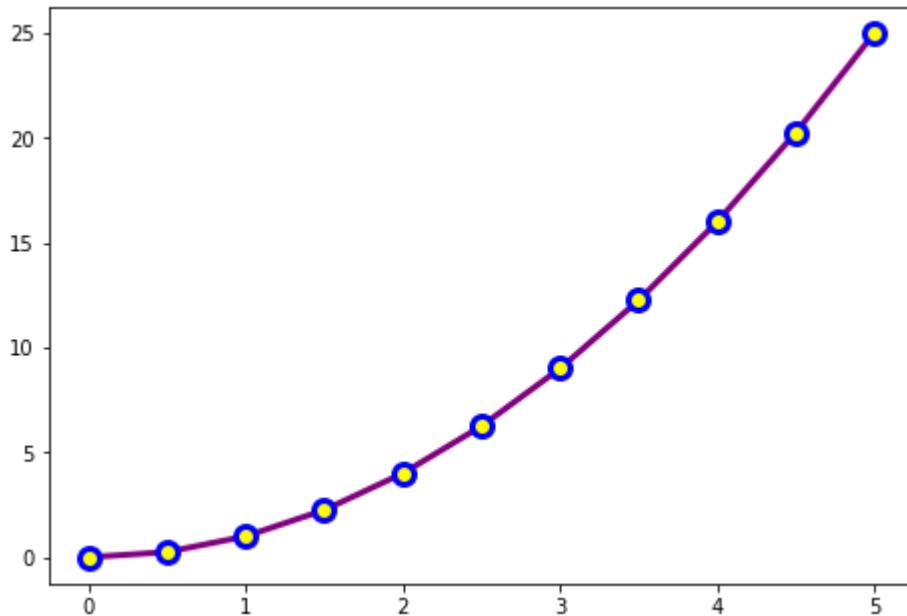


MARKERS IN THE GRAPH PLOT i.e: CUSTAMIZATION IN THE GRAPH

```
In [127]: figure=plt.figure()
axes=figure.add_axes([0,0,1,1])
#Here the thickness of the line can be changed with linewidth
#Marker is used to plot te values in the graph as dots(.)
#Marker size is to increase the size of the plots in the graph
#Markerface color is the color for the circle plots in the graph
#Marker edge width is the size of the circles edge
#Marker edge colour is the color of the circle edges in the graph line plots
```

```
axes.plot(x,y,color="purple",linewidth=3,linestyle="--",marker="o",markersize=10,
          markerfacecolor="yellow",markeredgecolor="blue")
```

Out[127...]: <matplotlib.lines.Line2D at 0x1bca0951fd0>



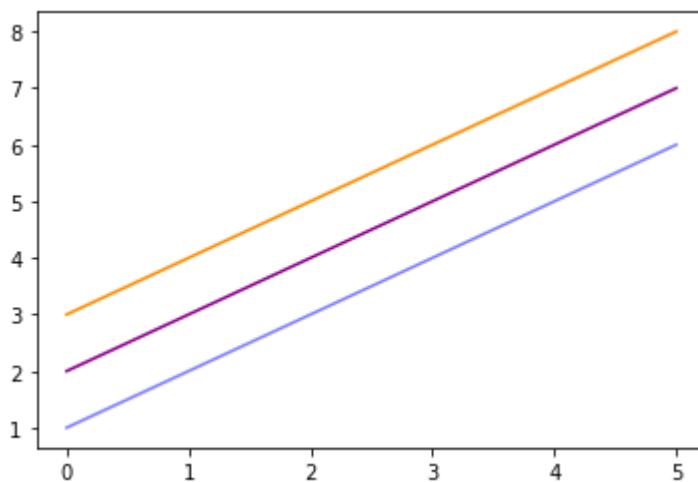
Colors with the color= parameter

We can also define colors by their names or RGB hex codes and optionally provide an alpha value using the `color` and `alpha` keyword arguments. Alpha indicates opacity.

```
In [129...]: fig, ax = plt.subplots()

ax.plot(x, x+1, color="blue", alpha=0.5) # half-transparent
ax.plot(x, x+2, color="#8B008B")        # RGB hex code
ax.plot(x, x+3, color="#FF8C00")        # RGB hex code
```

Out[129...]: <matplotlib.lines.Line2D at 0x1bca09b75e0>



Line and marker styles

To change the line width, we can use the `linewidth` or `lw` keyword argument. The line style can be selected using the `linestyle` or `ls` keyword arguments:

In [131...]

```
fig, ax = plt.subplots(figsize=(12,6))

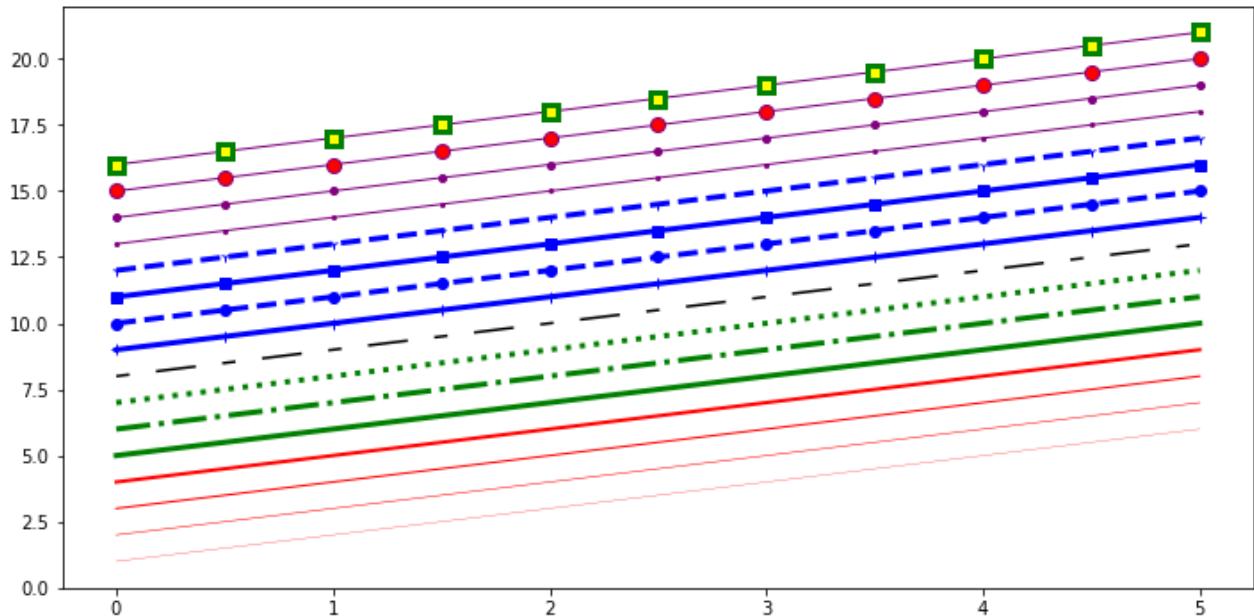
ax.plot(x, x+1, color="red", linewidth=0.25)
ax.plot(x, x+2, color="red", linewidth=0.50)
ax.plot(x, x+3, color="red", linewidth=1.00)
ax.plot(x, x+4, color="red", linewidth=2.00)

# possible linestyle options '--', '-.', '-.', ':', 'steps'
ax.plot(x, x+5, color="green", lw=3, linestyle='--')
ax.plot(x, x+6, color="green", lw=3, ls='-.')
ax.plot(x, x+7, color="green", lw=3, ls=':')

# custom dash
line, = ax.plot(x, x+8, color="black", lw=1.50)
line.set_dashes([5, 10, 15, 10]) # format: line length, space length, ...

# possible marker symbols: marker = '+', 'o', '*', 's', ' ', '.', '1', '2', '3', '4', .
ax.plot(x, x+ 9, color="blue", lw=3, ls='--', marker='+')
ax.plot(x, x+10, color="blue", lw=3, ls='--', marker='o')
ax.plot(x, x+11, color="blue", lw=3, ls='--', marker='s')
ax.plot(x, x+12, color="blue", lw=3, ls='--', marker='1')

# marker size and color
ax.plot(x, x+13, color="purple", lw=1, ls='--', marker='o', markersize=2)
ax.plot(x, x+14, color="purple", lw=1, ls='--', marker='o', markersize=4)
ax.plot(x, x+15, color="purple", lw=1, ls='--', marker='o', markersize=8, markerfacecolor="yellow", markeredgewidth=3, markeredgecolor="green");
```



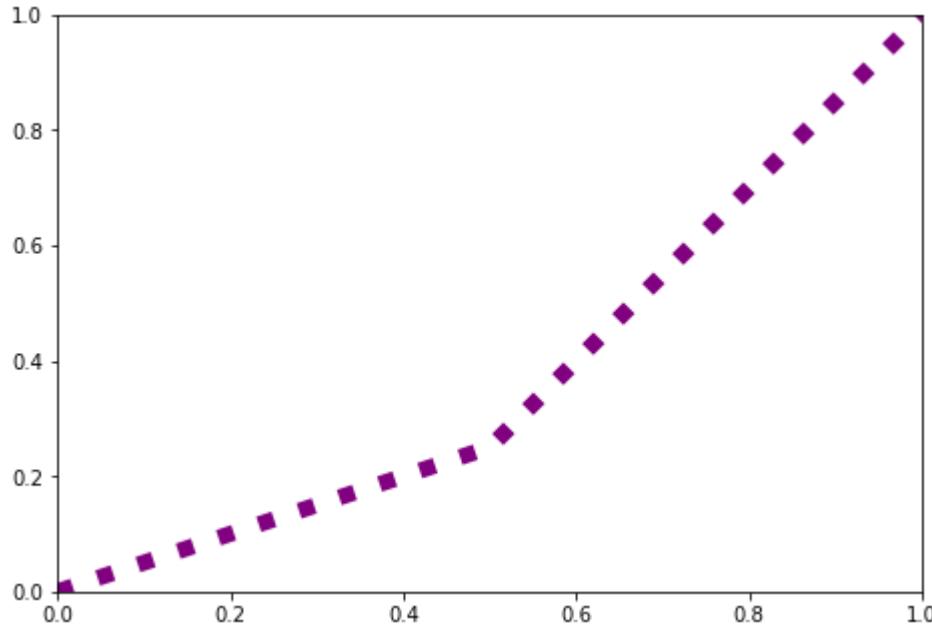
Control over axis appearance

In [135...]

```
#Here i want to show only 0 to 1 in x-axis and 0 to 5 scale in y axis
#i,e like zoom it out
figure=plt.figure()
axes=figure.add_axes([0,0,1,1])
#Here the thickness of the line can be changed with linewidth
#Line style is used to change it into different styles of the line
axes.plot(x,y,color="purple",linewidth=8,linestyle=":")
```

```
axes.set_xlim([0,1])
axes.set_ylim([0,1])
```

Out[135... (0.0, 1.0)



Plot range

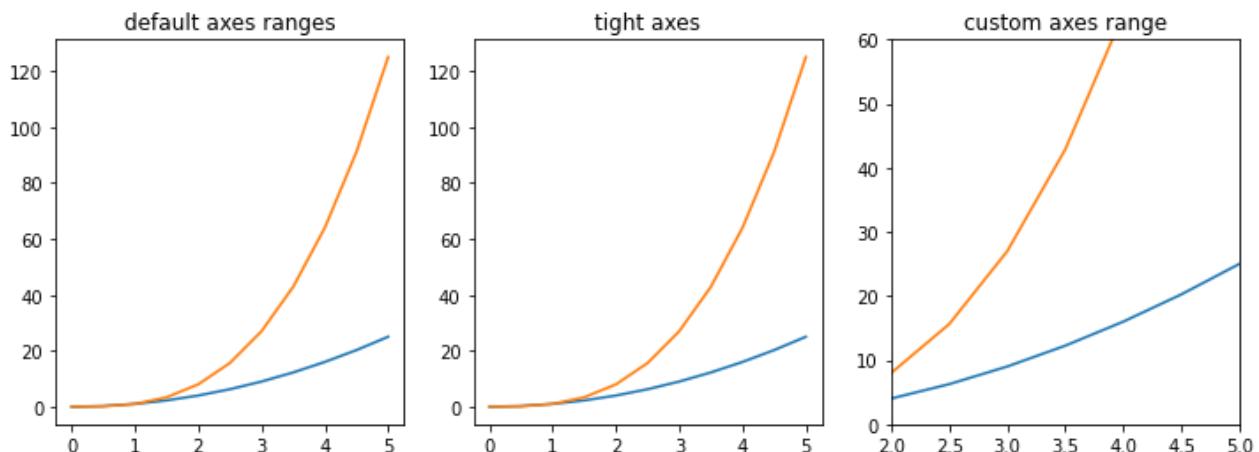
We can configure the ranges of the axes using the `set_ylim` and `set_xlim` methods in the axis object, or `axis('tight')` for automatically getting "tightly fitted" axes ranges:

```
In [137... fig, axes = plt.subplots(1, 3, figsize=(12, 4))

axes[0].plot(x, x**2, x, x**3)
axes[0].set_title("default axes ranges")

axes[1].plot(x, x**2, x, x**3)
axes[1].axis('tight')
axes[1].set_title("tight axes")

axes[2].plot(x, x**2, x, x**3)
axes[2].set_xlim([2, 5])
axes[2].set_ylim([0, 60])
axes[2].set_title("custom axes range");
```

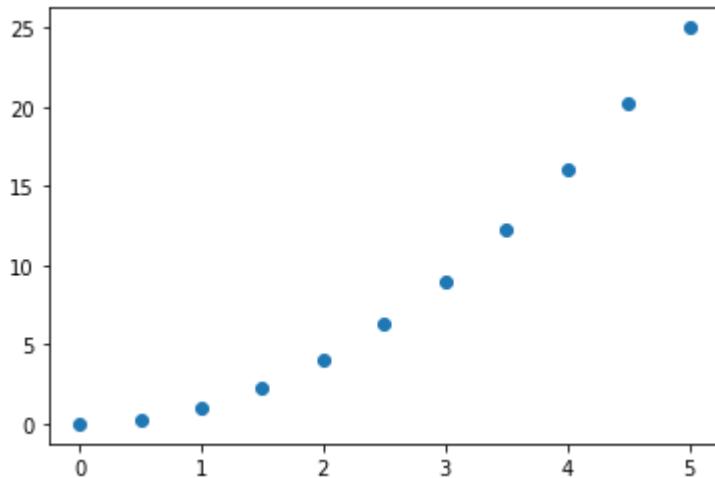


Special Plot Types

There are many specialized plots we can create, such as barplots, histograms, scatter plots, and much more. Most of these type of plots we will actually create using seaborn, a statistical plotting library for Python. But here are a few examples of these type of plots:

```
In [138... plt.scatter(x,y)
```

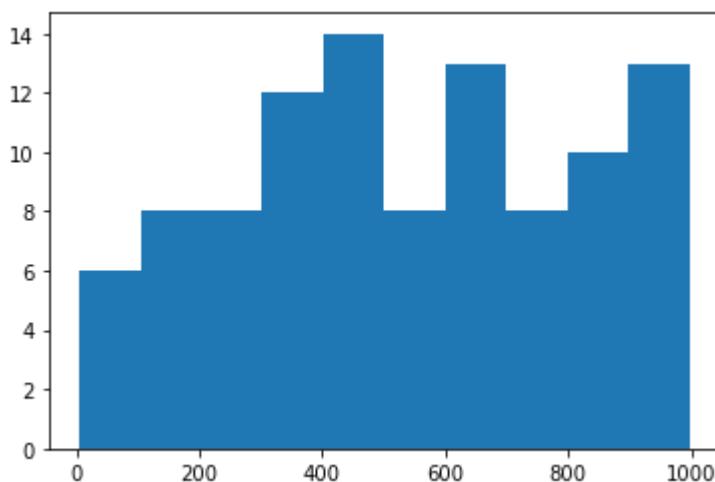
```
Out[138... <matplotlib.collections.PathCollection at 0x1bca0b33730>
```



```
In [139... #Creating an histogram
```

```
from random import sample
data = sample(range(1, 1000), 100)
plt.hist(data)
```

```
Out[139... (array([ 6.,  8.,  8., 12., 14.,  8., 13.,  8., 10., 13.]),
 array([ 5. , 104.1, 203.2, 302.3, 401.4, 500.5, 599.6, 698.7, 797.8,
 896.9, 996. ]),
 <BarContainer object of 10 artists>)
```

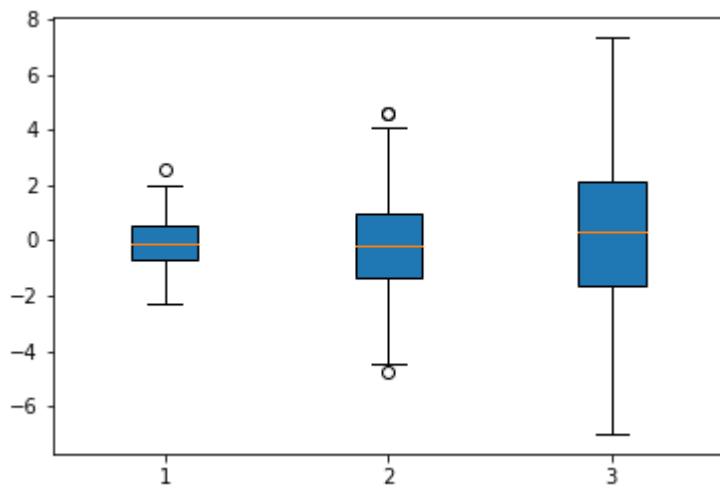


```
In [141... #Creating an boxplot plots
```

```
data = [np.random.normal(0, std, 100) for std in range(1, 4)]
```

```
# rectangular box plot
```

```
plt.boxplot(data,vert=True,patch_artist=True);
```



SEABORN

-->Seaborn is a statistical plotting library -->It has beautiful default style -->It also is designed to work very well with pandas dataframe object-->Official website for seaborn is: seaborn github python -->or head over to :<http://seaborn.pydata.org/>

Seaborn-->Distribution Plots

```
In [1]: #Importing seaborn module package
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
In [2]: tips=sns.load_dataset("tips")
```

```
In [146...]: tips
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

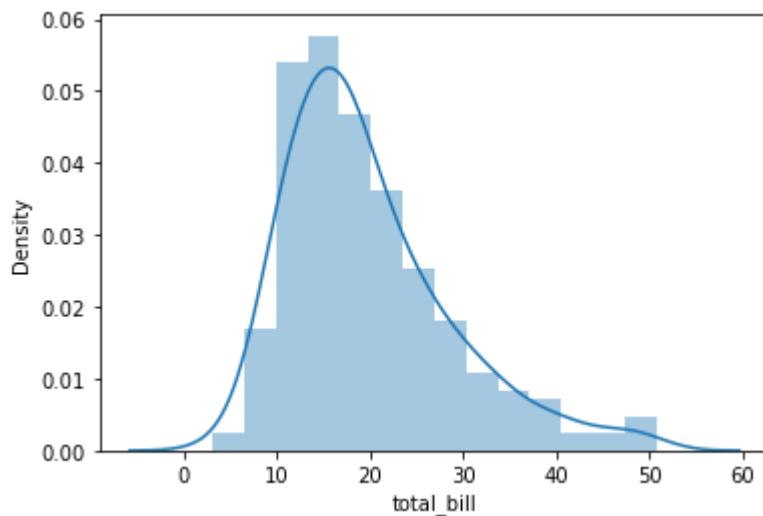
In [147...]: `tips.head()`

Out[147...]:

	total_bill	tip	sex	smoker	day	time	size	
0	16.99	1.01	Female		No	Sun	Dinner	2
1	10.34	1.66	Male		No	Sun	Dinner	3
2	21.01	3.50	Male		No	Sun	Dinner	3
3	23.68	3.31	Male		No	Sun	Dinner	2
4	24.59	3.61	Female		No	Sun	Dinner	4

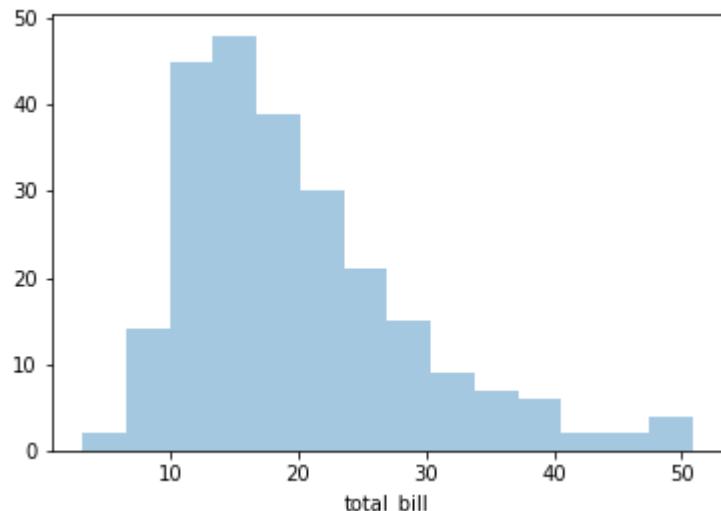
In [149...]: `# Dist Plot with kde plot
sns.distplot(tips["total_bill"])`

Out[149...]: <AxesSubplot:xlabel='total_bill', ylabel='Density'>



In [151...]: `# Dist Plot without kde plot
sns.distplot(tips["total_bill"], kde=False)`

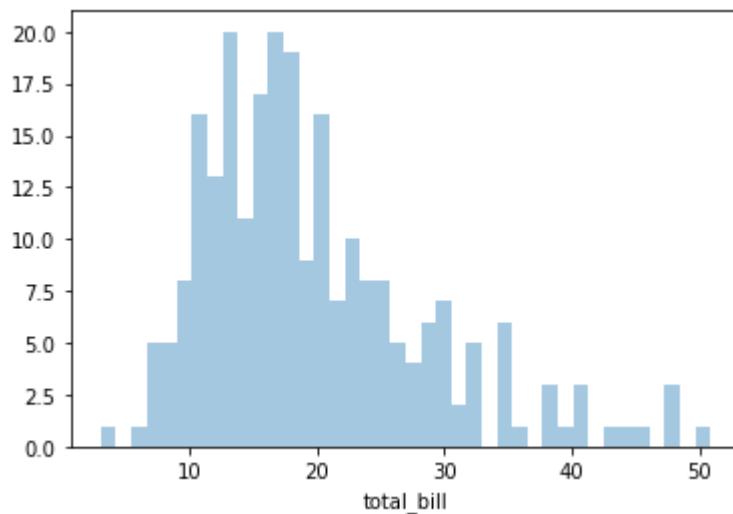
Out[151...]: <AxesSubplot:xlabel='total_bill'>



In [157...]: `# Dist Plot without kde plot
#Bins means no of persons in the graph`

```
#So it displays 40 member of data's in the plot
sns.distplot(tips["total_bill"],kde=False,bins=40)
```

Out[157... <AxesSubplot:xlabel='total_bill'>

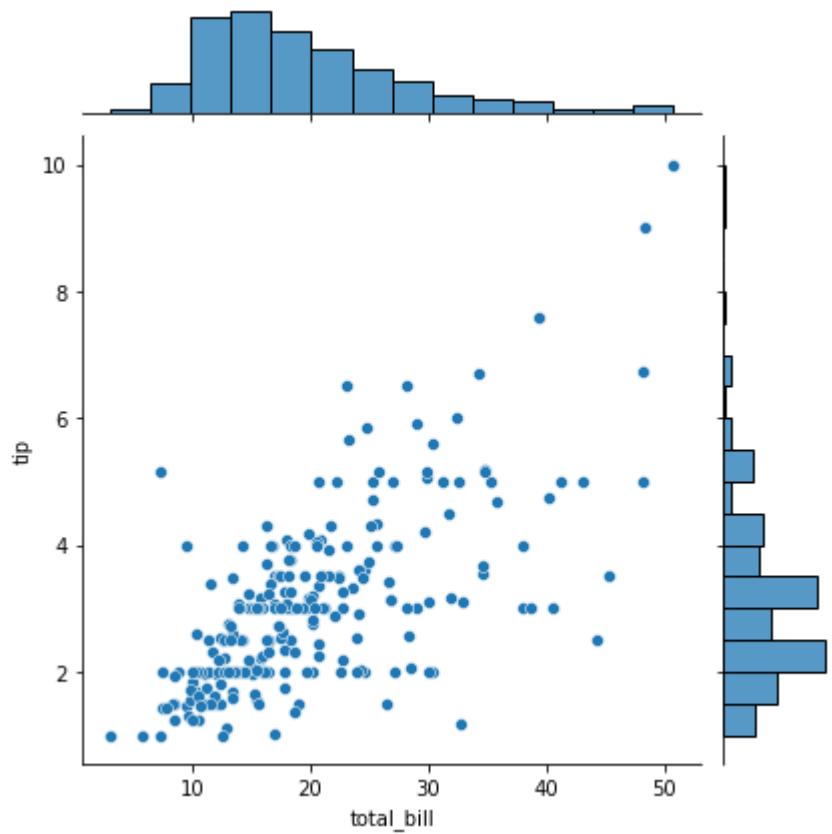


Seaborn-->Joint Plot

In []: *#Meaning of joint plot is making two different plots in same one plots*

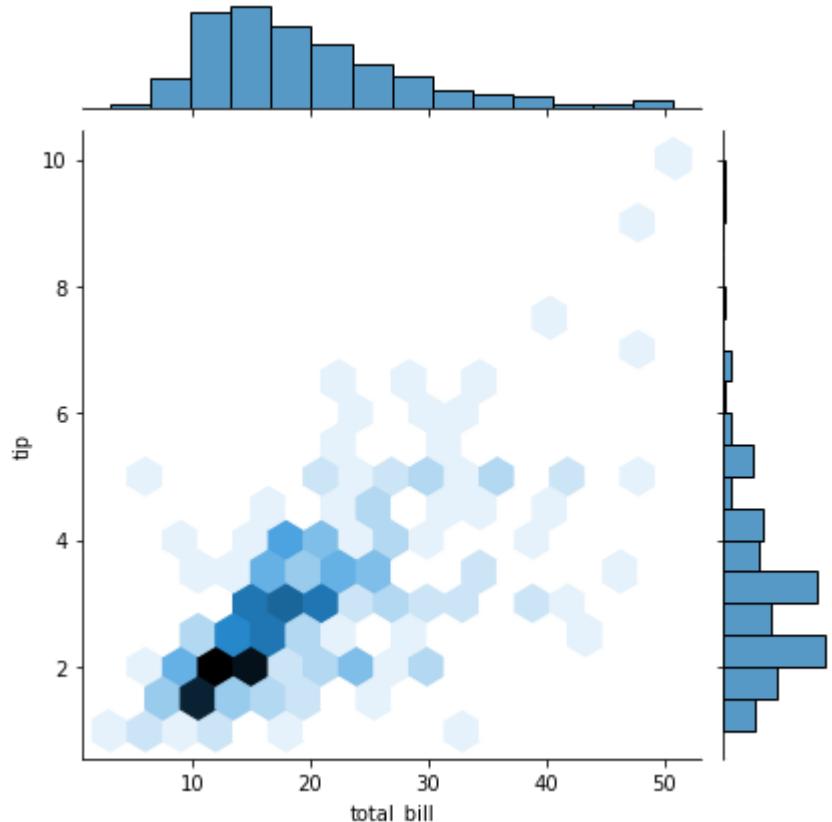
In [155... *#there are three parameters in the joint plot which is x,y and data importantly
#x is one of the variable in the data for comparision
#y is another one of the variable in the data for comparision
#Data is the dataset we use
#It's Like two distribution plots
#In joint plot kind default is scatter plot
sns.jointplot(x="total_bill",y="tip",data=tips)*

Out[155... <seaborn.axisgrid.JointGrid at 0x1bca3bf42e0>



```
In [158...]: #Changing the plot it into hexagon  
sns.jointplot(x="total_bill",y="tip",data=tips,kind="hex")
```

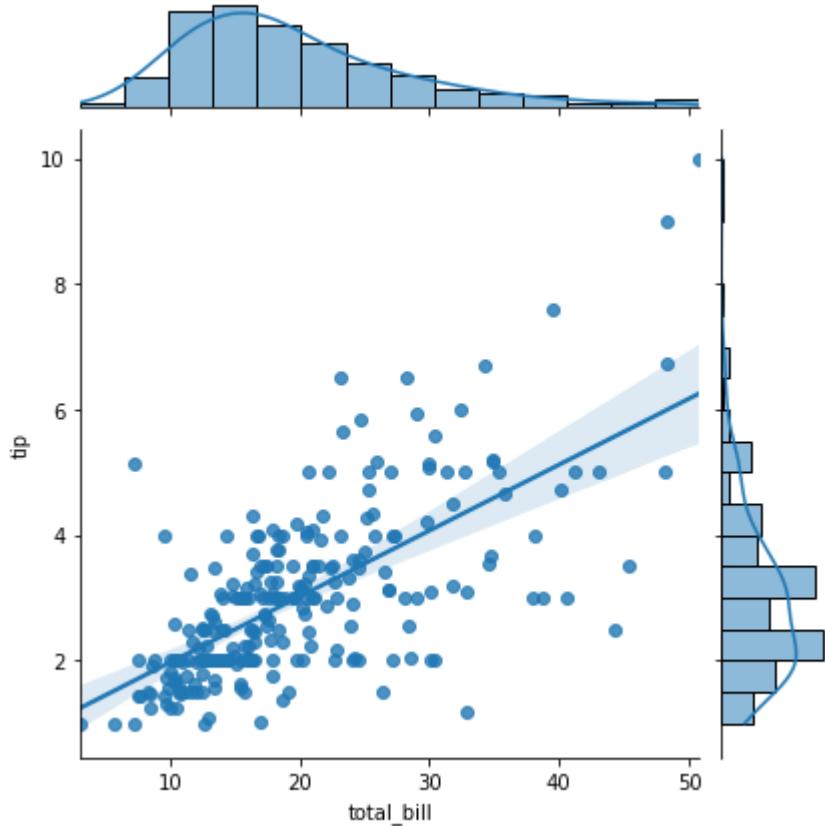
```
Out[158...]: <seaborn.axisgrid.JointGrid at 0x1bca3e65e20>
```



```
In [159...]: # With Regression plot
```

```
sns.jointplot(x="total_bill",y="tip",data=tips,kind="reg")
```

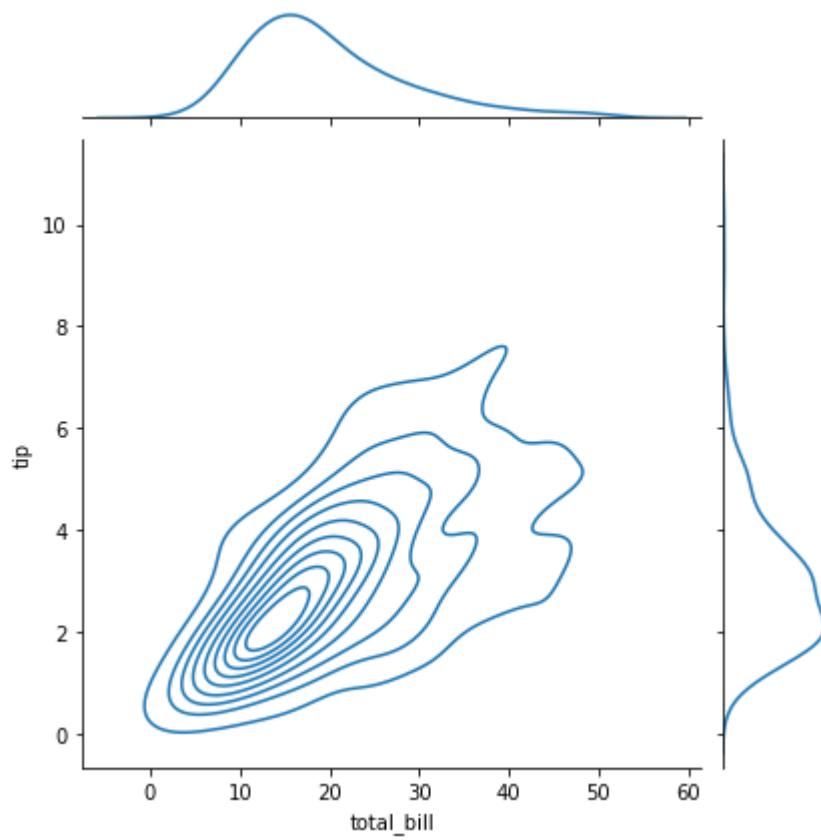
Out[159... <seaborn.axisgrid.JointGrid at 0x1bca3f84fa0>



In [160... #Two dimensional KDE

```
sns.jointplot(x="total_bill",y="tip",data=tips,kind="kde")
```

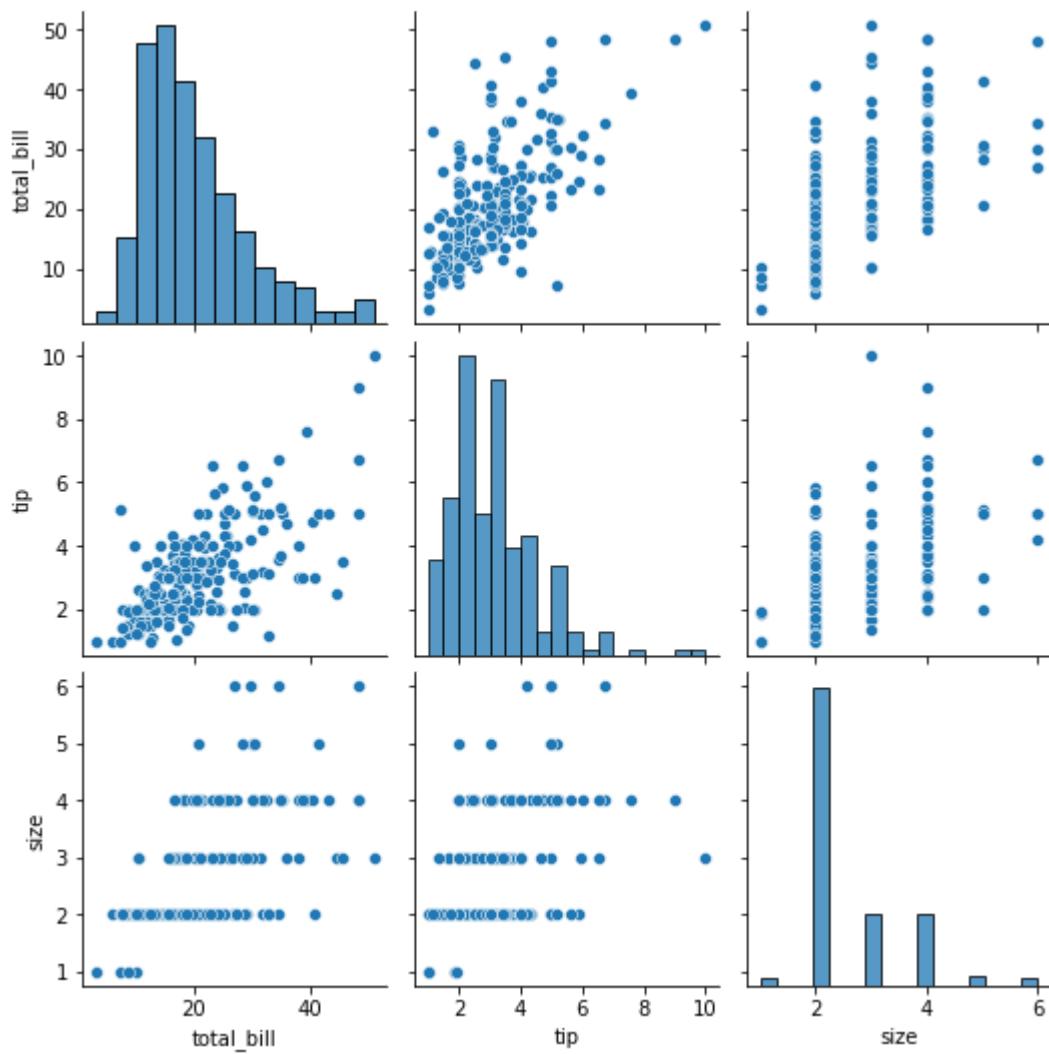
Out[161... <seaborn.axisgrid.JointGrid at 0x1bca40b3d00>



SeaBorn-->PairPlot

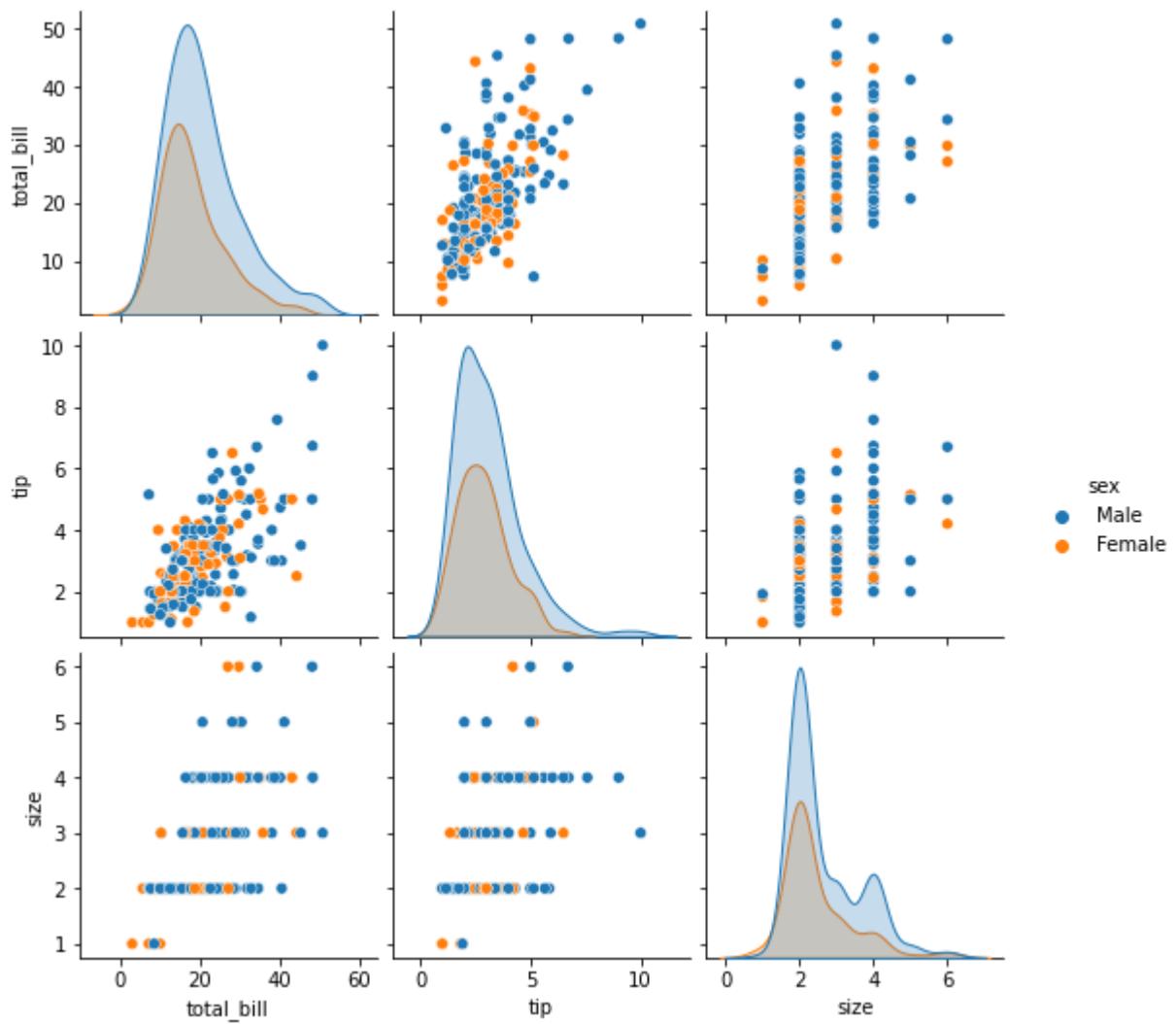
```
In [162...]: #Plotting all the types of possible plots  
sns.pairplot(tips)
```

```
Out[162...]: <seaborn.axisgrid.PairGrid at 0x1bca5138e80>
```



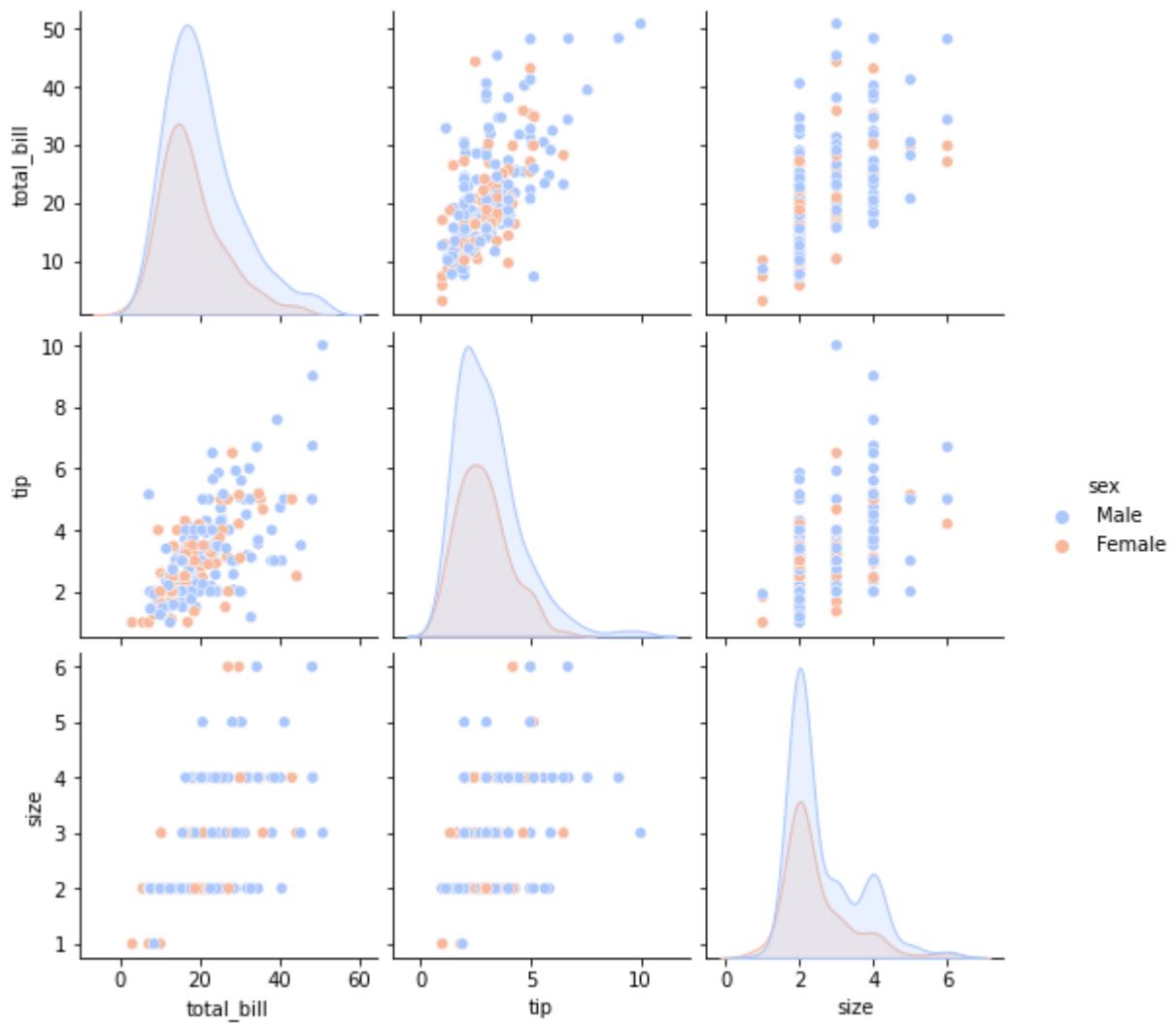
```
In [3]: sns.pairplot(tips,hue="sex")
#Here the graph is coloured according to the sex index
#Blue is taken as male and orange is taken as female
```

```
Out[3]: <seaborn.axisgrid.PairGrid at 0x28e06736fd0>
```



```
In [6]: #Palette is taken as coolwarm for compatible data in the dataset  
sns.pairplot(tips,hue="sex",palette="coolwarm")
```

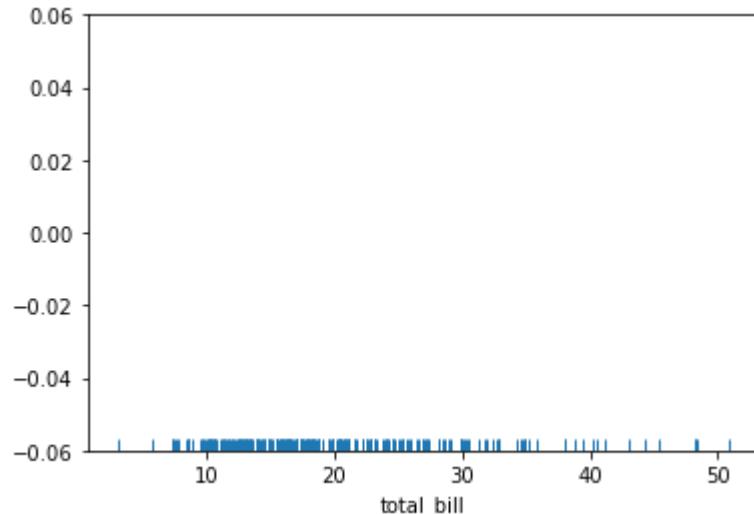
```
Out[6]: <seaborn.axisgrid.PairGrid at 0x28e073a8580>
```



Seaborn-->Rugplot

```
In [7]: #It just draws a dash mark for every points on uniform or unique varient distribution
#i,e Like one single variable
sns.rugplot(tips["total_bill"])
```

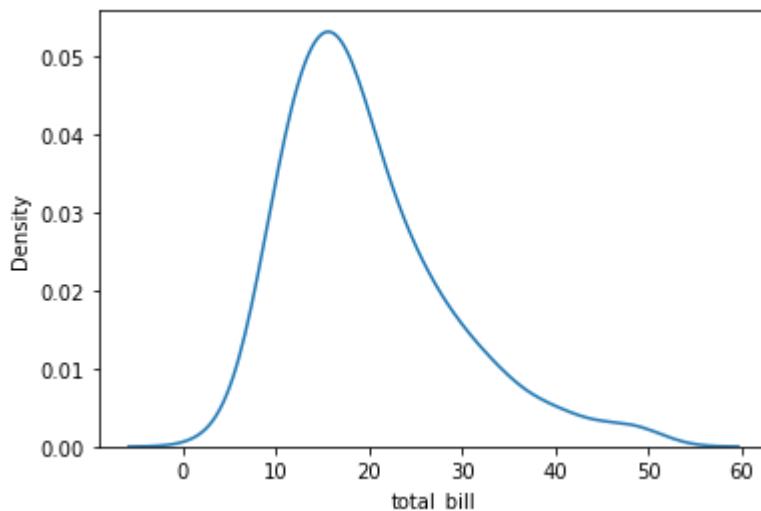
```
Out[7]: <AxesSubplot:xlabel='total_bill'>
```



Seaborn-->KDE(Kernel Distribution Estimation Plots)

```
In [10]: sns.kdeplot(tips["total_bill"])
```

```
Out[10]: <AxesSubplot:xlabel='total_bill', ylabel='Density'>
```



Seaborn-->Categorical Plots

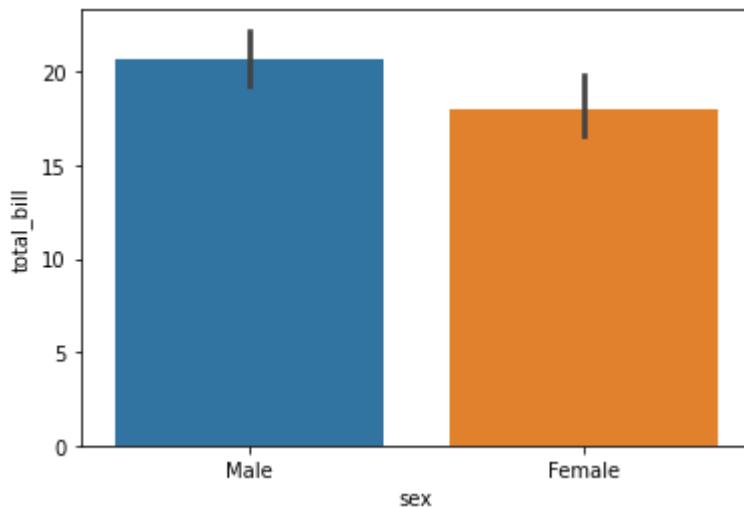
```
In [15]: import seaborn as sns
import numpy as np
%matplotlib inline
tips=sns.load_dataset("tips")
tips.head()
```

```
Out[15]:   total_bill  tip    sex  smoker  day    time  size
0      16.99  1.01  Female     No  Sun  Dinner     2
1      10.34  1.66    Male     No  Sun  Dinner     3
2      21.01  3.50    Male     No  Sun  Dinner     3
3      23.68  3.31    Male     No  Sun  Dinner     2
4      24.59  3.61  Female     No  Sun  Dinner     4
```

Seaborn-Barplot

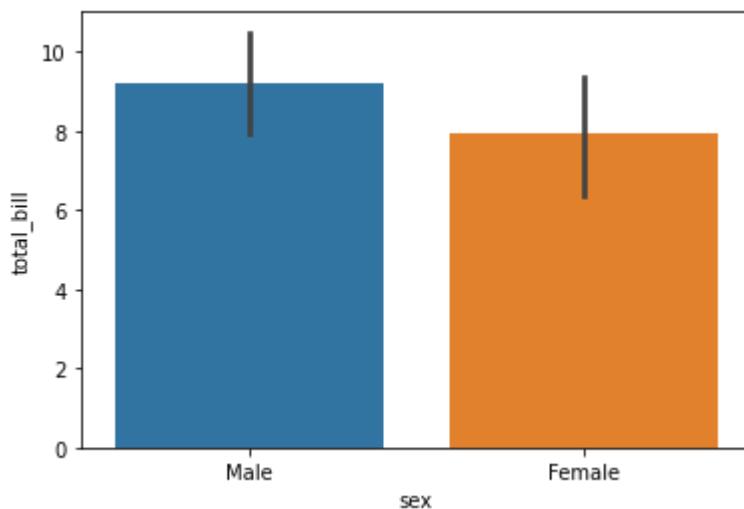
```
In [14]: #Creating an barplot for males and females with regarding to total_bill
sns.barplot(x="sex",y="total_bill",data=tips)
```

```
Out[14]: <AxesSubplot:xlabel='sex', ylabel='total_bill'>
```



```
In [16]: sns.barplot(x="sex",y="total_bill",data=tips,estimator=np.std)
```

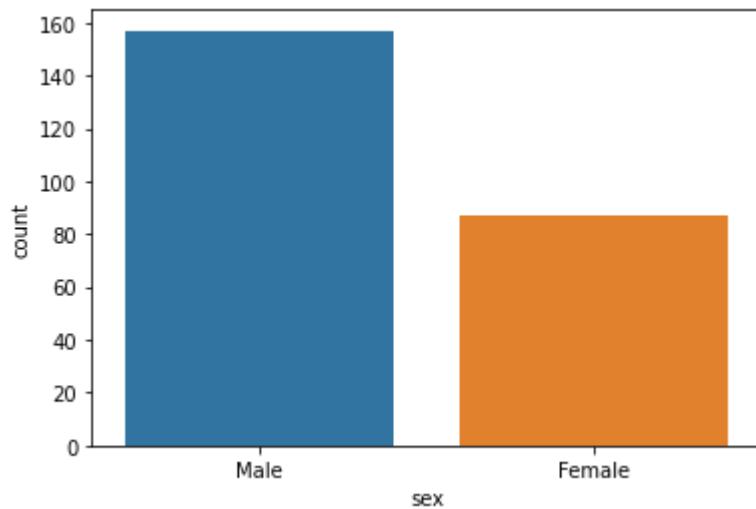
```
Out[16]: <AxesSubplot:xlabel='sex', ylabel='total_bill'>
```



Seaborn->CountPlot

```
In [17]: #Counting the number of males and females in the data  
sns.countplot(x="sex",data=tips)
```

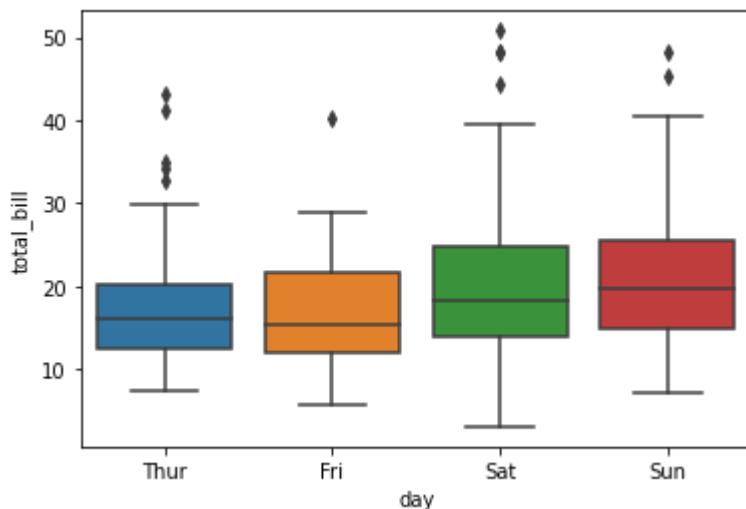
```
Out[17]: <AxesSubplot:xlabel='sex', ylabel='count'>
```



Seaborn->BoxPlot

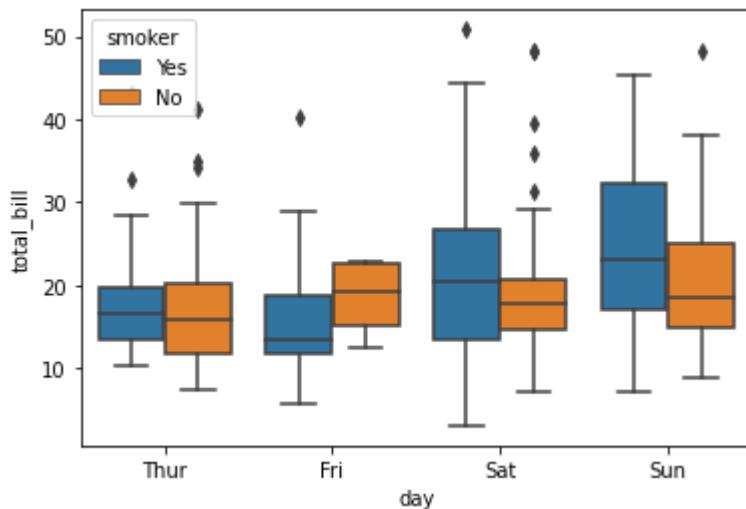
```
In [18]: #Create a box plot for the comparision of the total_bills in each day
sns.boxplot(x="day",y="total_bill",data=tips)
```

```
Out[18]: <AxesSubplot:xlabel='day', ylabel='total_bill'>
```



```
In [19]: #here hue is used in the data weather the person is a smoker or not
sns.boxplot(x="day",y="total_bill",data=tips,hue="smoker")
```

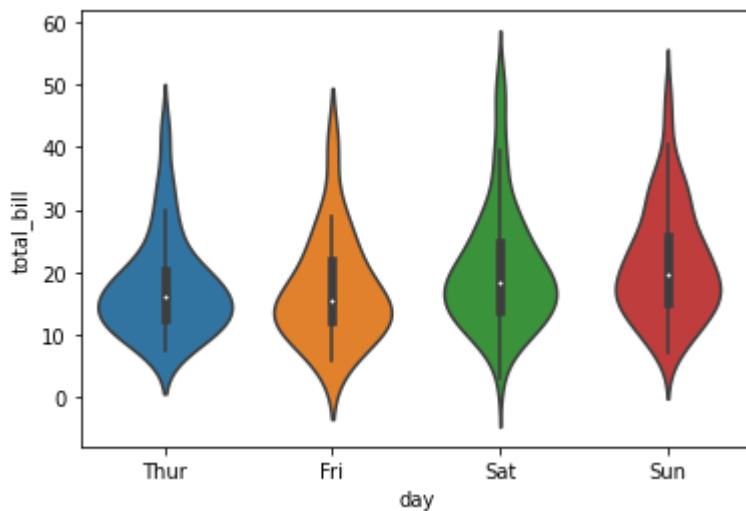
```
Out[19]: <AxesSubplot:xlabel='day', ylabel='total_bill'>
```



Seaborn->Violin Plot

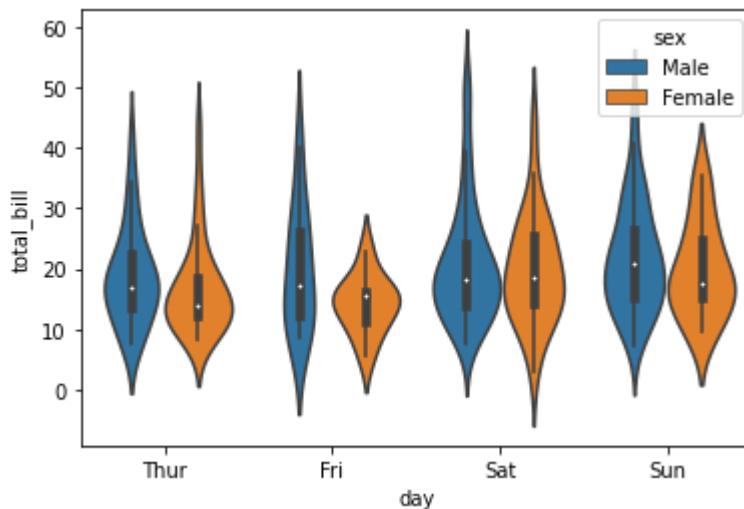
```
In [20]: sns.violinplot(x="day",y="total_bill",data=tips)
```

```
Out[20]: <AxesSubplot:xlabel='day', ylabel='total_bill'>
```



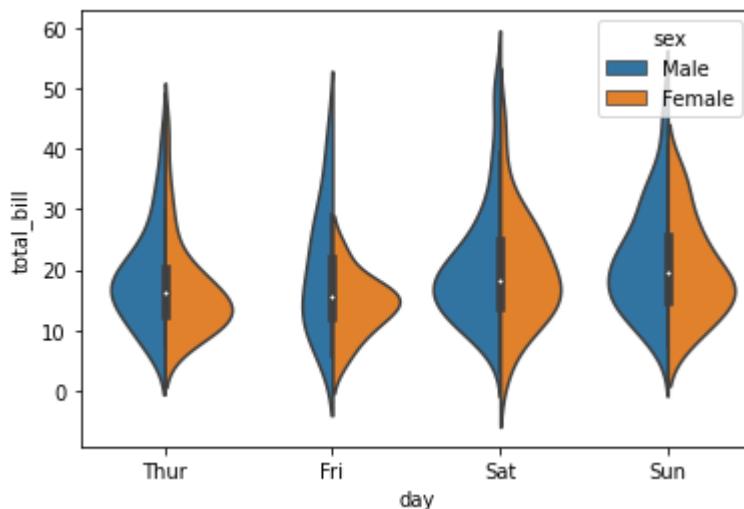
```
In [21]: #Hue is taken as sex in the comparision of male or female
sns.violinplot(x="day",y="total_bill",data=tips,hue="sex")
```

```
Out[21]: <AxesSubplot:xlabel='day', ylabel='total_bill'>
```



```
In [22]: #Hue is taken as sex in the comparision of male or female
#We combine the two plots for more understanding
sns.violinplot(x="day",y="total_bill",data=tips,hue="sex",split=True)
```

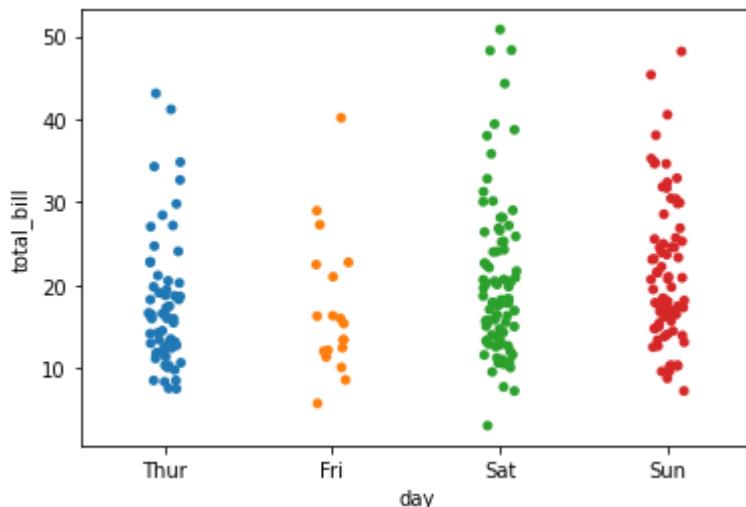
```
Out[22]: <AxesSubplot:xlabel='day', ylabel='total_bill'>
```



Seaborn-->Strip plot

```
In [25]: sns.stripplot(x="day",y="total_bill",data=tips)
```

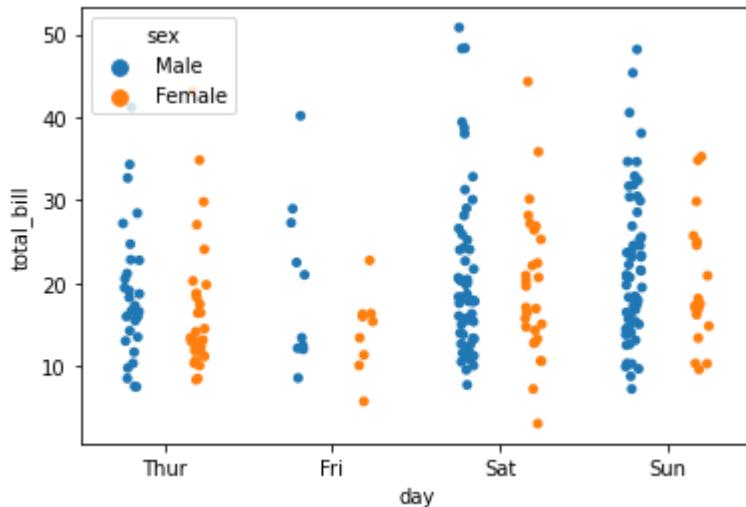
```
Out[25]: <AxesSubplot:xlabel='day', ylabel='total_bill'>
```



```
In [30]: #Here we take hue as sex for the comparision of male and female
#We use split in the parameter to make the plot more understandable
sns.stripplot(x="day",y="total_bill",data=tips,hue="sex",split=True)
```

C:\Users\srena\anaconda3\lib\site-packages\seaborn\categorical.py:2792: UserWarning: The `split` parameter has been renamed to `dodge`.
warnings.warn(msg, UserWarning)

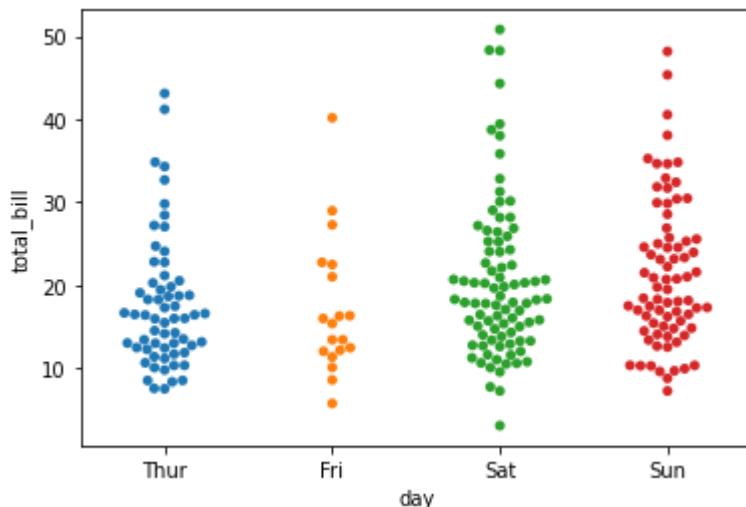
```
Out[30]: <AxesSubplot:xlabel='day', ylabel='total_bill'>
```



Seaborn->Swarm Plot

```
In [31]: sns.swarmplot(x="day",y="total_bill",data=tips)
```

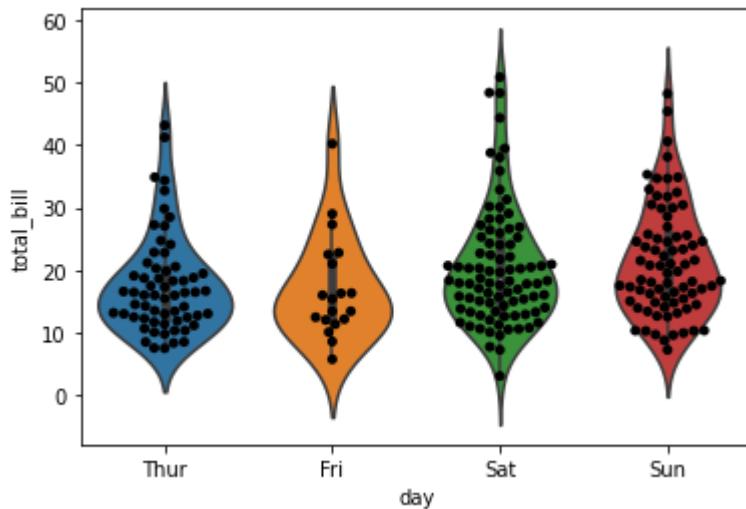
```
Out[31]: <AxesSubplot:xlabel='day', ylabel='total_bill'>
```



```
In [34]: #Adding violin plot in swarmplot for differentiation
```

```
sns.violinplot(x="day",y="total_bill",data=tips)
sns.swarmplot(x="day",y="total_bill",data=tips,color="black")
```

```
Out[34]: <AxesSubplot:xlabel='day', ylabel='total_bill'>
```

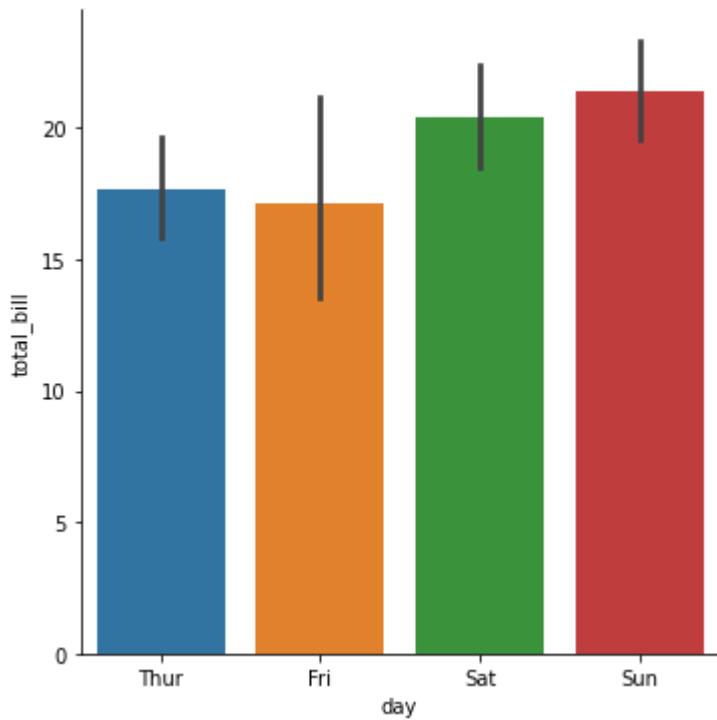


Seaborn->Factor Plot

```
In [38]: #Creating an factor plot with kind as bars used for differentiation
sns.factorplot(x="day",y="total_bill",data=tips,kind="bar")
```

```
C:\Users\srena\anaconda3\lib\site-packages\seaborn\categorical.py:3704: UserWarning: The
`factorplot` function has been renamed to `catplot`. The original name will be removed i
n a future release. Please update your code. Note that the default `kind` in `factorplot
` ('point') has changed `strip` in `catplot`.
warnings.warn(msg)
```

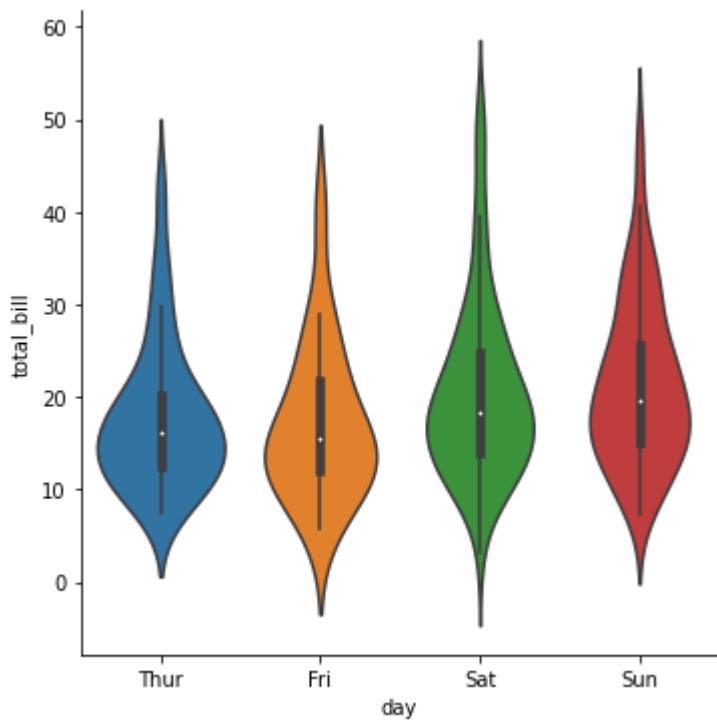
```
Out[38]: <seaborn.axisgrid.FacetGrid at 0x28e09575bb0>
```



```
In [40]: #Making the factorplot as violin as kind
sns.factorplot(x="day",y="total_bill",data=tips,kind="violin")
```

C:\Users\srena\anaconda3\lib\site-packages\seaborn\categorical.py:3704: UserWarning: The `factorplot` function has been renamed to `catplot`. The original name will be removed in a future release. Please update your code. Note that the default `kind` in `factorplot` ('point') has changed to 'strip' in `catplot`.
warnings.warn(msg)

```
Out[40]: <seaborn.axisgrid.FacetGrid at 0x28e0a65e610>
```



```
In [5]: import seaborn as sns
%matplotlib inline
tips=sns.load_dataset("tips")
```

```
In [4]: flights=sns.load_dataset("flights")
```

```
In [6]: tips.head()
```

```
Out[6]:   total_bill  tip  sex  smoker  day  time  size
0      16.99  1.01  Female    No  Sun Dinner     2
1      10.34  1.66   Male    No  Sun Dinner     3
2      21.01  3.50   Male    No  Sun Dinner     3
3      23.68  3.31   Male    No  Sun Dinner     2
4      24.59  3.61 Female    No  Sun Dinner     4
```

```
In [7]: flights.head()
```

```
Out[7]:   year  month  passengers
0  1949    Jan        112
1  1949    Feb        118
2  1949    Mar        132
3  1949    Apr        129
4  1949    May        121
```

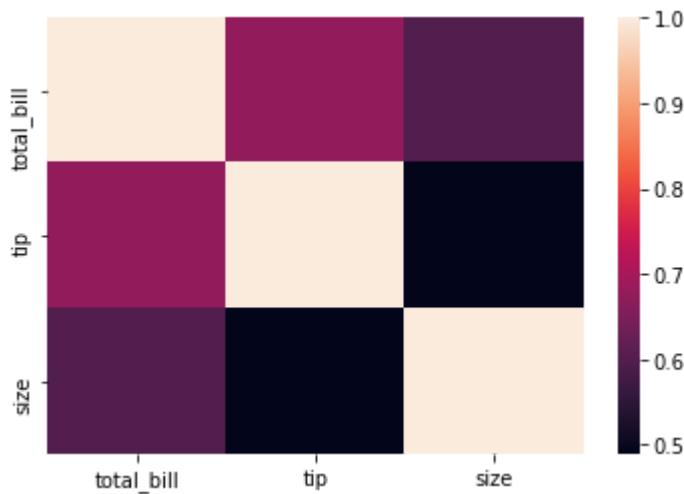
Seaborn-->Matrix Plot-->HeatMap

```
In [12]: #Correlation of the data
tc=tips.corr()
tc
```

```
Out[12]:      total_bill      tip      size
total_bill  1.000000  0.675734  0.598315
tip        0.675734  1.000000  0.489299
size       0.598315  0.489299  1.000000
```

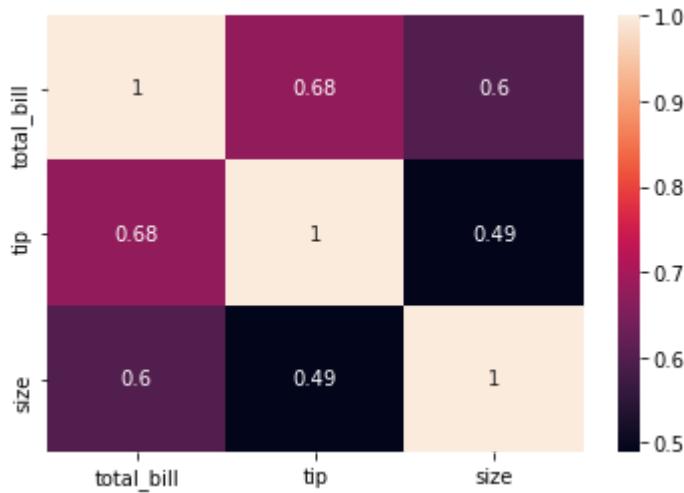
```
In [14]: #Creating an heat map which is most commonly used to find values which the color
sns.heatmap(tc)
```

```
Out[14]: <AxesSubplot:>
```



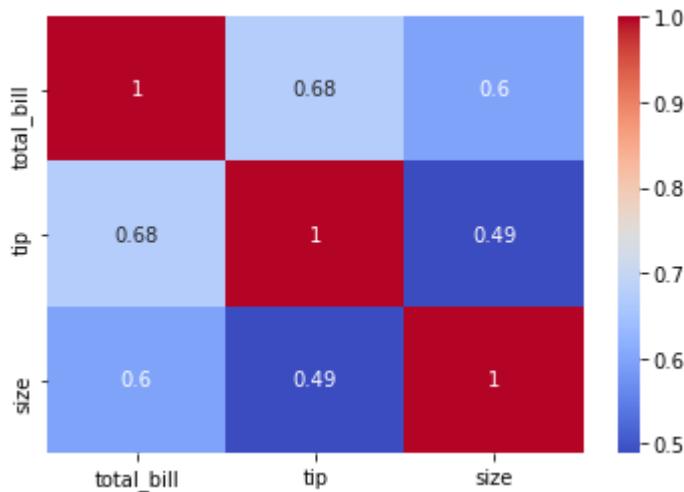
```
In [16]: #Annotations in heat map
sns.heatmap(tc, annot=True)
```

Out[16]: <AxesSubplot:>



```
In [17]: #coolwarm is used to make it in understandable way
sns.heatmap(tc, annot=True, cmap="coolwarm")
```

Out[17]: <AxesSubplot:>

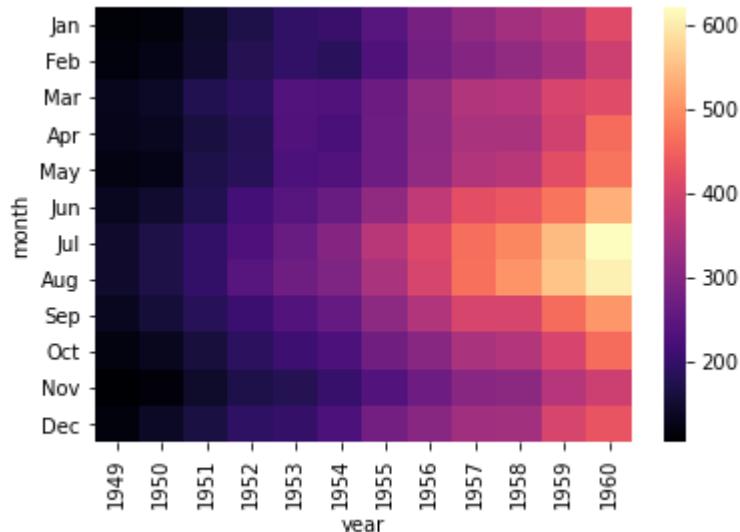


```
fp=flights.pivot_table(index="month",columns="year",values="passengers")
```

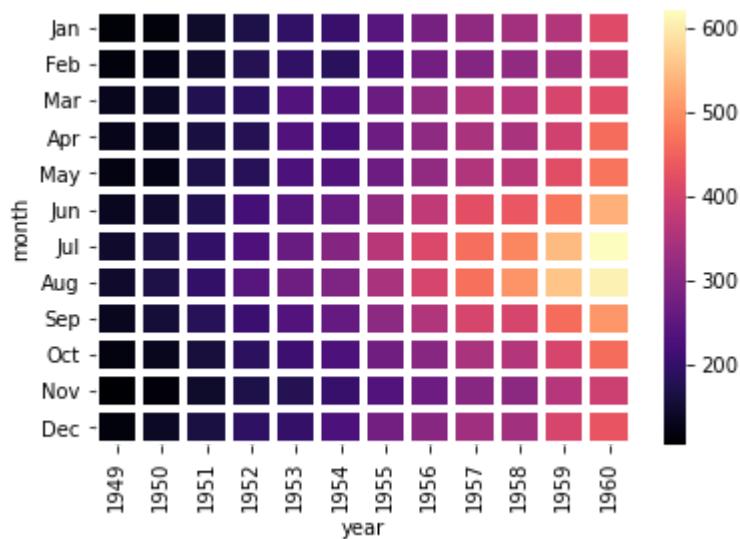
In [20]:

In [23]: #Magma in cmap to reduce the colour
sns.heatmap(fp,cmap="magma")

Out[23]: <AxesSubplot:xlabel='year', ylabel='month'>

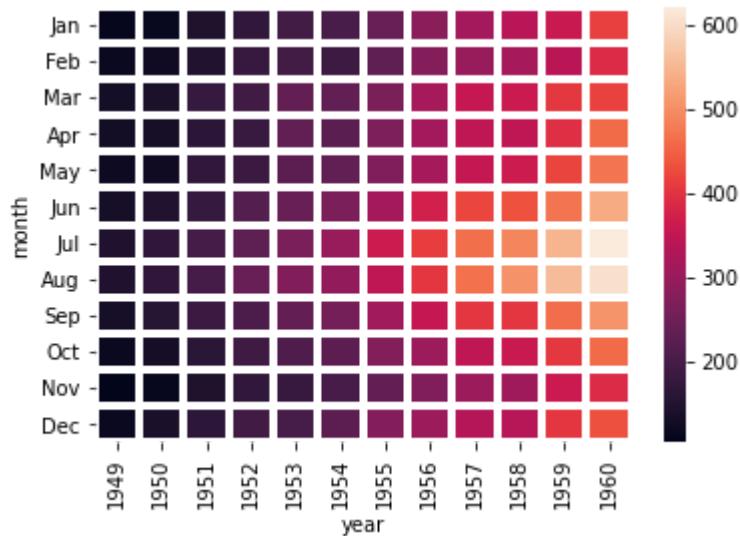
In [26]: #For the separation of the heat map we use Linecolor and Linewidth
sns.heatmap(fp,cmap="magma",linecolor="white",linewidth=3)

Out[26]: <AxesSubplot:xlabel='year', ylabel='month'>



In [27]: sns.heatmap(fp,linecolor="white",linewidth=3)

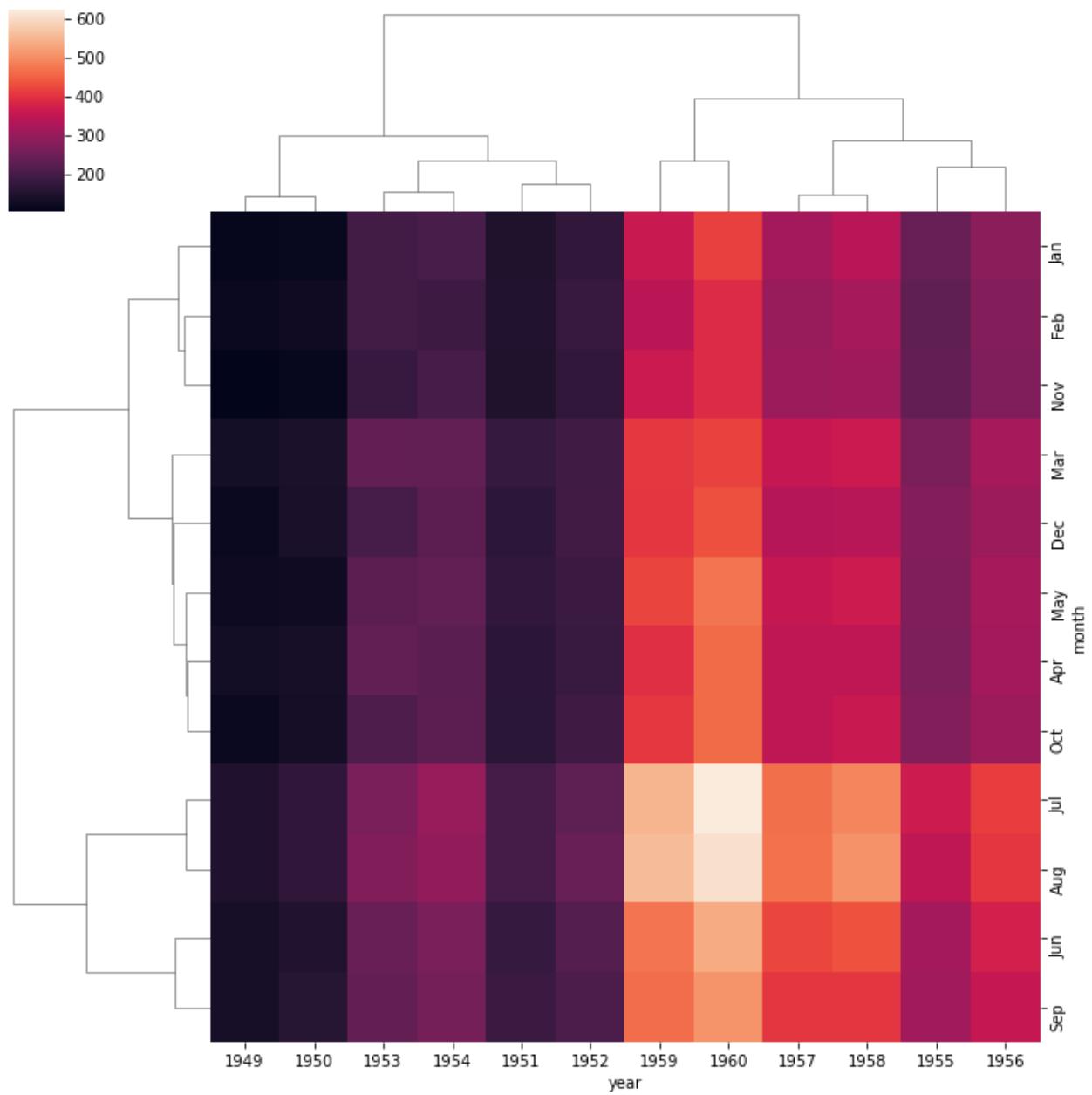
Out[27]: <AxesSubplot:xlabel='year', ylabel='month'>



Seaborn-->Cluster Map

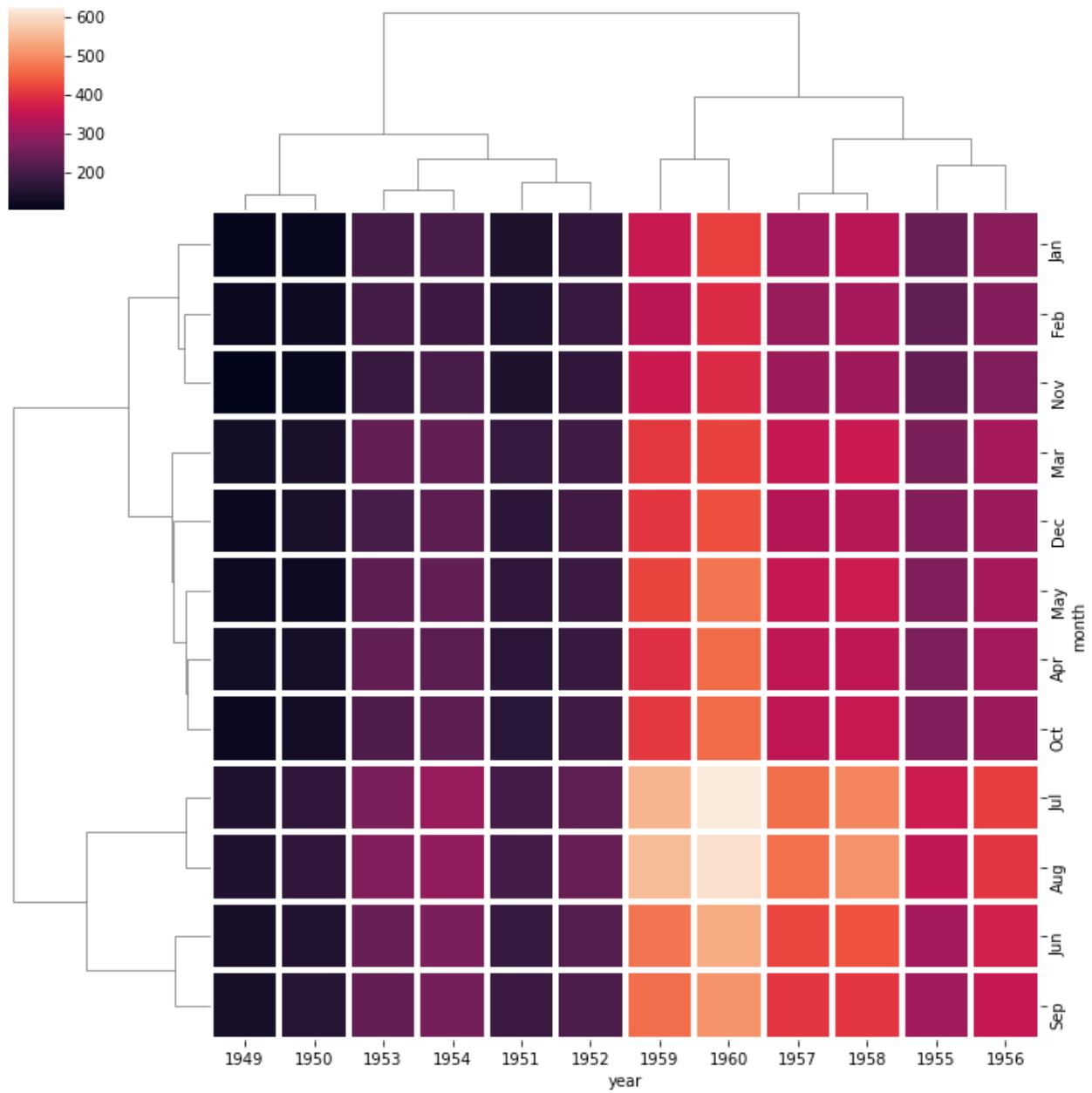
```
In [30]: sns.clustermap(fp)
```

```
Out[30]: <seaborn.matrix.ClusterGrid at 0x1bf5b9b2d00>
```



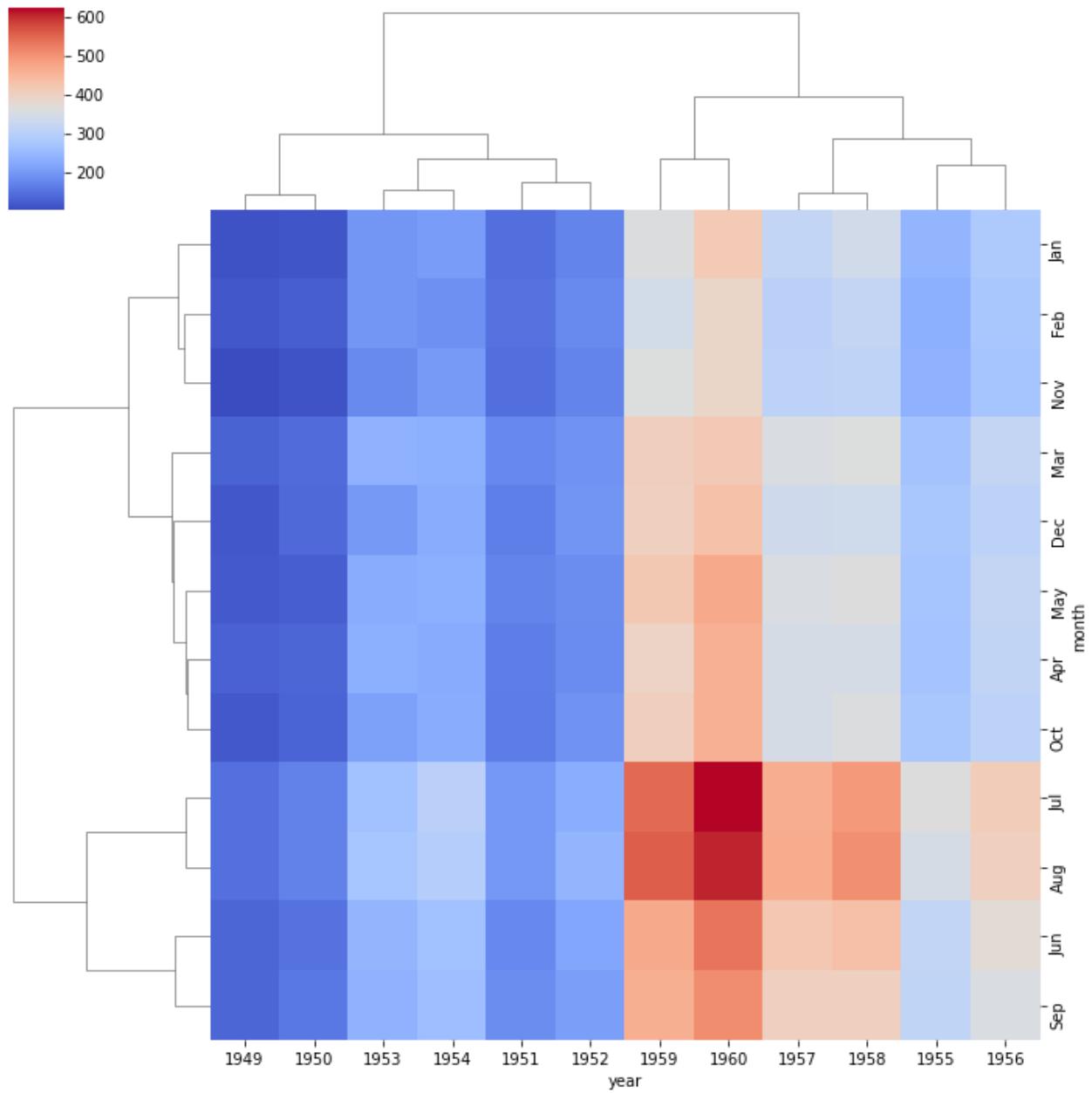
In [29]: `#It is a hierarchical cluster which gets connected between them
sns.clustermap(fp, linecolor="white", linewidth=3)`

Out[29]: <seaborn.matrix.ClusterGrid at 0x1bf5b98e940>



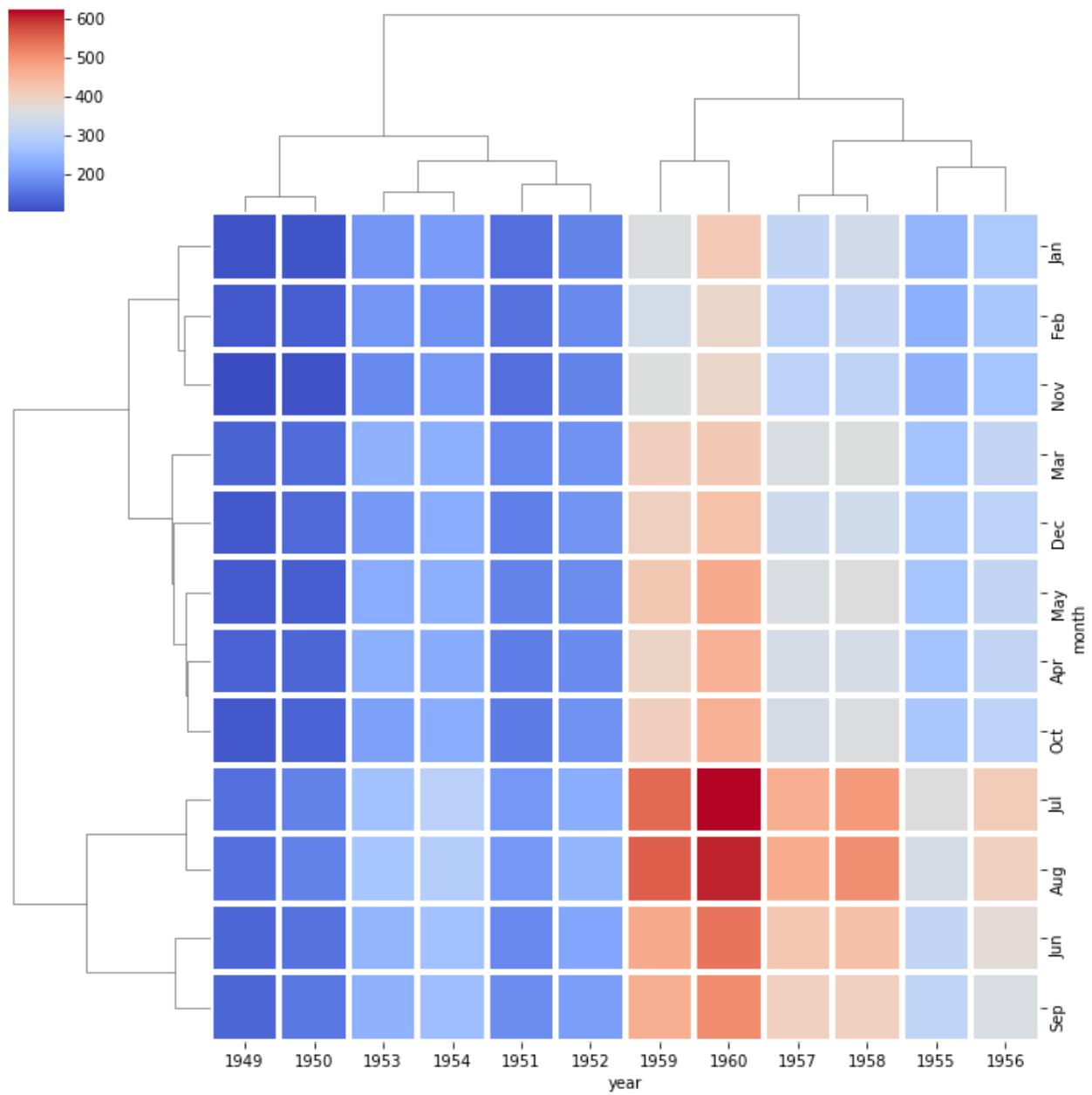
```
In [31]: sns.clustermap(fp,cmap="coolwarm")
```

```
Out[31]: <seaborn.matrix.ClusterGrid at 0x1bf5c20af0>
```



```
In [32]: sns.clustermap(fp,cmap="coolwarm",linecolor="white", linewidth=3)
```

```
Out[32]: <seaborn.matrix.ClusterGrid at 0x1bf5b889ac0>
```



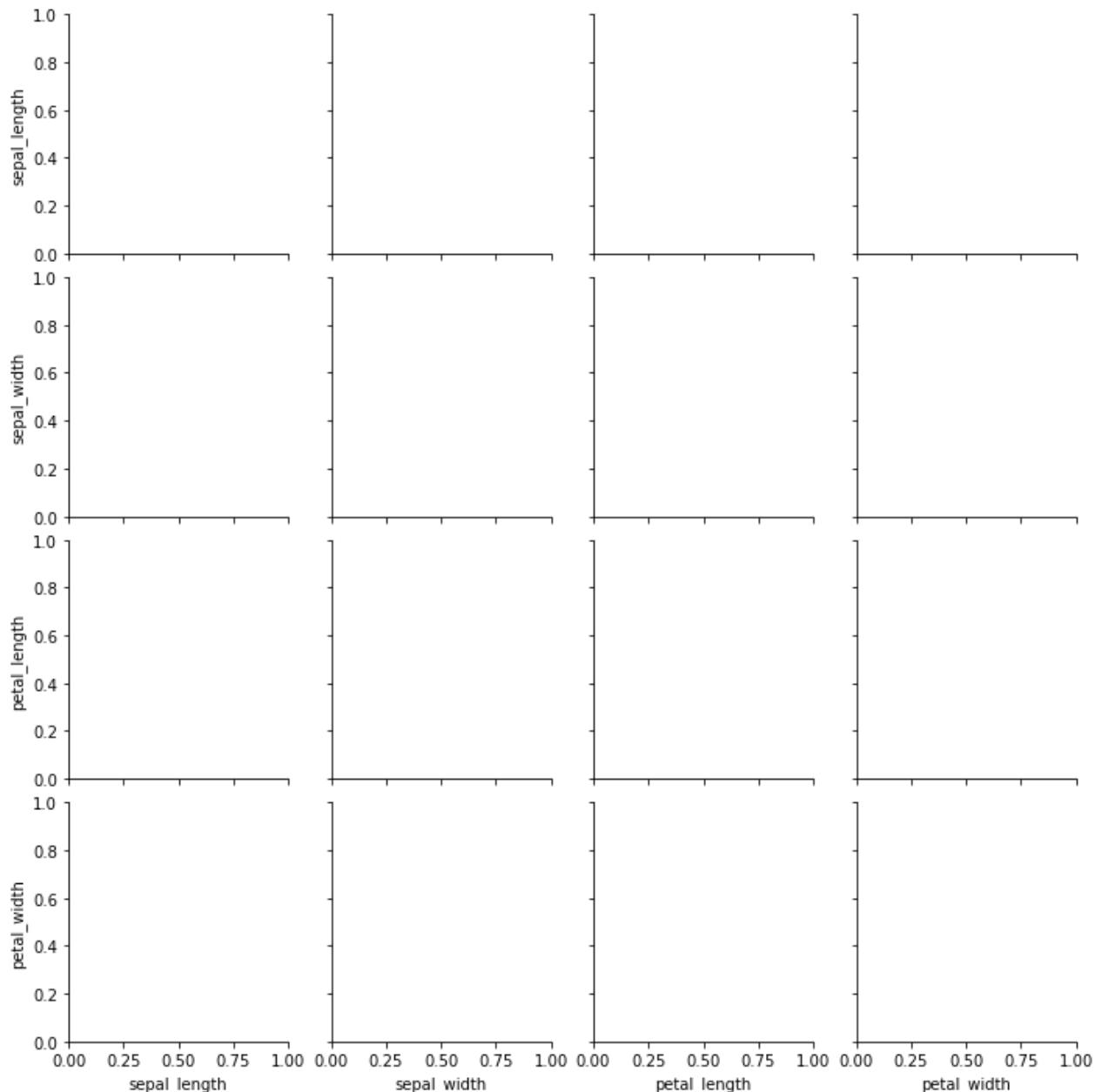
Seaborns->Grids

```
In [40]: import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
iris=sns.load_dataset("iris")
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

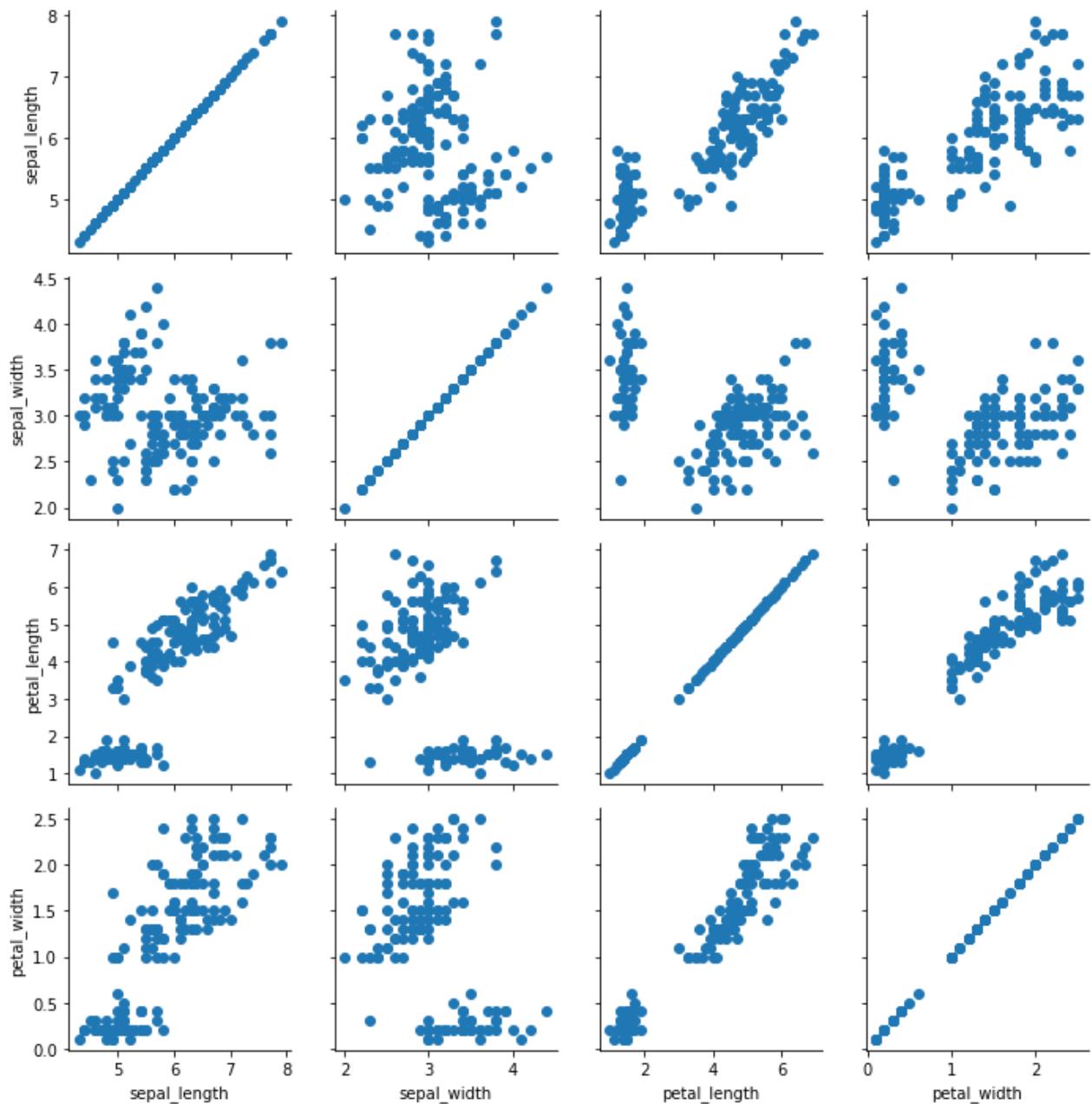
```
In [37]: #PairGrid is the simplified plot of pair plot  
sns.PairGrid(iris)
```

```
Out[37]: <seaborn.axisgrid.PairGrid at 0x1bf5e9cc4c0>
```



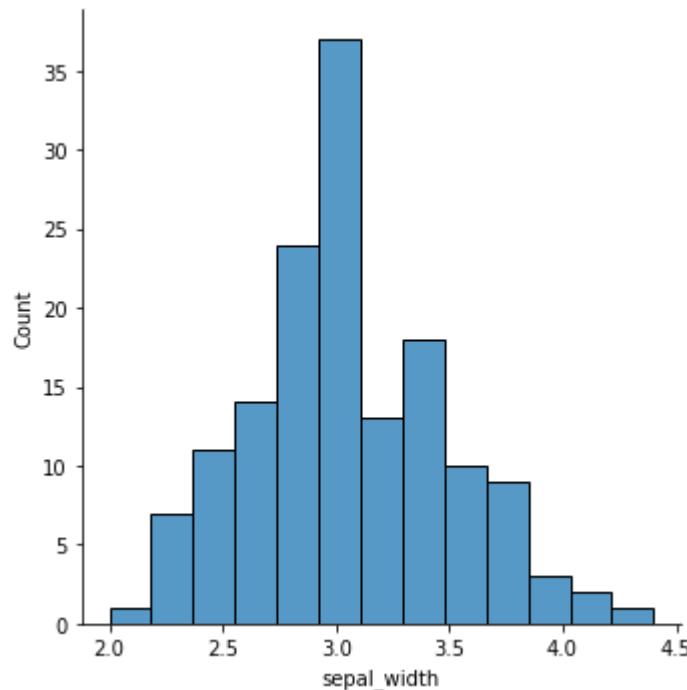
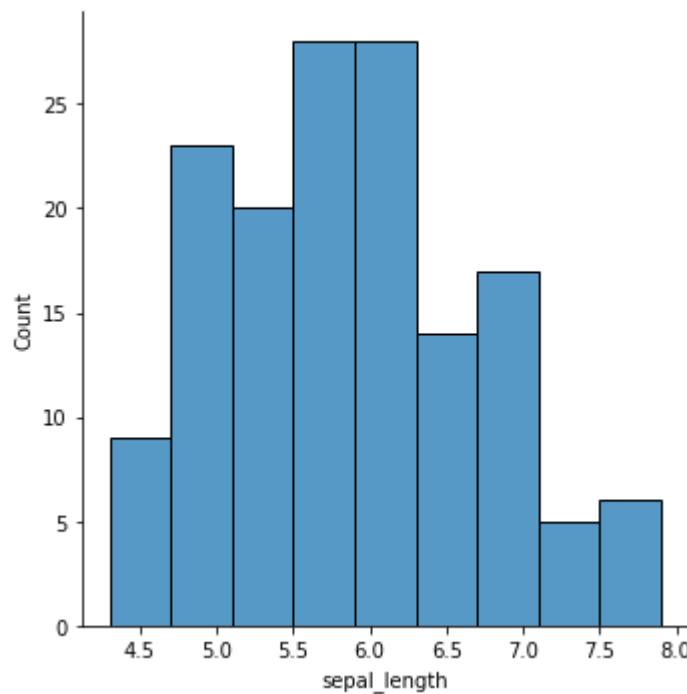
```
In [41]: #Here the Pair Grid iwill give an empty plot graph  
#We can add any plots we want in grids  
g=sns.PairGrid(iris)  
g.map(plt.scatter)
```

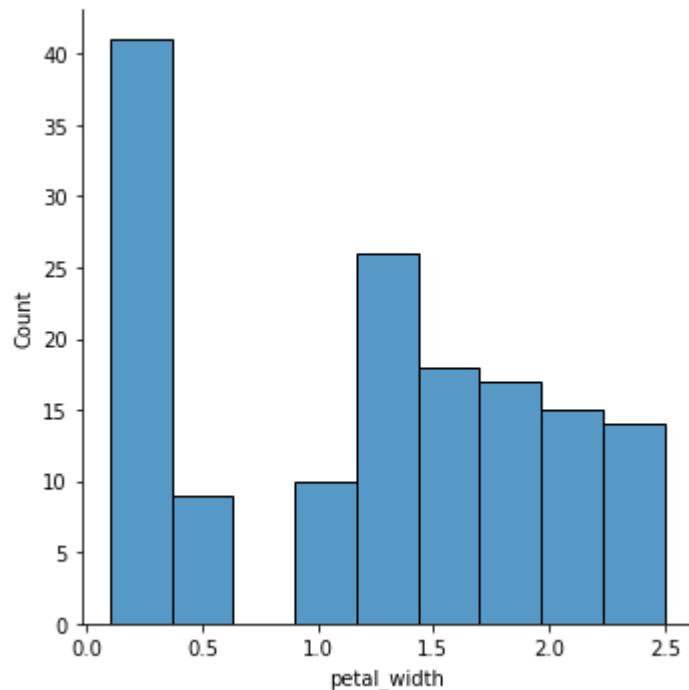
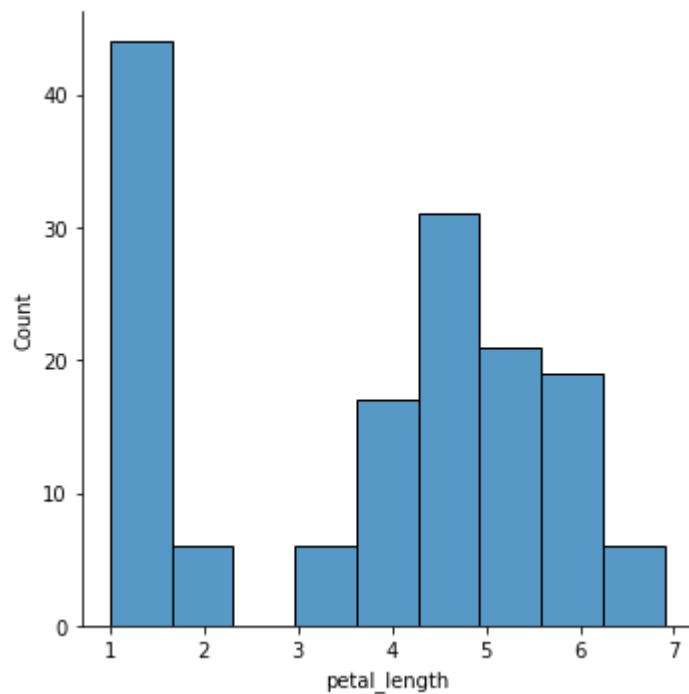
```
Out[41]: <seaborn.axisgrid.PairGrid at 0x1bf5fa11880>
```

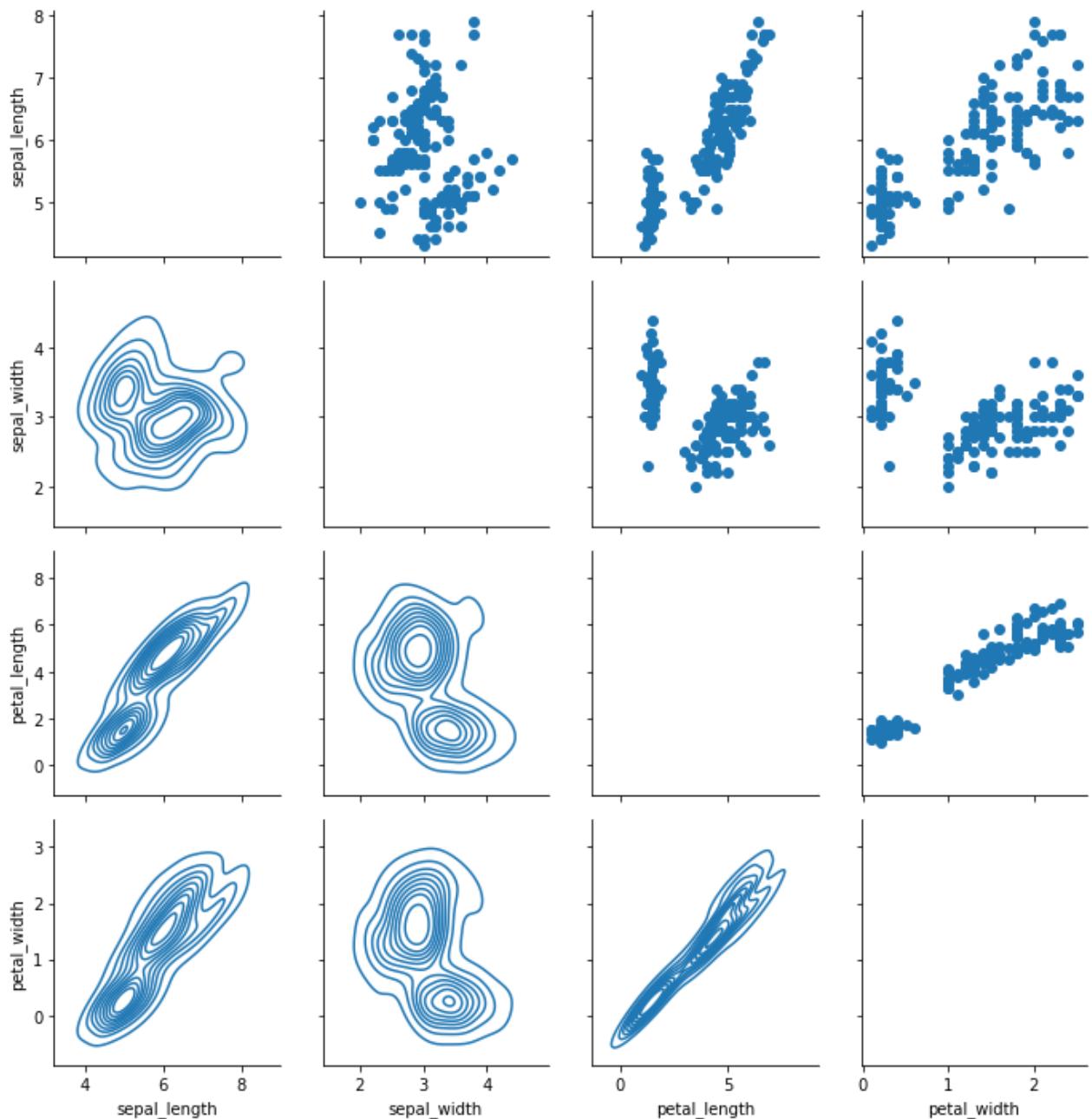


```
In [43]: #Creating an pairgrid adding distplots in diagonal  
#Adding scatter plot in upper adn kdeplot in Lower  
g=sns.PairGrid(iris)  
g.map_diag(sns.displot)  
g.map_upper(plt.scatter)  
g.map_lower(sns.kdeplot)
```

```
Out[43]: <seaborn.axisgrid.PairGrid at 0x1bf61a14c40>
```







```
In [45]: tips=sns.load_dataset("tips")
```

```
In [46]: tips.head()
```

```
Out[46]:
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

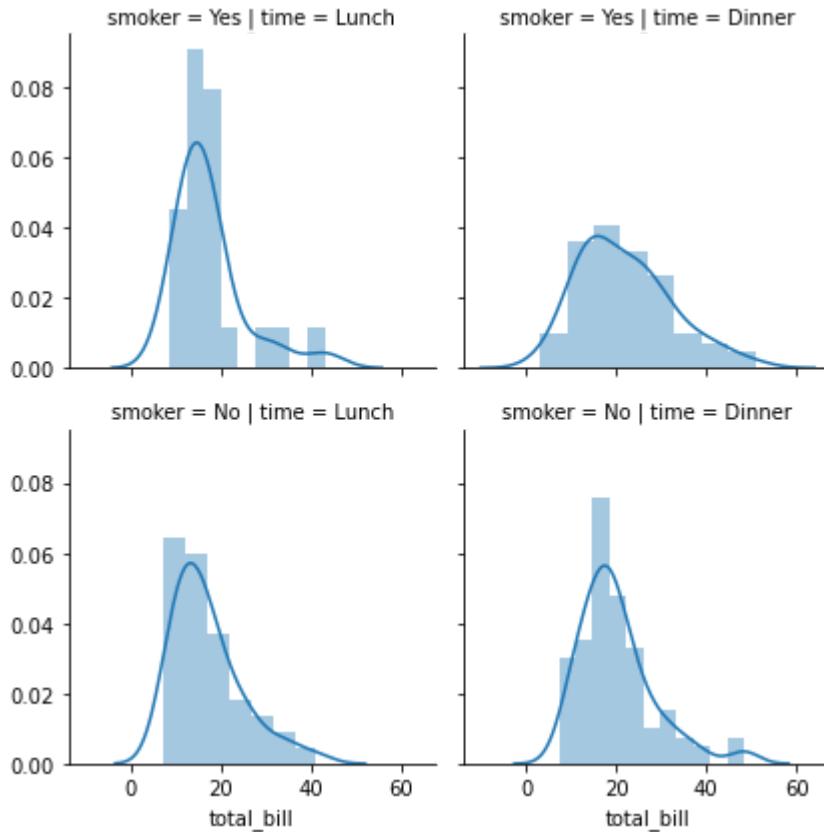
Seaborn-->FacetGrid

In [51]:

```
#It creates a empt grids accoridng to the data
g=sns.FacetGrid(data=tips,col="time",row="smoker")
g.map(sns.distplot,"total_bill")
```

C:\Users\srena\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
 warnings.warn(msg, FutureWarning)
 C:\Users\srena\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
 warnings.warn(msg, FutureWarning)
 C:\Users\srena\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
 warnings.warn(msg, FutureWarning)
 C:\Users\srena\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
 warnings.warn(msg, FutureWarning)

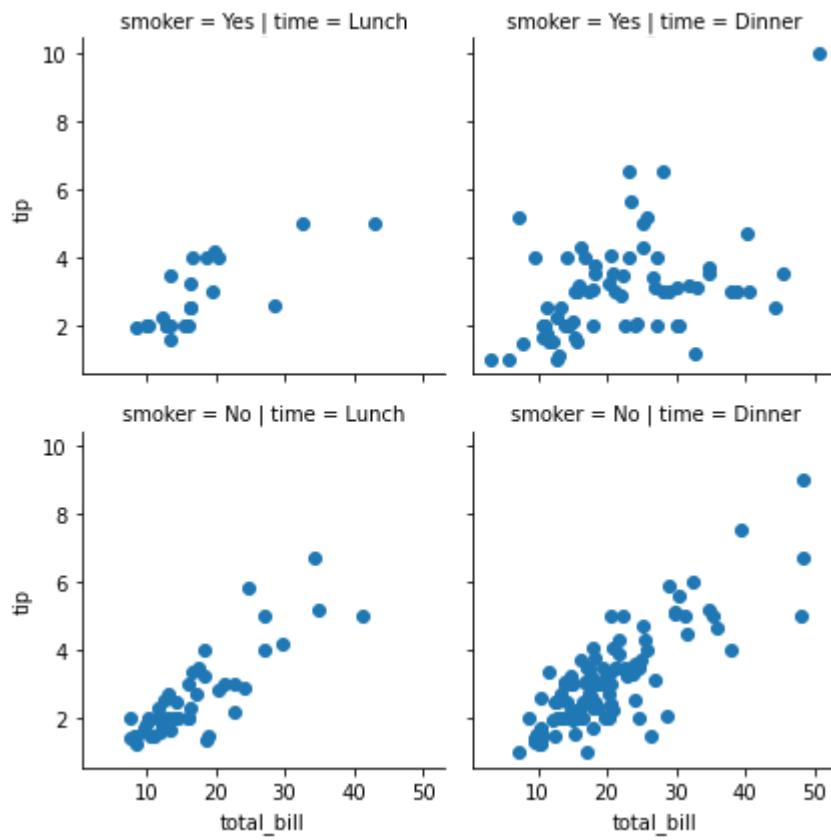
Out[51]: <seaborn.axisgrid.FacetGrid at 0x1bf6244a9d0>



In [52]:

```
g=sns.FacetGrid(data=tips,col="time",row="smoker")
g.map(plt.scatter,"total_bill","tip")
```

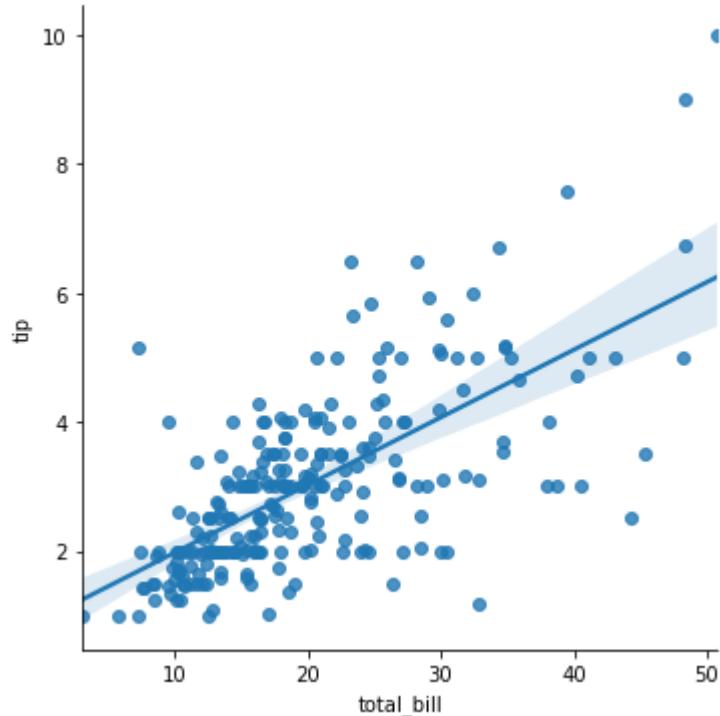
Out[52]: <seaborn.axisgrid.FacetGrid at 0x1bf6356b460>



Seaborn-->Regression Plots

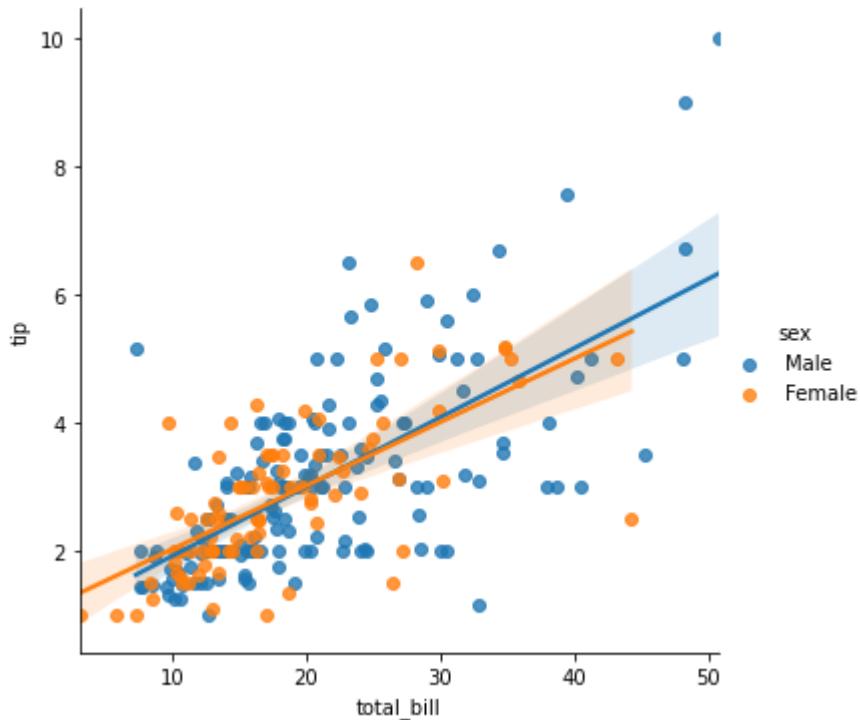
```
In [53]: ##### LM Plot
sns.lmplot(x="total_bill",y="tip",data=tips)
```

```
Out[53]: <seaborn.axisgrid.FacetGrid at 0x1bf5fc6e940>
```



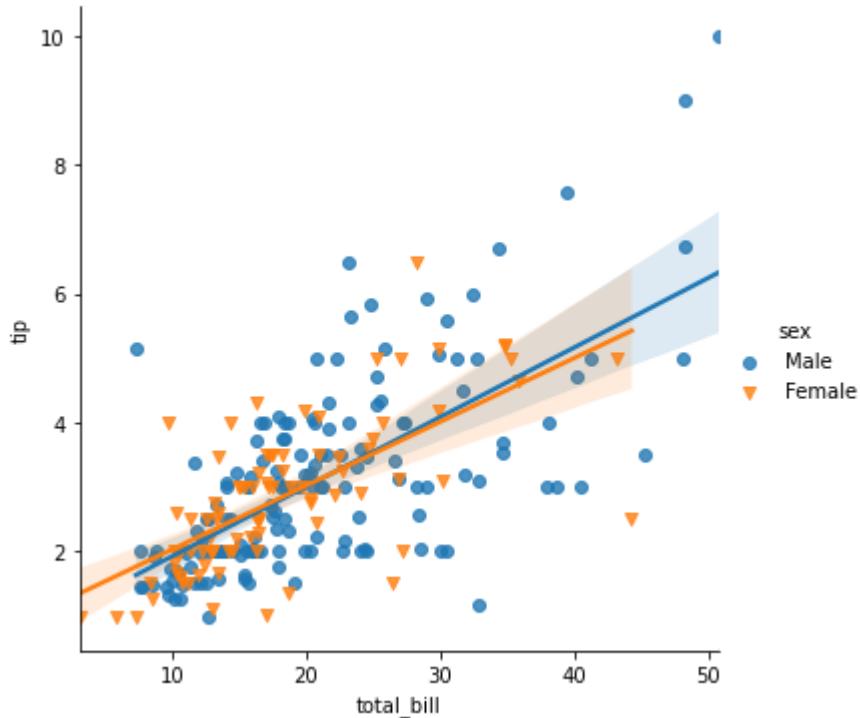
```
In [54]: #splitment of male and female in the plots
sns.lmplot(x="total_bill",y="tip",data=tips,hue="sex")
```

Out[54]: <seaborn.axisgrid.FacetGrid at 0x1bf62431f10>



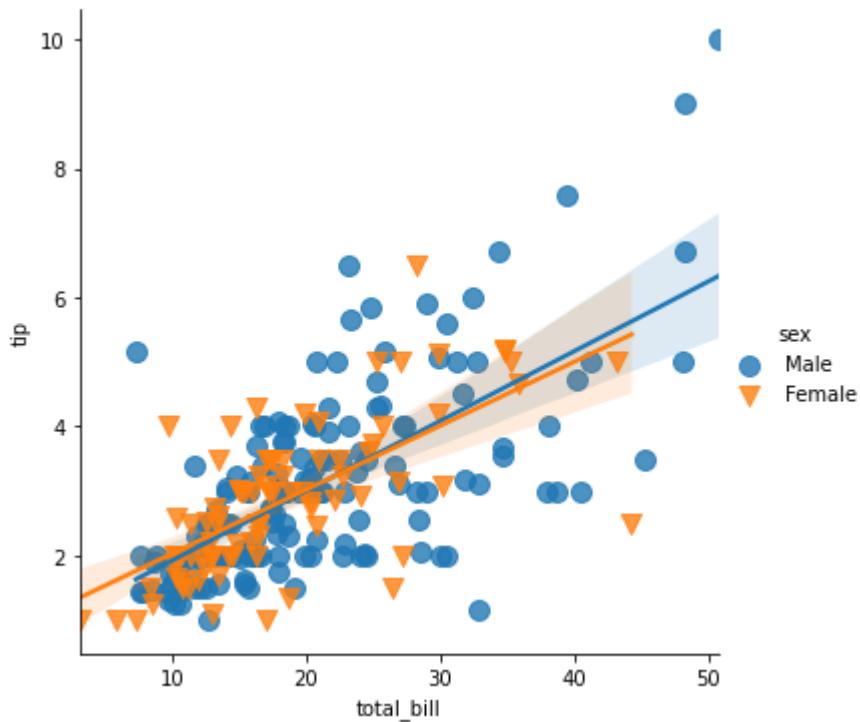
```
In [56]: #Here we changed the shapes to differentiate
# We use markers to change o for male and v for female
sns.lmplot(x="total_bill",y="tip",data=tips,hue="sex",markers=["o","v"])
```

Out[56]: <seaborn.axisgrid.FacetGrid at 0x1bf5fc77490>



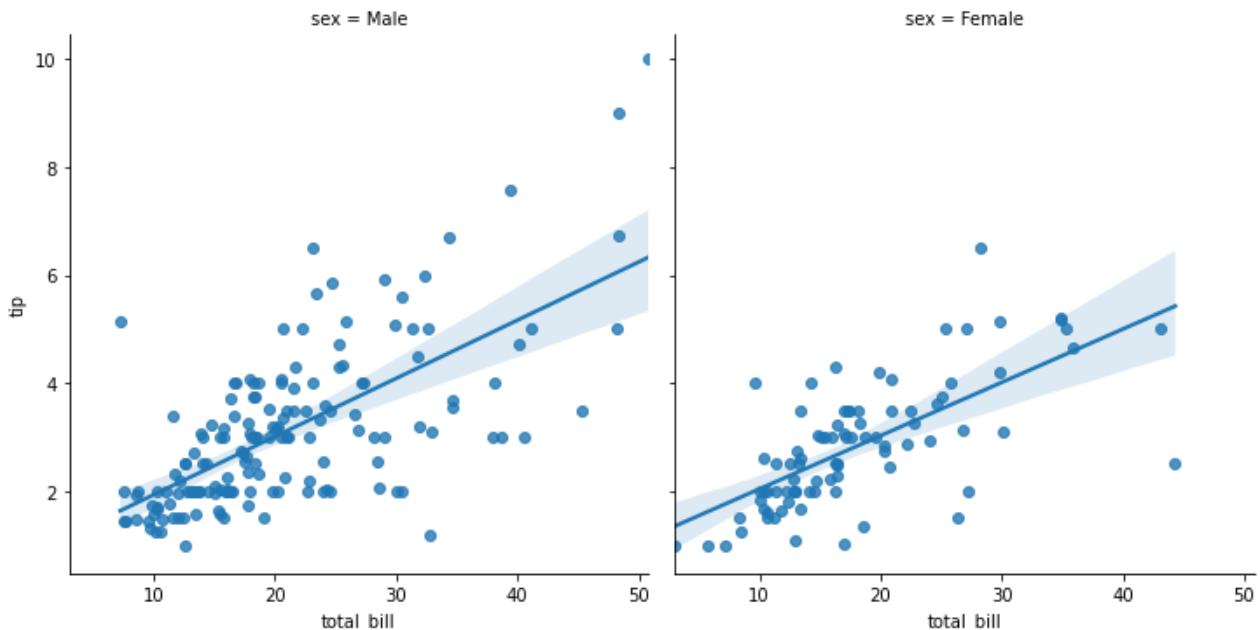
```
In [59]: #scatter_kws is used to increase the size
sns.lmplot(x="total_bill",y="tip",data=tips,hue="sex",markers=["o","v"],
            scatter_kws={"s":100})
```

Out[59]: <seaborn.axisgrid.FacetGrid at 0x1bf61989c40>



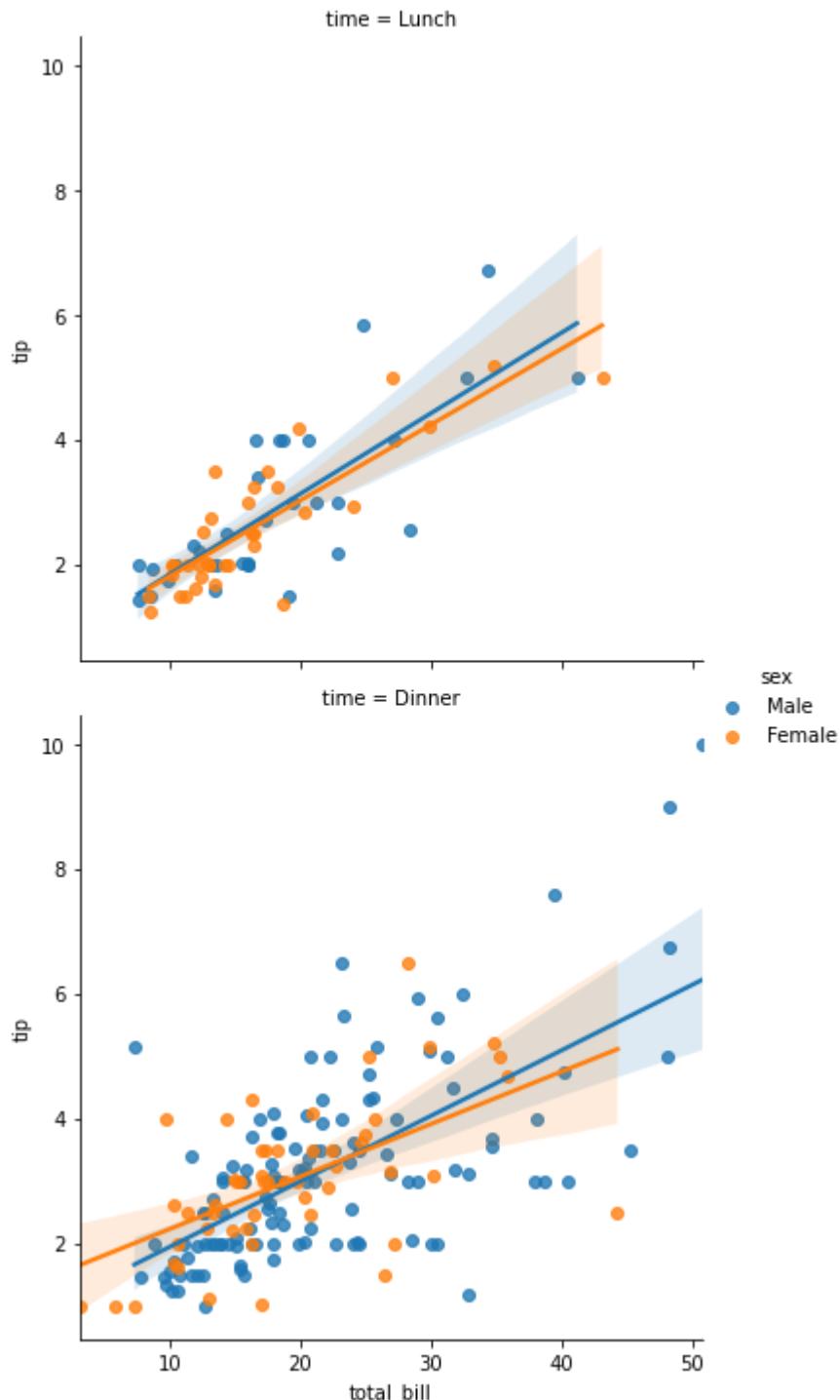
```
In [61]: #Creating a two separated plots with col parameter which is initialized as sex
sns.lmplot(x="total_bill",y="tip",data=tips,col="sex")
```

```
Out[61]: <seaborn.axisgrid.FacetGrid at 0x1bf61ba4250>
```



```
In [63]: #Creating an two row plots according to the gender of male and female
sns.lmplot(x="total_bill",y="tip",data=tips,hue="sex",row="time")
```

```
Out[63]: <seaborn.axisgrid.FacetGrid at 0x1bf6061d100>
```



Seaborn--> Styles and Colours

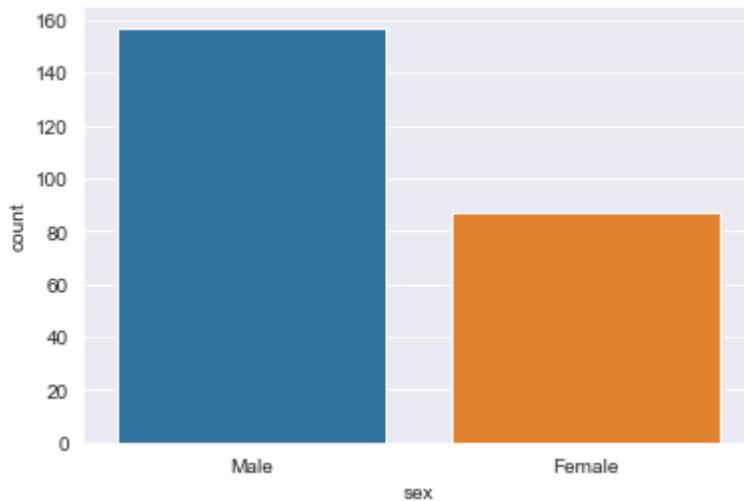
```
In [64]: import seaborn as sns
tips=sns.load_dataset("tips")
tips.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3

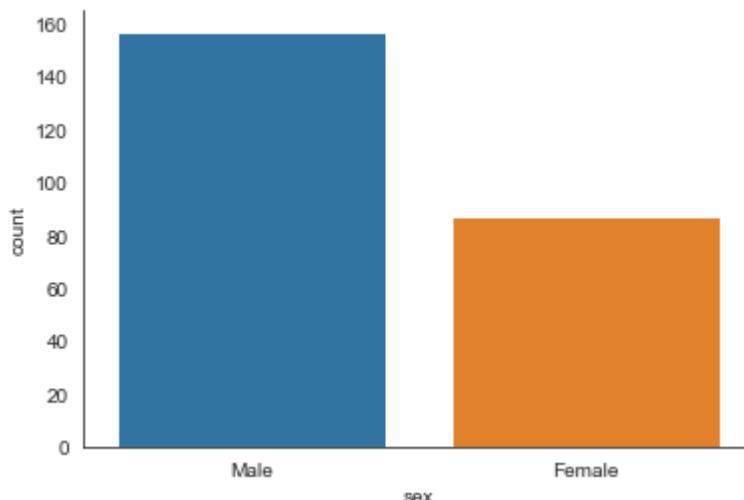
	total_bill	tip	sex	smoker	day	time	size
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
In [68]: #We can also set the style of the background with set_style
sns.set_style("darkgrid")
sns.countplot(x="sex",data=tips)
```

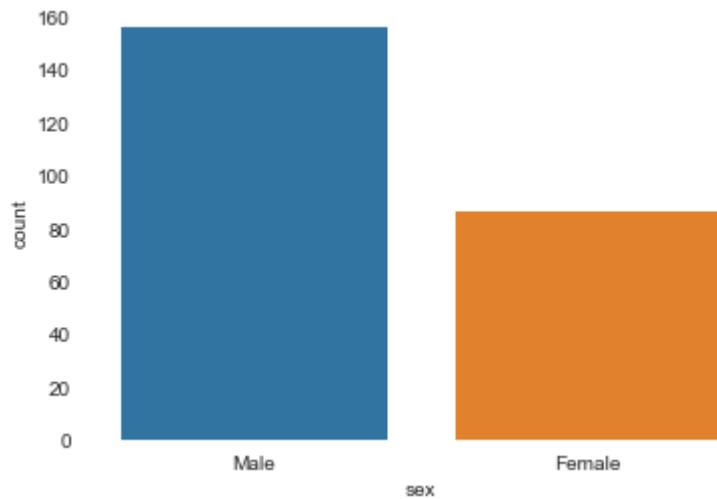
Out[68]: <AxesSubplot:xlabel='sex', ylabel='count'>



```
In [69]: #despine is used to remove top and right corner Line
sns.set_style("white")
sns.countplot(x="sex",data=tips)
sns.despine()
```

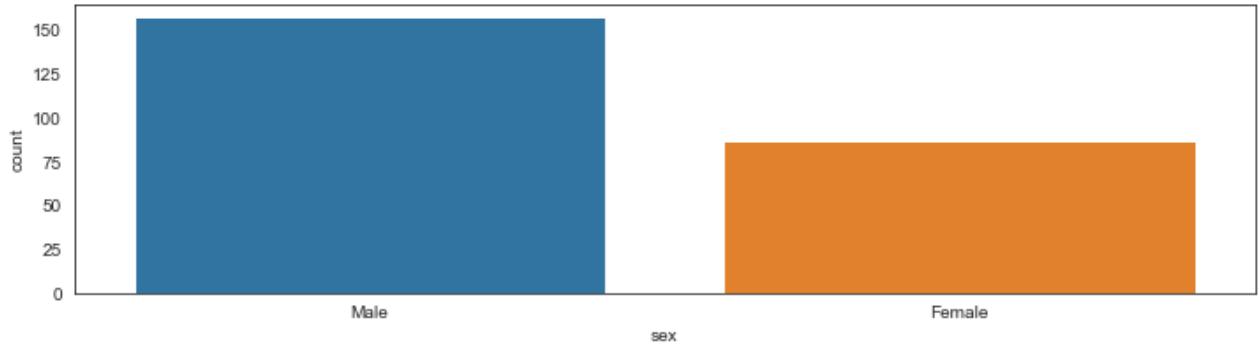


```
In [71]: sns.set_style("white")
sns.countplot(x="sex",data=tips)
sns.despine(left=True,bottom=True)
```



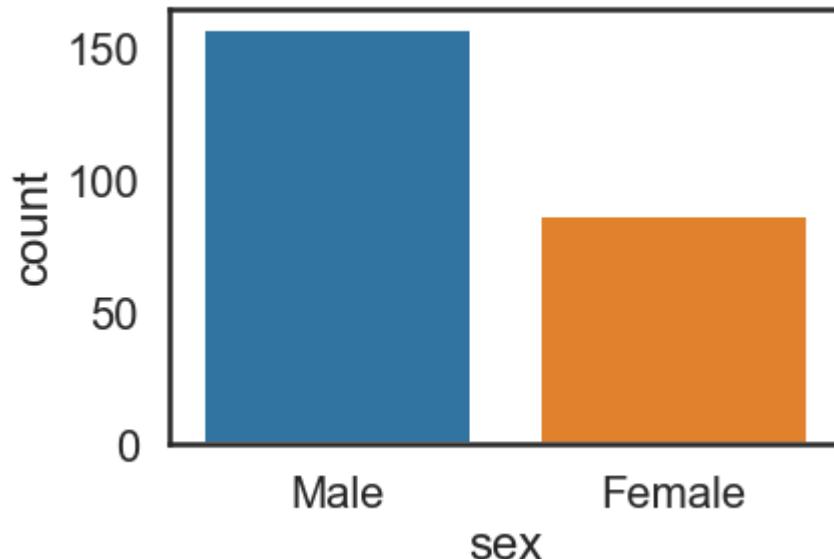
```
In [74]: #It just overrides the size with seaborn  
plt.figure(figsize=(12,3))  
sns.countplot(x="sex",data=tips)
```

```
Out[74]: <AxesSubplot:xlabel='sex', ylabel='count'>
```



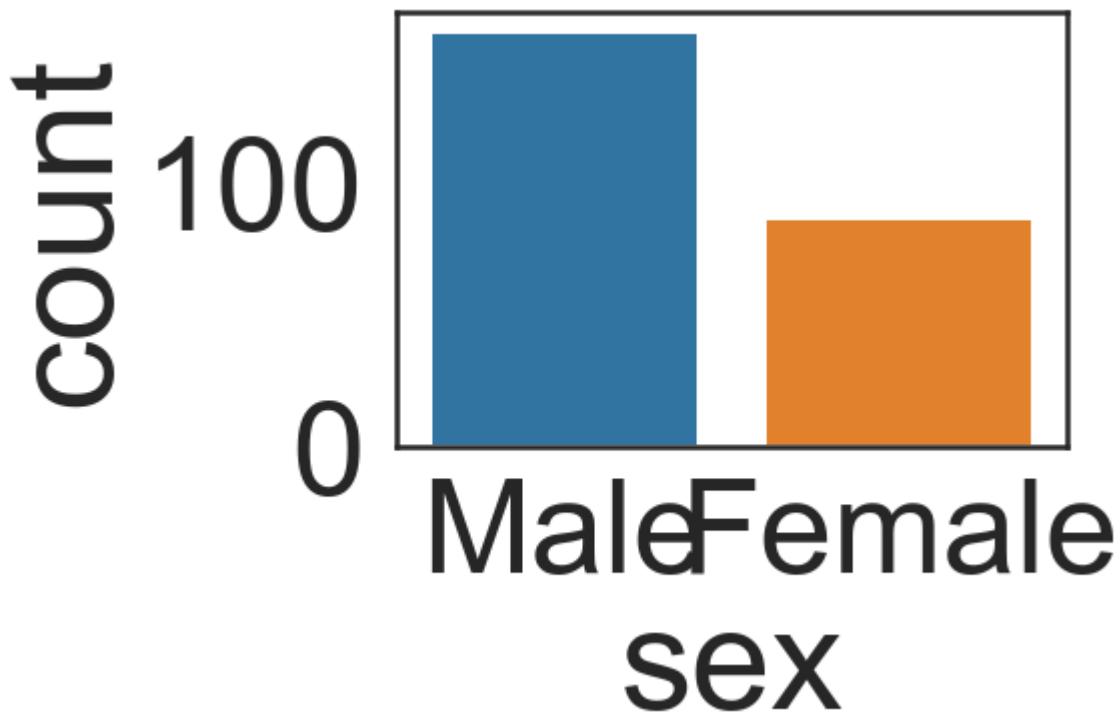
```
In [77]: #This set_context is taken as poster to increase the size like poster  
sns.set_context("poster")  
sns.countplot(x="sex",data=tips)
```

```
Out[77]: <AxesSubplot:xlabel='sex', ylabel='count'>
```



```
In [81]: #Fontscale is used to increase the size
sns.set_context("poster", font_scale=3)
sns.countplot(x="sex", data=tips)
```

```
Out[81]: <AxesSubplot:xlabel='sex', ylabel='count'>
```



PANDAS BUILT IN DATA VISULLIZATION

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

```
In [2]: df1=pd.read_csv("df1",index_col=0)
df1.head()
```

```
Out[2]:
```

	A	B	C	D
2000-01-01	1.339091	-0.163643	-0.646443	1.041233
2000-01-02	-0.774984	0.137034	-0.882716	-2.253382
2000-01-03	-0.921037	-0.482943	-0.417100	0.478638
2000-01-04	-1.738808	-0.072973	0.056517	0.015085
2000-01-05	-0.905980	1.778576	0.381918	0.291436

```
In [3]: df2=pd.read_csv("df2")
```

```
In [4]: df2
```

```
Out[4]:
```

	a	b	c	d
0	0.039762	0.218517	0.103423	0.957904
1	0.937288	0.041567	0.899125	0.977680

	a	b	c	d
2	0.780504	0.008948	0.557808	0.797510
3	0.672717	0.247870	0.264071	0.444358
4	0.053829	0.520124	0.552264	0.190008
5	0.286043	0.593465	0.907307	0.637898
6	0.430436	0.166230	0.469383	0.497701
7	0.312296	0.502823	0.806609	0.850519
8	0.187765	0.997075	0.895955	0.530390
9	0.908162	0.232726	0.414138	0.432007

In [5]: `df3=pd.read_csv("df3")
df3`

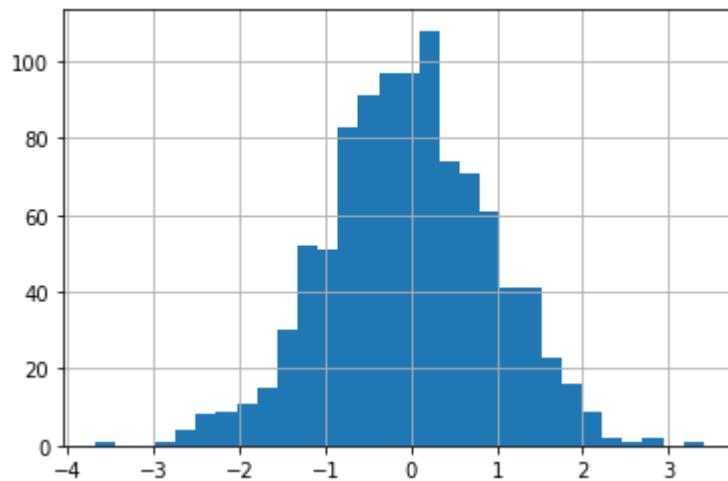
Out[5]:

	a	b	c	d
0	0.336272	0.325011	0.001020	0.401402
1	0.980265	0.831835	0.772288	0.076485
2	0.480387	0.686839	0.000575	0.746758
3	0.502106	0.305142	0.768608	0.654685
4	0.856602	0.171448	0.157971	0.321231
...
495	0.528705	0.226122	0.055835	0.131962
496	0.324730	0.215201	0.935302	0.794115
497	0.118036	0.264574	0.629206	0.824062
498	0.227021	0.660209	0.851353	0.478676
499	0.466157	0.753000	0.115391	0.279712

500 rows × 4 columns

In [10]: `#Creating an histogram
df1["A"].hist(bins=30)`

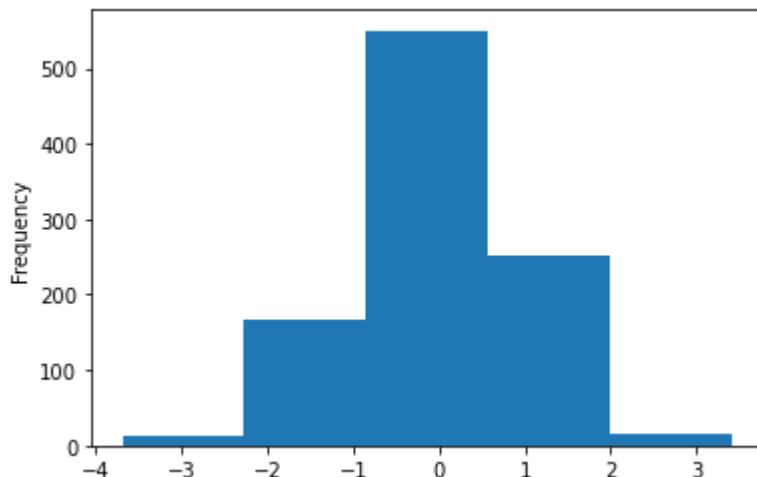
Out[10]: <AxesSubplot:>



```
In [9]: import seaborn as sns
```

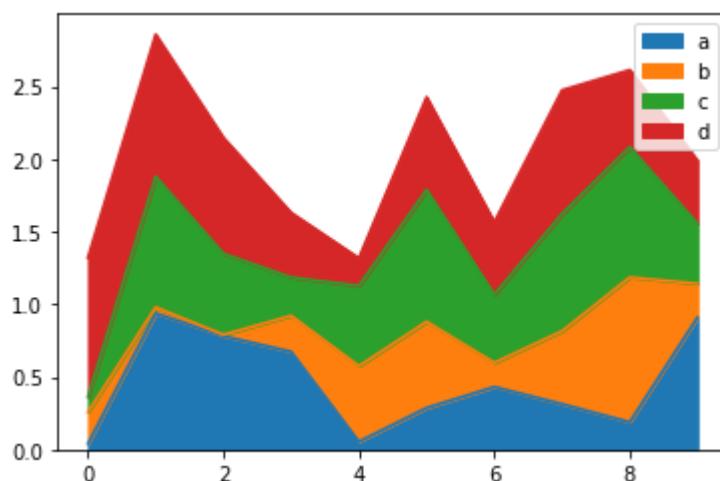
```
In [13]: #Creating an histogram by plot  
#Bins helps to show the plots in zoomed format  
df1["A"].plot(kind="hist",bins=5)
```

```
Out[13]: <AxesSubplot:ylabel='Frequency'>
```



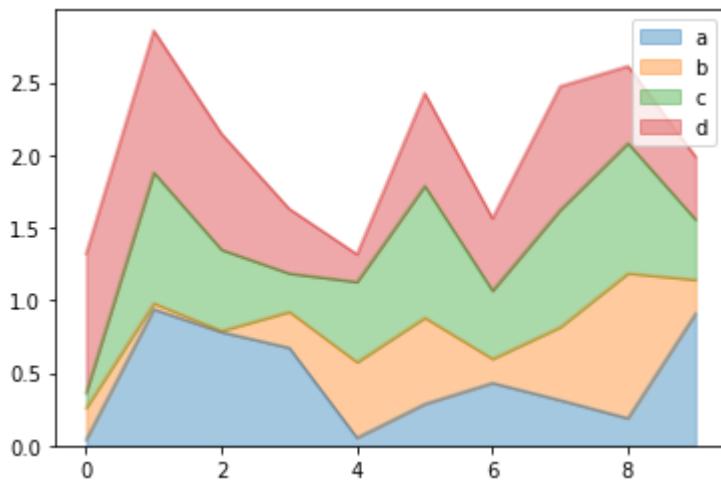
```
In [14]: df2.plot.area()
```

```
Out[14]: <AxesSubplot:>
```



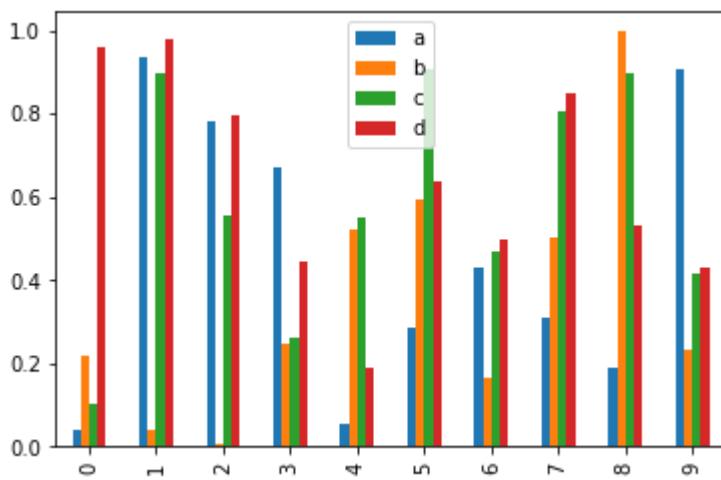
```
In [15]: #Alpha is used for the transparency of the plotted graph  
df2.plot.area(alpha=0.4)
```

Out[15]: <AxesSubplot:>



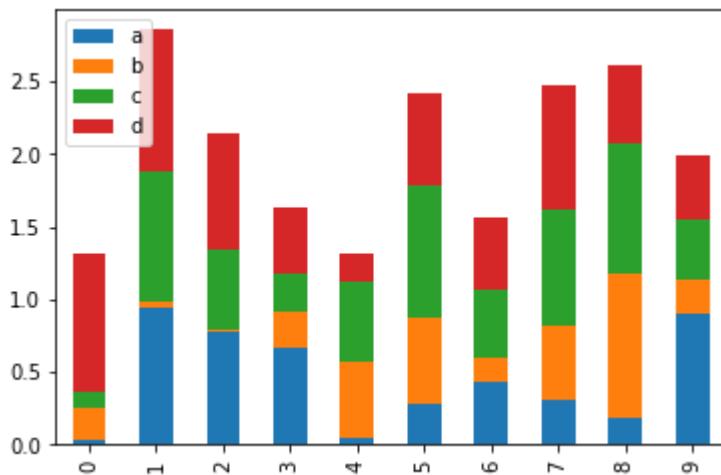
```
In [16]: #Creating an bar plot  
df2.plot.bar()
```

Out[16]: <AxesSubplot:>



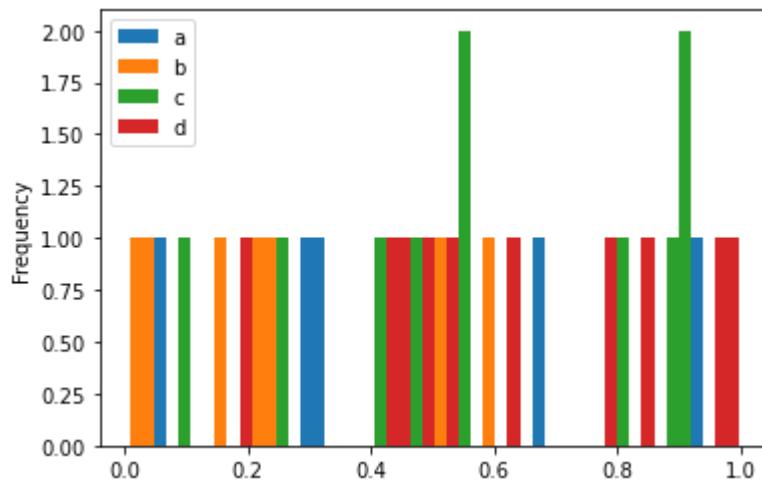
```
In [17]: #Creating an plot in stacked order  
df2.plot.bar(stacked=True)
```

Out[17]: <AxesSubplot:>



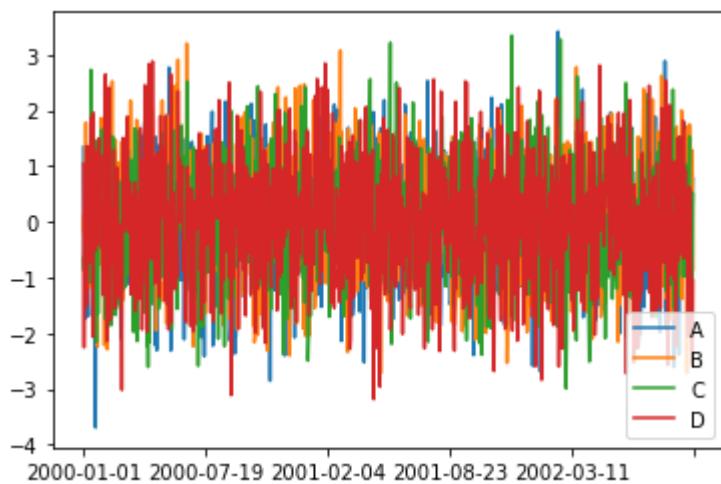
```
In [19]: df2.plot.hist(bins=50)
```

```
Out[19]: <AxesSubplot:ylabel='Frequency'>
```



```
In [21]: df1.plot.line()
```

```
Out[21]: <AxesSubplot:>
```

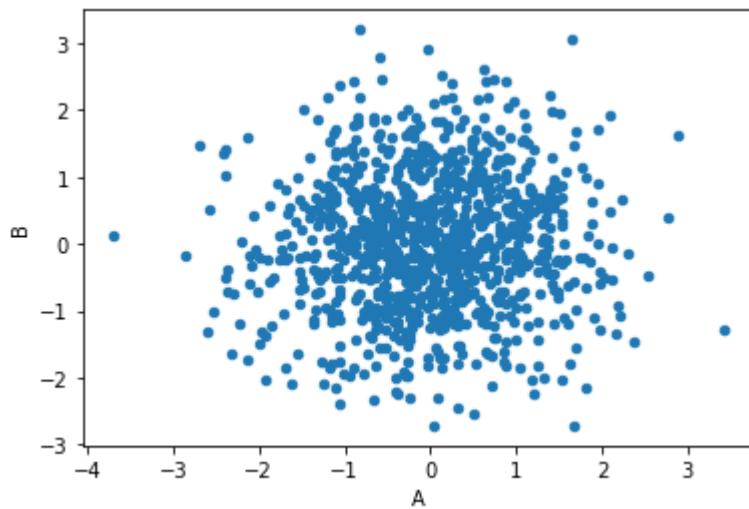


```
In [25]: #Creating an scatter plot
```

```
df1.plot.scatter(x="A",y="B")
```

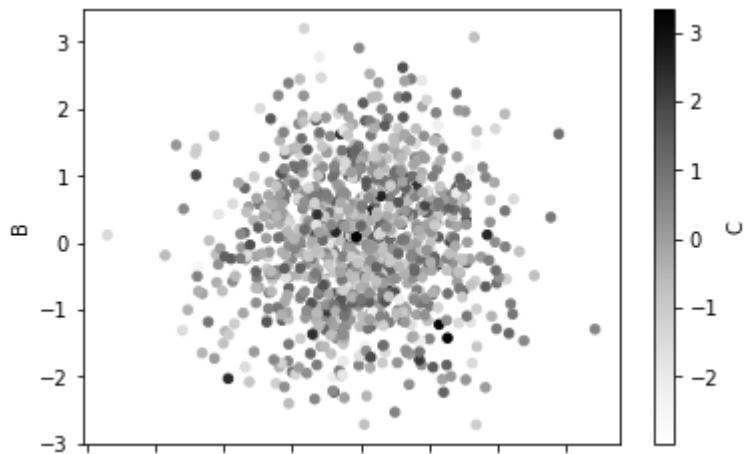
```
<AxesSubplot:xlabel='A', ylabel='B'>
```

Out[25]:



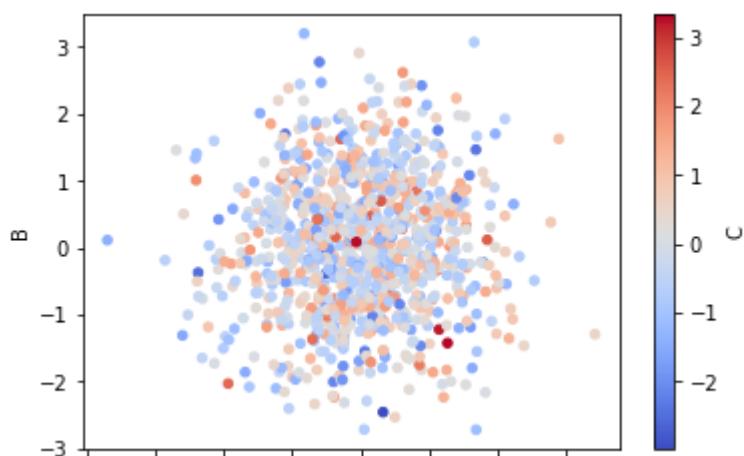
```
In [24]: #Creating an scatter plot with grayscale spaces  
df1.plot.scatter(x="A",y="B",c="C")
```

Out[24]: <AxesSubplot:xlabel='A', ylabel='B'>



```
In [26]: df1.plot.scatter(x="A",y="B",c="C",cmap="coolwarm")
```

Out[26]: <AxesSubplot:xlabel='A', ylabel='B'>

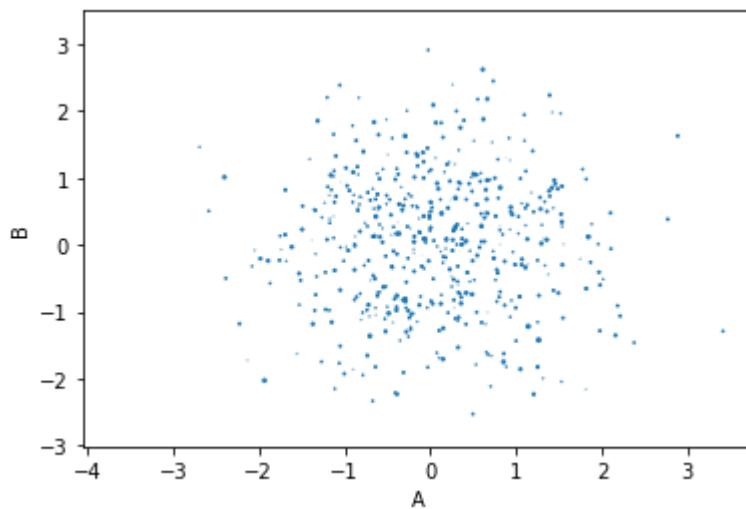


```
In [27]: #Displaying only d plots
```

```
df1.plot.scatter(x="A",y="B",s=df1["C"])
```

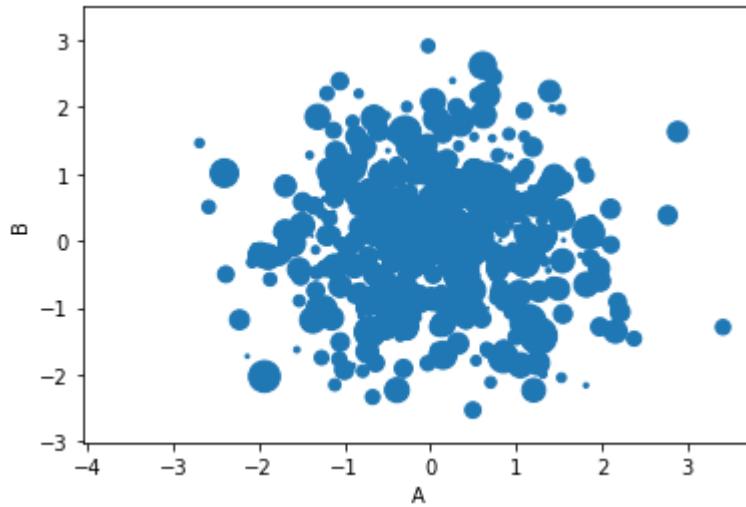
```
C:\Users\srena\anaconda3\lib\site-packages\matplotlib\collections.py:922: RuntimeWarning:  
g: invalid value encountered in sqrt  
    scale = np.sqrt(self._sizes) * dpi / 72.0 * self._factor
```

```
Out[27]: <AxesSubplot:xlabel='A', ylabel='B'>
```



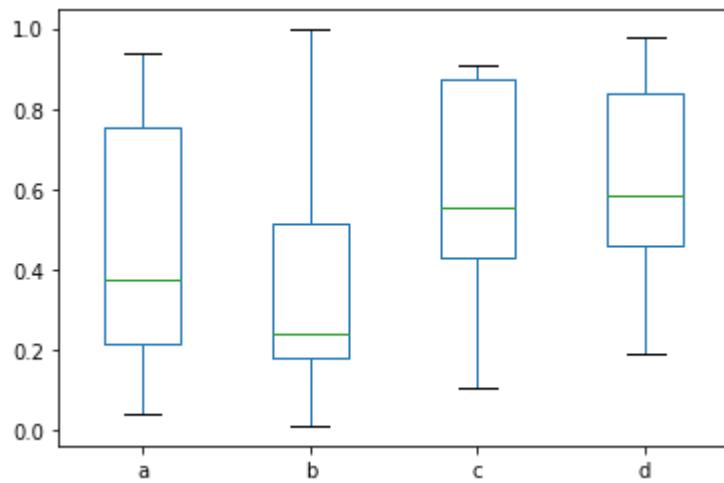
```
In [28]: #We multiply it by 100 to increase the size  
df1.plot.scatter(x="A",y="B",s=df1["C"]*100)
```

```
Out[28]: <AxesSubplot:xlabel='A', ylabel='B'>
```



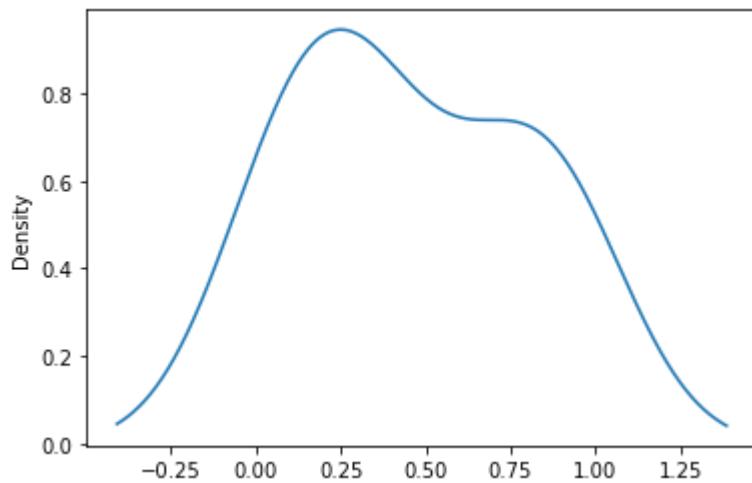
```
In [30]: df2.plot.box()
```

```
Out[30]: <AxesSubplot:>
```



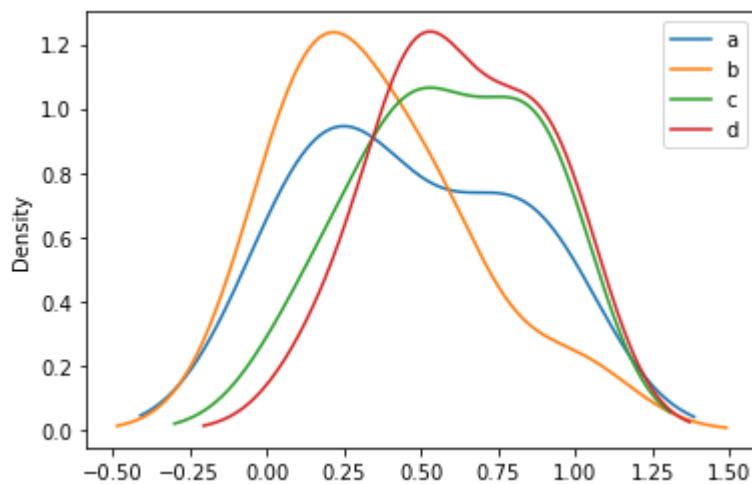
```
In [31]: #Creating an density plot only for a  
df2["a"].plot.density()
```

```
Out[31]: <AxesSubplot:ylabel='Density'>
```



```
In [32]: df2.plot.density()
```

```
Out[32]: <AxesSubplot:ylabel='Density'>
```



Plotly and Cufflinks

-->Plotly is an interactive visualization library -->Cufflinks connects plotly with pandas
 Official Website for plotly: <https://plot.ly/>
 Official Website for cufflinks: <https://github.com/santosjorge/cufflinks>

```
In [3]: import pandas as pd
import numpy as np
from plotly import __version__
%matplotlib inline
```

```
In [4]: #To view the version of -plotly
print(__version__)
```

4.14.3

```
In [5]: #Importing cufflinks module as cf
import cufflinks as cf
```

```
In [6]: #We are importing plotly offline which helps to view the data visualization in offline mode
#Importing all the required functions from the module plotly in offline mode
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
```

```
In [7]: #WE need to make it connected because plotly default will be disconnected
#This helps to connect the data visualization plots
init_notebook_mode(connected=True)
```

```
In [8]: #This go.offline() helps to work in offline mode
cf.go_offline()
```

```
In [9]: #Creating the data
df=pd.DataFrame(np.random.randn(100,4),columns="A B C D".split())
```

```
In [10]: df
```

	A	B	C	D
0	-1.381168	-0.690346	-1.737674	-0.332257
1	0.308550	-0.691469	-0.317033	0.558965
2	0.291141	-0.570416	1.778396	0.116234
3	0.992396	0.257372	0.676751	-0.340807
4	0.894680	0.992369	-1.719878	-0.137530
...
95	0.861656	-0.050806	0.255882	-1.388298
96	-0.314507	1.054642	-0.780965	0.764544
97	-0.436698	0.968113	1.714676	-1.030148
98	-0.454062	1.302098	-0.061179	1.174992
99	0.289068	-0.446791	-0.564385	-0.181287

100 rows × 4 columns

In [11]: `df.head()`

Out[11]:

	A	B	C	D
0	-1.381168	-0.690346	-1.737674	-0.332257
1	0.308550	-0.691469	-0.317033	0.558965
2	0.291141	-0.570416	1.778396	0.116234
3	0.992396	0.257372	0.676751	-0.340807
4	0.894680	0.992369	-1.719878	-0.137530

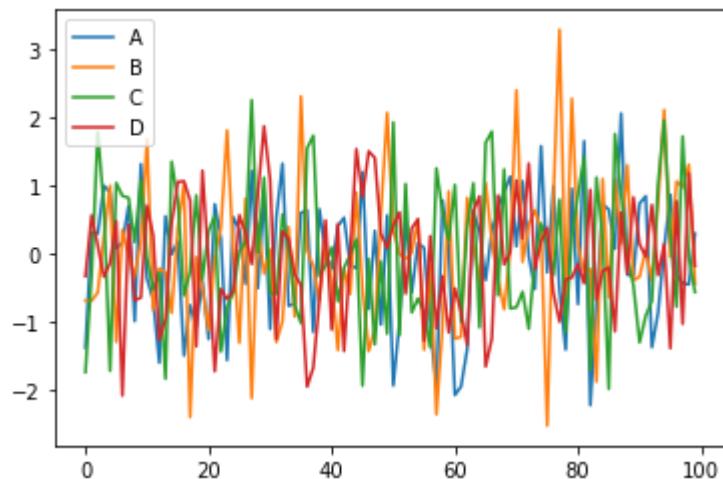
In [12]: `df2=pd.DataFrame({"Category":["A","B","C"],"Values":[32,43,50]})`In [13]: `df2`

Out[13]:

	Category	Values
0	A	32
1	B	43
2	C	50

In [14]: `df.plot()`

Out[14]: <AxesSubplot:>





Introduction to Machine Learning



Companion Book

We will be using **Introduction to Statistical Learning** by Gareth James as a companion book.

It's freely available online, let's see how to get it



Companion Book

Students who want the mathematical theory
should do the reading.

Students who just want light theory and more
interested in Python Applications.



Companion Book

Read Chapters 1 & 2 to gain a background understanding before continuing to the Machine Learning Lectures.



What is Machine Learning?

- Machine learning is a method of data analysis that automates analytical model building.
- Using algorithms that iteratively learn from data, machine learning allows computers to find hidden insights without being explicitly programmed where to look.

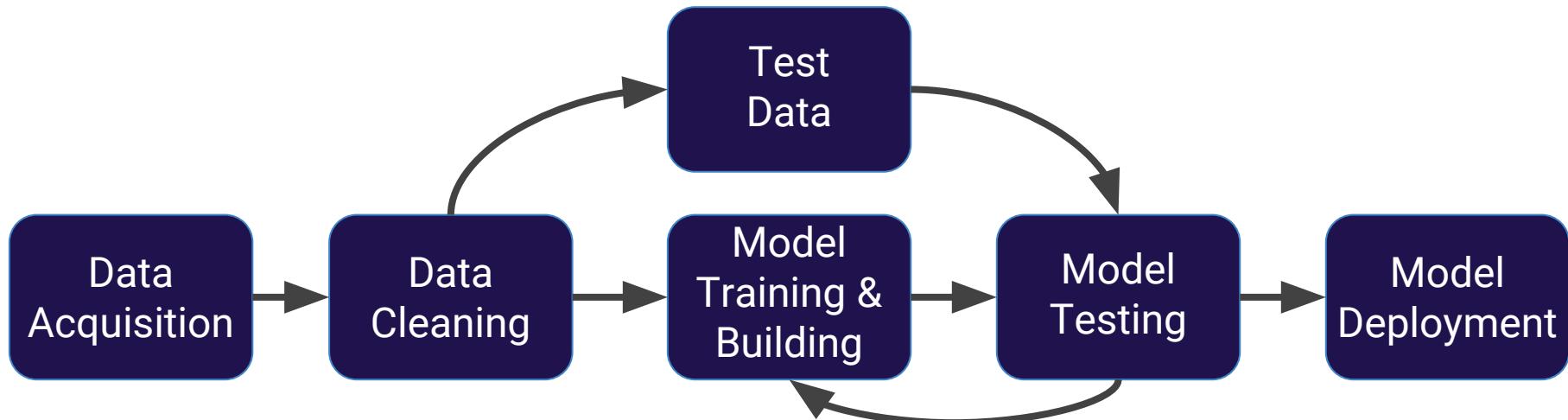


What is it used for?

- Fraud detection.
- Web search results.
- Real-time ads on web pages
- Credit scoring and next-best offers.
- Prediction of equipment failures.
- New pricing models.
- Network intrusion detection.
- Recommendation Engines
- Customer Segmentation
- Text Sentiment Analysis
- Predicting Customer Churn
- Pattern and image recognition.
- Email spam filtering.
- Financial Modeling



Machine Learning Process





Supervised Learning

- **Supervised learning** algorithms are trained using **labeled** examples, such as an input where the desired output is known.
- For example, a piece of equipment could have data points labeled either “F” (failed) or “R” (runs).



Supervised Learning

- The learning algorithm receives a set of inputs along with the corresponding correct outputs, and the algorithm learns by comparing its actual output with correct outputs to find errors.
- It then modifies the model accordingly.



Supervised Learning

- Through methods like classification, regression, prediction and gradient boosting, supervised learning uses patterns to predict the values of the label on additional unlabeled data.
- Supervised learning is commonly used in applications where historical data predicts likely future events.



Supervised Learning

- For example, it can anticipate when credit card transactions are likely to be fraudulent or which insurance customer is likely to file a claim.
- Or it can attempt to predict the price of a house based on different features for houses for which we have historical price data.



Unsupervised Learning

- **Unsupervised learning** is used against data that has no historical labels.
- The system is not told the "right answer." The algorithm must figure out what is being shown.
- The goal is to explore the data and find some structure within.



Unsupervised Learning

- Or it can find the main attributes that separate customer segments from each other.
- Popular techniques include self-organizing maps, nearest-neighbor mapping, k-means clustering and singular value decomposition.



Unsupervised Learning

- These algorithms are also used to segment text topics, recommend items and identify data outliers.



Reinforcement Learning

- **Reinforcement learning** is often used for robotics, gaming and navigation.
- With reinforcement learning, the algorithm discovers through trial and error which actions yield the greatest rewards.



Reinforcement Learning

- This type of learning has three primary components: the agent (the learner or decision maker), the environment (everything the agent interacts with) and actions (what the agent can do).



Reinforcement Learning

- The objective is for the agent to choose actions that maximize the expected reward over a given amount of time.
- The agent will reach the goal much faster by following a good policy.



Reinforcement Learning

- So the goal in reinforcement learning is to learn the best policy.



Machine Learning in this Course

- For each algorithm or Machine Learning topic:
 - Reading Assignment
 - Light Overview of Theory
 - Demonstration Lecture with Python
 - Machine Learning Project Assignment
 - Overview of Solution for Project
- Let's get a brief tour of the Machine Learning notes!



Reinforcement Learning

- Machine Learning takes time to learn.
- Be patient with yourself and feel free to post to the QA forums.
- No one course can be a reference for all Machine Learning topics, but I'm always happy to point you in the right direction!



Let's start using Python for Machine Learning!



Introduction to Linear Regression



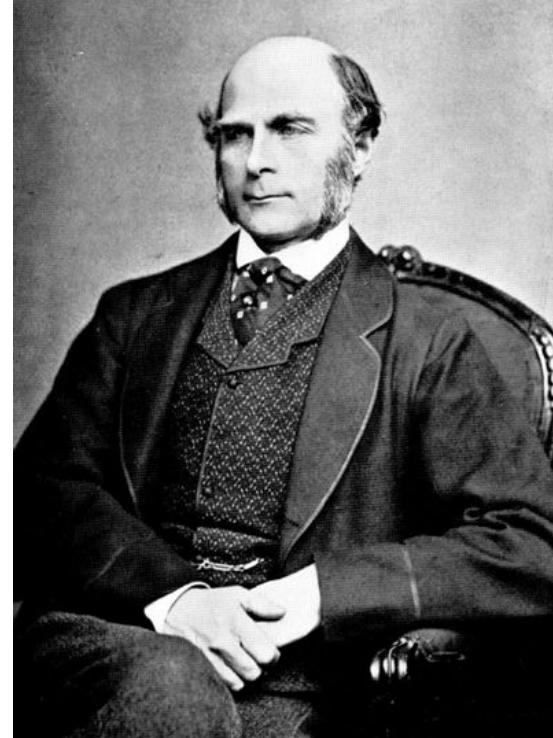
Reading Assignment

Chapters 2 & 3 of
Introduction to Statistical Learning
By Gareth James, et al.



History

This all started in the 1800s with a guy named [Francis Galton](#). Galton was studying the relationship between parents and their children. In particular, he investigated the relationship between the heights of fathers and their sons.

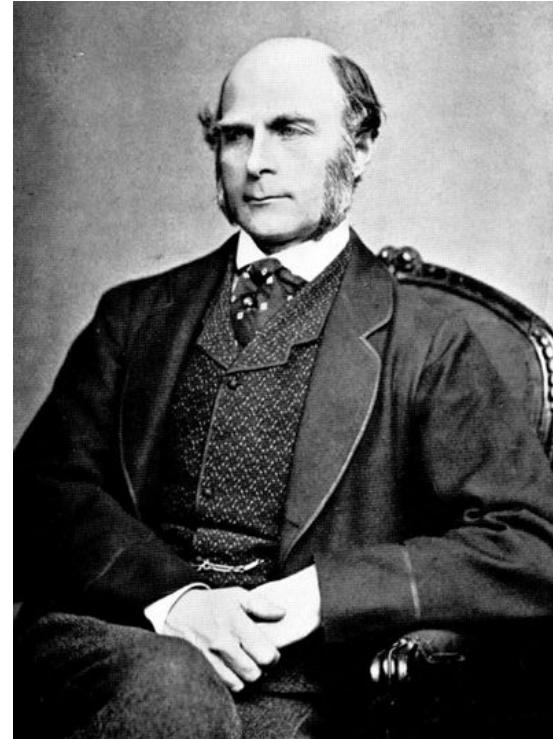




History

What he discovered was that a man's son tended to be roughly as tall as his father.

However Galton's breakthrough was that the son's height **tended to be closer to the overall average** height of all people.

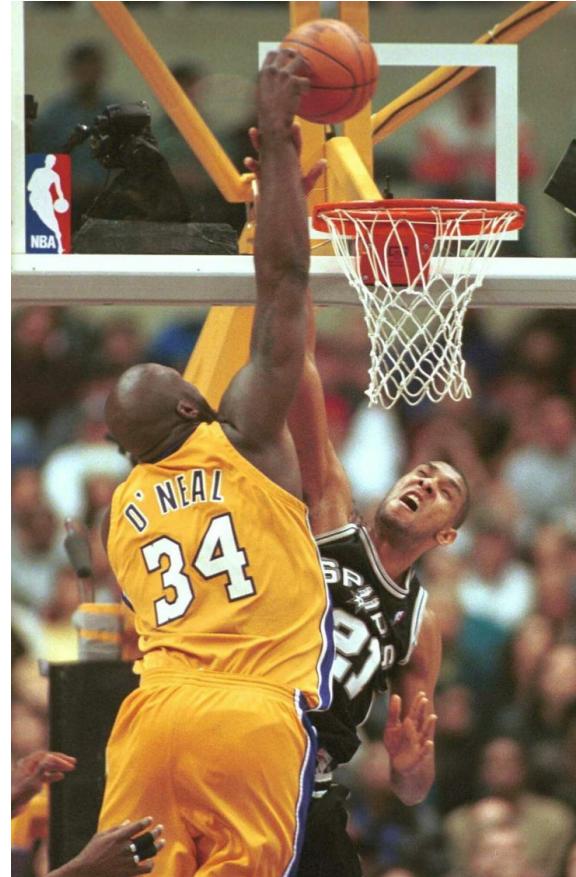




Example

Let's take Shaquille O'Neal as an example. Shaq is really tall: 7ft 1in (2.2 meters).

If Shaq has a son, chances are he'll be pretty tall too. However, Shaq is such an anomaly that there is also a very good chance that his son will be **not be as tall as Shaq**.

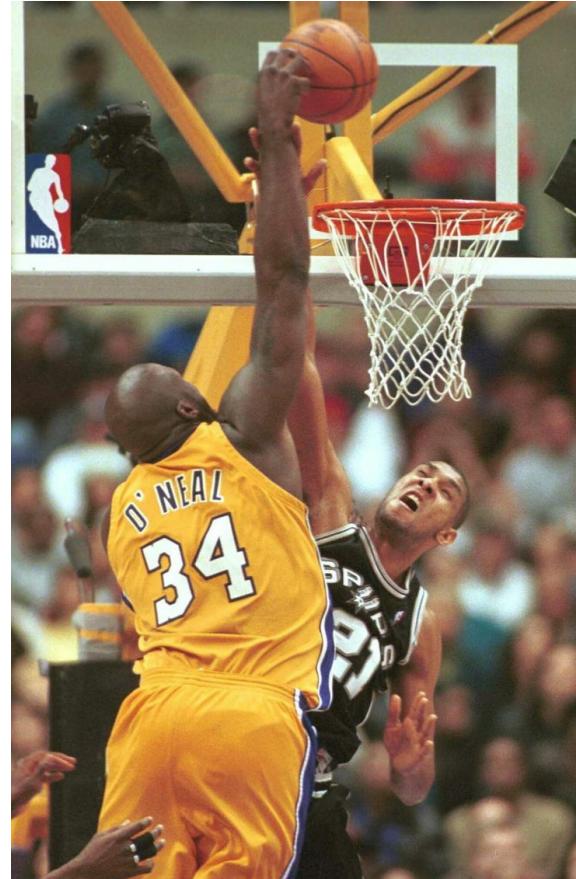




Example

Turns out this is the case:
Shaq's son is pretty tall (6 ft 7 in), but not nearly as tall as his dad.

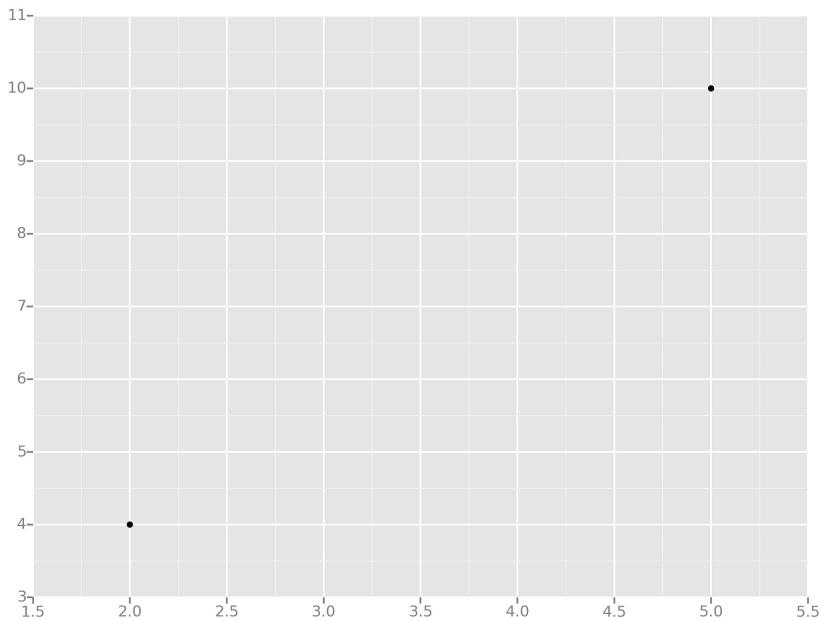
Galton called this phenomenon **regression**, as in "A father's son's height tends to regress (or drift towards) the mean (average) height."





Example

Let's take the simplest possible example:
calculating a regression
with only 2 data points.

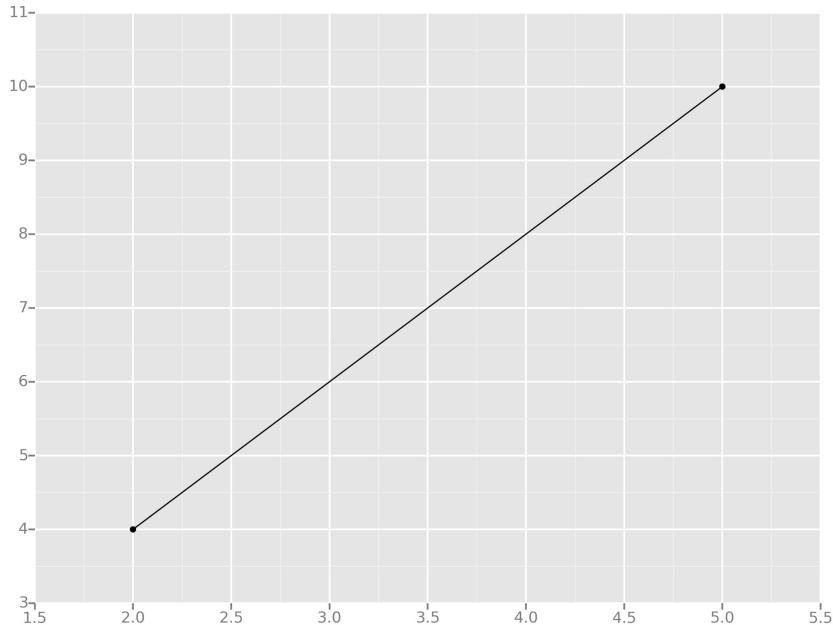




Example

All we're trying to do when we calculate our regression line is draw a line that's as close to every dot as possible.

For classic linear regression, or "Least Squares Method", you only measure the closeness in the "up and down" direction

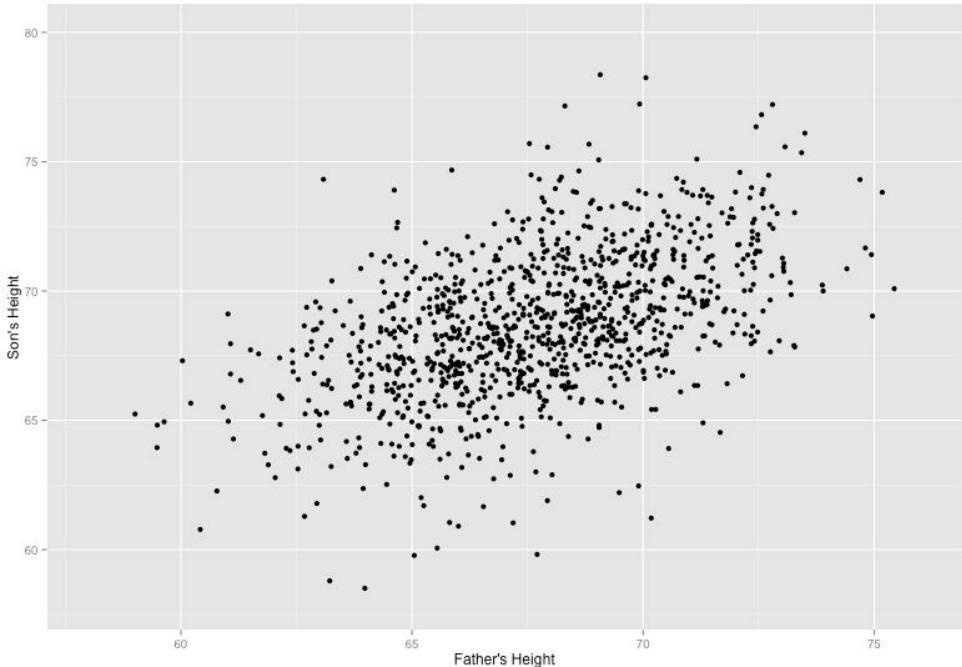




Example

Now wouldn't it be great if we could apply this same concept to a graph with more than just two data points?

By doing this, we could take multiple men and their son's heights and do things like tell a man how tall we expect his son to be...before he even has a son!

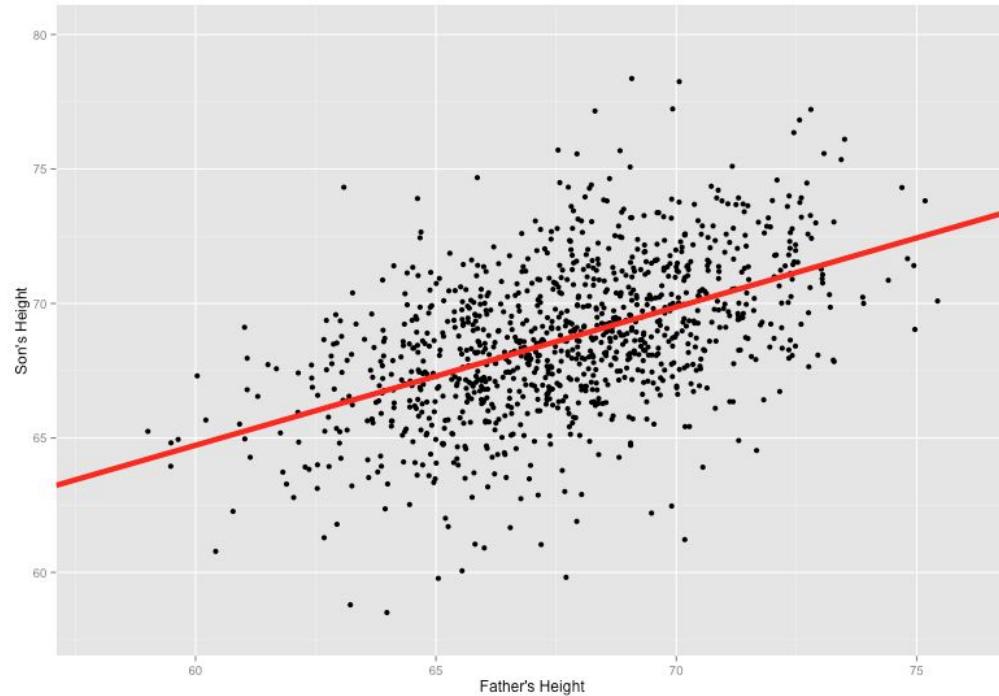




Example

Our goal with linear regression is to **minimize the vertical distance** between all the data points and our line.

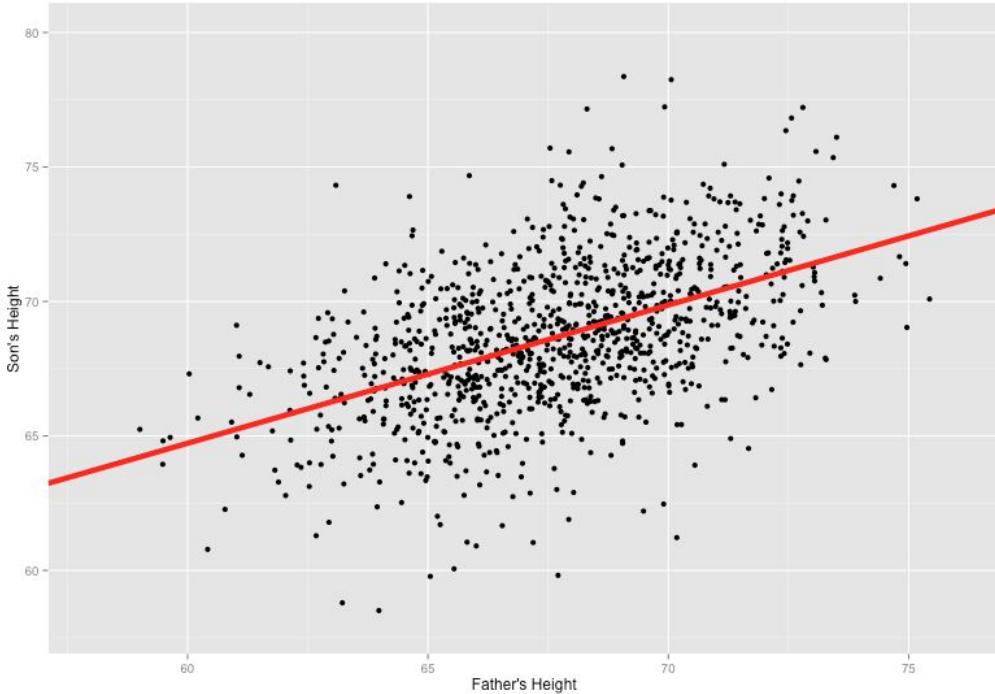
So in determining the **best line**, we are attempting to minimize the distance between **all** the points and their distance to our line.





Example

There are lots of different ways to minimize this, (sum of squared errors, sum of absolute errors, etc), but all these methods have a general goal of minimizing this distance.

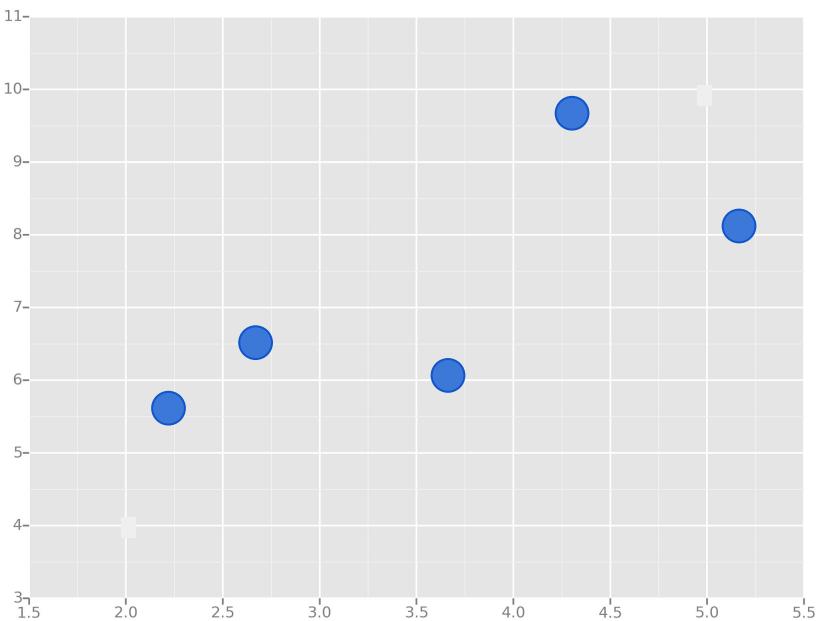




Example

For example, one of the most popular methods is the least squares method.

Here we have blue data points along an x and y axis.

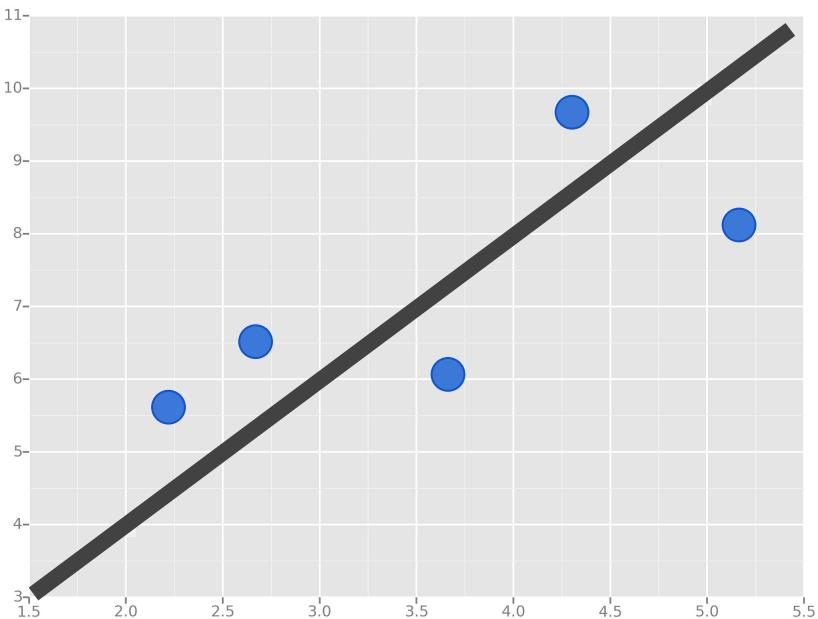




Example

Now we want to fit a linear regression line.

The question is, how do we decide which line is the best fitting one?

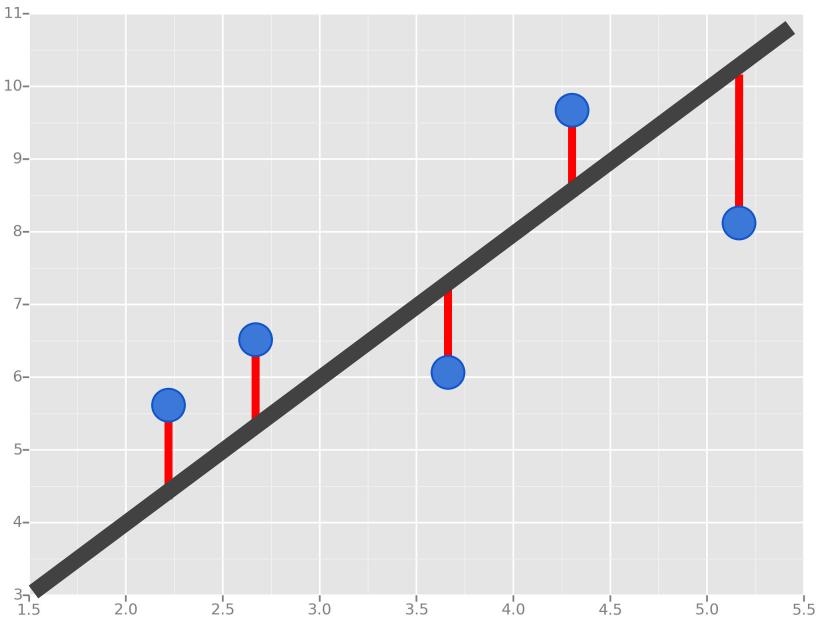




Example

We'll use the Least Squares Method, which is fitted by minimizing the ***sum of squares of the residuals***.

The residuals for an observation is the difference between the observation (the y-value) and the fitted line.





Example with Python

In the next lecture we'll use SciKit-Learn and Python to create a linear regression model.

Then you'll have your own portfolio project exercise and afterwards we'll go over the solutions to that project.



Example with Python

See you in the next lecture!

LINEAR REGRESSION

```
In [2]: #Importing all the required packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
In [3]: #Reading the dataset
df=pd.read_csv("USA_Housing.csv")
```

```
In [4]: #Displaying the dataset
df
```

Out[4]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386
...
4995	60567.944140	7.830362	6.137356	3.46	22837.361035	1.060194e+06	USNS Williams\nFPO AP 30153-7653
4996	78491.275435	6.999135	6.576763	4.02	25616.115489	1.482618e+06	PSC 9258, Box 8489\nAPO AA 42991- 3352
4997	63390.686886	7.250591	4.805081	2.13	33266.145490	1.030730e+06	4215 Tracy Garden Suite 076\nJoshualand, VA 01...
4998	68001.331235	5.534388	7.130144	5.44	42625.620156	1.198657e+06	USS Wallace\nFPO AE 73316
4999	65510.581804	5.992305	6.792336	4.07	46501.283803	1.298950e+06	37778 George Ridges Apt. 509\nEast Holly, NV 2...

5000 rows × 7 columns

In [5]: #Reading first five dataset with head function
df.head()

Out[5]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

In [6]: #Reading the info details
df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   Avg. Area Income    5000 non-null   float64
 1   Avg. Area House Age 5000 non-null   float64
 2   Avg. Area Number of Rooms 5000 non-null   float64
 3   Avg. Area Number of Bedrooms 5000 non-null   float64
 4   Area Population     5000 non-null   float64
 5   Price              5000 non-null   float64
 6   Address            5000 non-null   object  
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

In [7]: #Reading the basic details like mean min max values of the house
df.describe()

Out[7]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

In [9]: `#displaying the columns
df.columns`

Out[9]: `Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
dtype='object')`

Testing and Training the Data

Here X is taking all the Features columns and y is taking the prediction columns

In []: The Predicted column should be taken as y rest all columns should be taken as X

--> The column for example price should be taken as y rest all the columns should be taken as X --> As we are predicting the price in this situation

In [11]: `#Taking the X as feature of the dataset columns for the prediction
#Here we use X as variable to store the feature columns
X=df[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms','Avg. Area`

In [12]: `#Taking the y as target in which we are doing the prediction
#Here we are predicting the price of the house after one year with all x variables sets
y=df["Price"]`

In [14]: `#We are importing train_test_split from the model_selection in sklearn
from sklearn.model_selection import train_test_split`

In [15]: `# Here we are splitting the data into 4 variables X_train,X_test,y_train,y_test with the
#We need to put train_size as 0.4 and random_state as 101
X_train,X_test,y_train,y_test=train_test_split(X,y,train_size=0.4,random_state=101)`

In []:

```
X_train--> Predicted Features values(Eg:Avg.Area.Income,Avg.Area.House.Age etc)
X_test--> Original Features values
Y_train--> Predicted Price
Y_test--> Original Price
```

In [16]: `#Displaying the X_train
X_train`

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population
1672	49775.405947	5.305108	6.178535	2.24	23557.361654
4557	75571.044023	6.114928	7.318214	4.44	33988.435859

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population
1040	66250.316685	4.858520	7.758692	3.08	24201.753077
514	58978.222191	6.658346	7.889948	3.01	44071.604628
2148	73053.953218	5.712999	6.673142	4.30	41686.518927
...
4171	56610.642563	4.846832	7.558137	3.29	25494.740298
599	70596.850945	6.548274	6.539986	3.10	51614.830136
1361	55621.899104	3.735942	6.868291	2.30	63184.613147
1547	63044.460096	5.935261	5.913454	4.10	32725.279544
4959	75078.791516	7.644779	8.440726	4.33	56148.449322

2000 rows × 5 columns

```
In [17]: #Displaying the y_train
y_train
```

```
Out[17]: 1672    4.540557e+05
4557    1.218264e+06
1040    1.094322e+06
514     1.308984e+06
2148    1.220724e+06
...
4171    7.296417e+05
599     1.599479e+06
1361    1.102641e+06
1547    8.650995e+05
4959    2.108376e+06
Name: Price, Length: 2000, dtype: float64
```

Creating an Linear Regression Model

```
In [19]: # Importing LinearRegression Model from Sklearn for creating an Model
from sklearn.linear_model import LinearRegression
```

```
In [20]: #This is the line to create the model
#We make a model named lm
lm=LinearRegression()
```

```
In [21]: #Fitting the trained data(X_train and y_train) into the Model lm
lm.fit(X_train,y_train)
```

```
Out[21]: LinearRegression()
```

```
In [23]: #Finding the intercept_ to find the common predicted value
lm.intercept_
```

```
Out[23]: -2616998.0228818767
```

```
In [24]: #Coefficient meaning:a numerical or constant quantity placed before and multiplying the
#in an algebraic expression (e.g. 4 in 4x y).
#i,e To predict the the price in multiple times
#For Example in 2020 the price of the house is 1 lakh after 2 years the price of the ho
#3 times the money so the 3 is the coefficient here
lm.coef_
```

```
Out[24]: array([2.13844438e+01, 1.62975483e+05, 1.21802121e+05, 1.93416306e+03,
   1.51896067e+01])
```

```
In [31]: #Displaying the coefficient in dataframe format
cdf=pd.DataFrame(lm.coef_,X.columns,columns=["Coefficient"])
```

```
In [32]: #Here after an year the Avg.Area.Income is increased 21 times of the price
cdf
```

	Coefficient
Avg. Area Income	21.384444
Avg. Area House Age	162975.483408
Avg. Area Number of Rooms	121802.121132
Avg. Area Number of Bedrooms	1934.163056
Area Population	15.189607

Predictions

```
In [33]: #We are predicting the price
#In that first step we are predicting the feature columns which is X_test taking as pre
#Lm is trained with model of predicted price
predictions=lm.predict(X_test)
```

```
In [34]: #Original values of the price
y_test
```

```
Out[34]: 1718    1.251689e+06
2511    8.730483e+05
345     1.696978e+06
2521    1.063964e+06
54      9.487883e+05
...
4829    1.500482e+06
2010    1.987332e+06
2460    1.536235e+06
2945    1.612102e+06
641     1.175869e+06
Name: Price, Length: 3000, dtype: float64
```

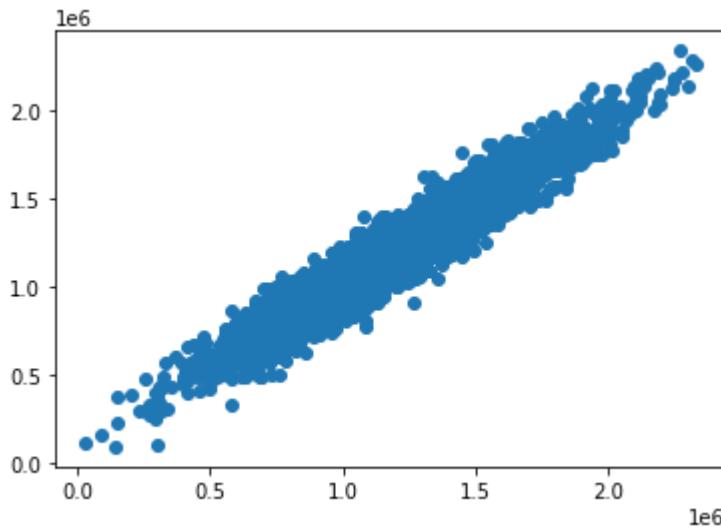
```
In [37]: #Predictied values
predictions
```

```
Out[37]: array([1259341.72755371, 827356.2861316 , 1738047.78105784, ...,
   1429750.59319652, 1410407.93111557, 1194408.00899732])
```

```
In [38]: #We are plotting the scatter plot for the accuracy of the prediction
#Here when the plots are nearby then the prediction is correct
```

```
#Eg here in the figure all the plots are near  
#If the plots are in separate the prediction is little not accuracy  
plt.scatter(y_test,predictions)
```

Out[38]: <matplotlib.collections.PathCollection at 0x22153bb9b20>



In [39]:

```
#Importing metrics from sklearn  
from sklearn import metrics
```

In [40]:

```
#Calculating the mean absolute error  
metrics.mean_absolute_error(y_test,predictions)
```

Out[40]: 82336.52594520213

In [43]:

```
metrics.mean_squared_error(y_test,predictions)
```

Out[43]: 10354334181.172688

In [44]:

```
np.sqrt(metrics.mean_squared_error(y_test,predictions))
```

Out[44]: 101756.2488556486

LINEAR REGRESSION COMPLETED

In []:



Model Evaluation



Companion Book

Model Evaluation is a fundamental topic of understanding your model's performance!

Review Chapter 2 of **Introduction to Statistical Learning** for a more in depth look!



Model Evaluation

- We'll discuss the following topics
 - Train Test Splits
 - Holdout Sets
 - Parameter Grids
 - Scala and Spark for Model Evaluation
 - Bias Variance Trade-Off
 - Documentation Exploration
 - Code through some Examples



Train Test Splits

- We've previously talked about Train Test Splits, but let's review the concept.
- You will always train a Machine Learning Algorithm on some data, but afterwards you will want some measure of how well it performed.
- Each main Machine Learning Task has different metrics for evaluation



Train Test Splits

- Regression
 - R²
 - RMSE
- Classification
 - Precision
 - Recall
- Clustering
 - Within Sum of Squares Error



Train Test Splits

- While you could get these measurements using the same data you trained your model on, that is not a good idea.
- Your model has already seen this data meaning it is not a good choice for evaluating your model's performance
- You should get these metrics off test data, which your model has not seen yet.
- This is known as a train-test split.



Holdout Data

- An expansion of this idea is the holdout data set.
- This is separate from the training and test sets.
- In this process you use the training data to fit your model, you use the test set to evaluate and adjust your model.
- You can use the test set over and over again.
- Finally, before deploying your model, you check it against the holdout to get some final metrics on performance.



Parameter Grids

- As we've seen, we can add optional parameters to Machine Learning Algorithms.
- Many times it is difficult to know what are good values for these parameters.
- Spark makes it possible to set up a grid of parameters to train across.
- You create multiple models, train them across the grid, and Spark reports back which model performed best.



Spark and Scala for Model Evaluation

- Spark makes all of these processes generally easy with the use of 3 object types:
 - Evaluators
 - ParamGridBuilders
 - TrainValidationSplit
- Later on in this section we will explore how to use these object types to implement the ideas discussed here.



Spark and Scala for Model Evaluation

- An important aspect to understanding all of this is the Bias-Variance Trade-Off.
- We won't directly explore this with Spark and Scala because it pertains more to theory than Data Engineering.
- However let's take the time now to at least understand the concept so we can have full context for this section of the course.



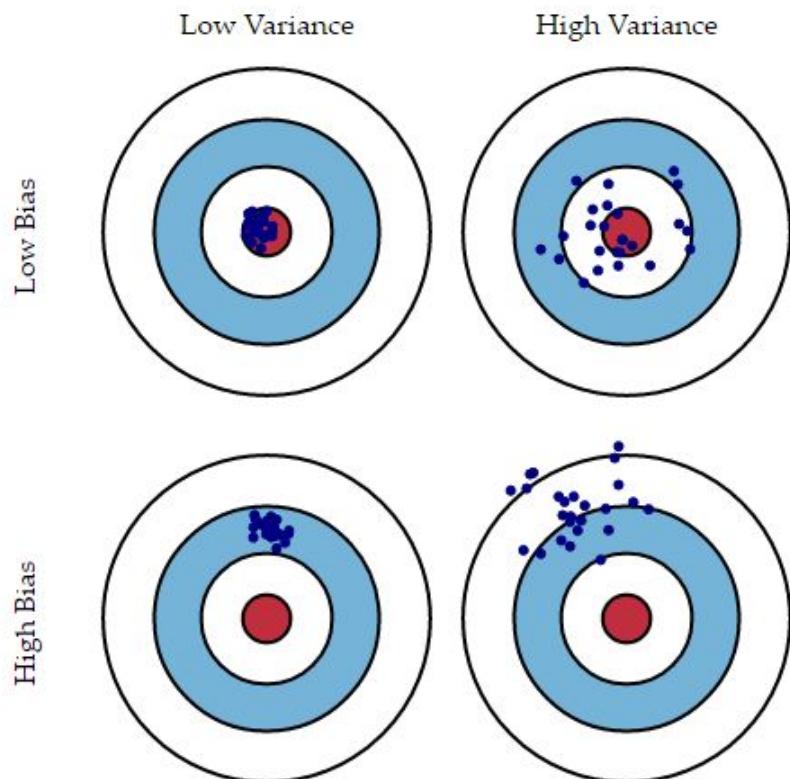
Bias Variance Trade-Off

- The bias-variance trade-off is the point where we are adding just noise by adding model complexity (flexibility).
- The training error goes down as it has to, but the test error is starting to go up.
- The model after the bias trade-off begins to overfit.



Bias Variance Trade-Off

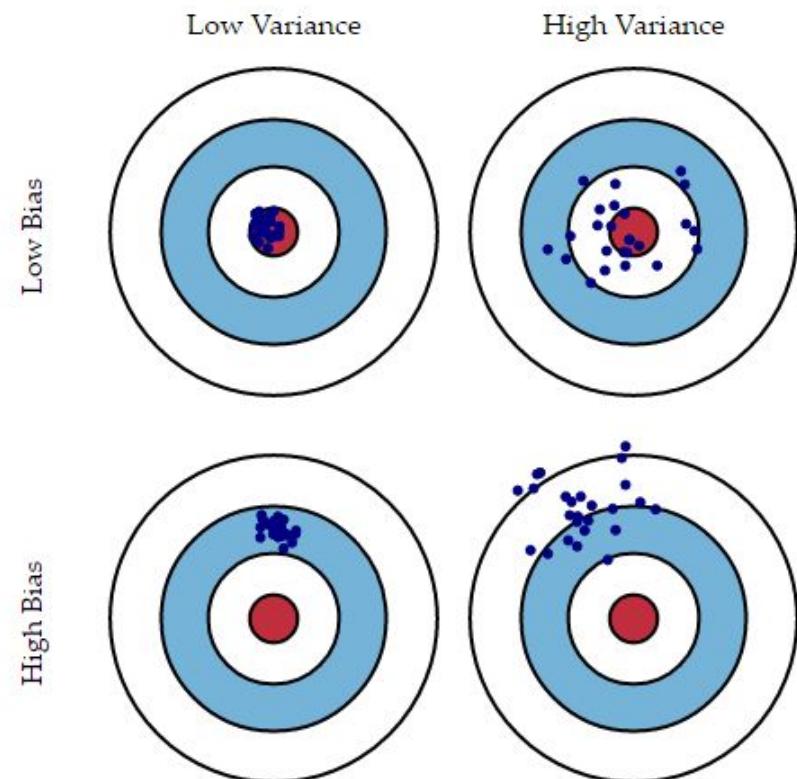
- Imagine that the center of the target is a model that perfectly predicts the correct values.
- As we move away from the bulls-eye, our predictions get worse and worse.





Bias Variance Trade-Off

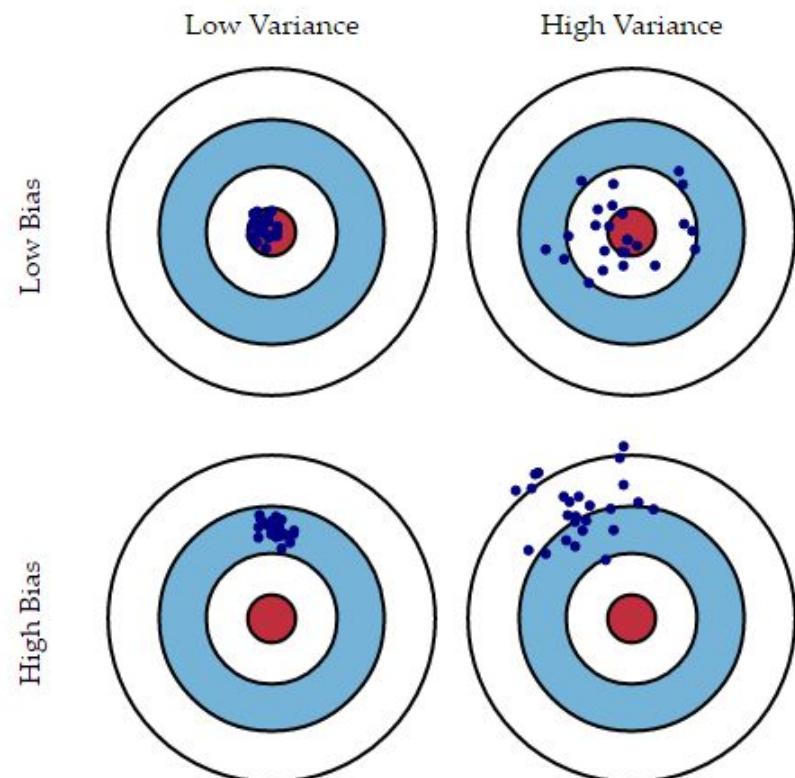
- Imagine we can repeat our entire model building process to get a number of separate hits on the target.
- Each hit represents an individual realization of our model, given the chance variability in the training data we gather.





Bias Variance Trade-Off

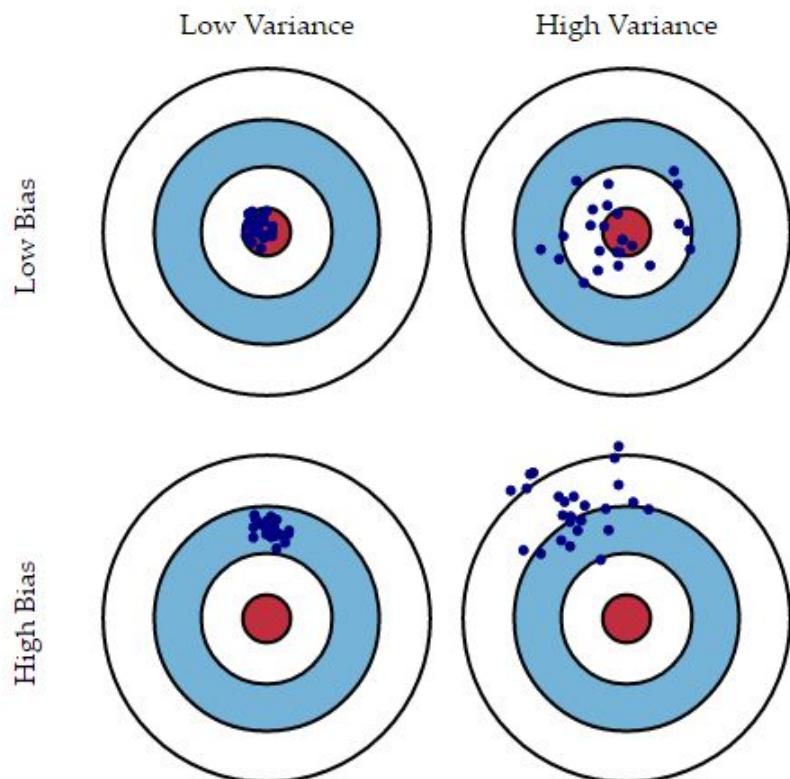
- Sometimes we will get a good distribution of training data so we predict very well and we are close to the bulls-eye, while sometimes our training data might be full of outliers or non-standard values resulting in poorer predictions.





Bias Variance Trade-Off

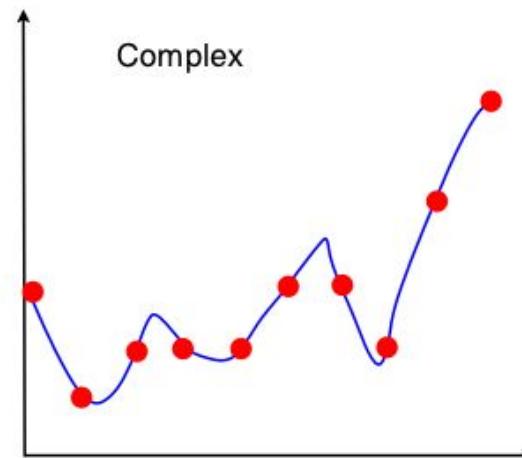
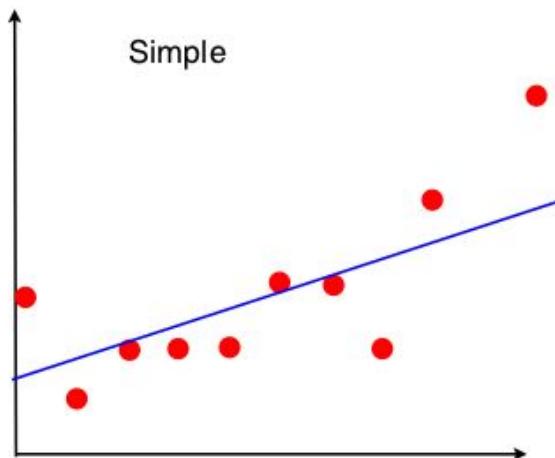
- These different realizations result in a scatter of hits on the target.





Bias Variance Trade-Off

- A common temptation for beginners is to continually add complexity to a model until it fits the training set very well.



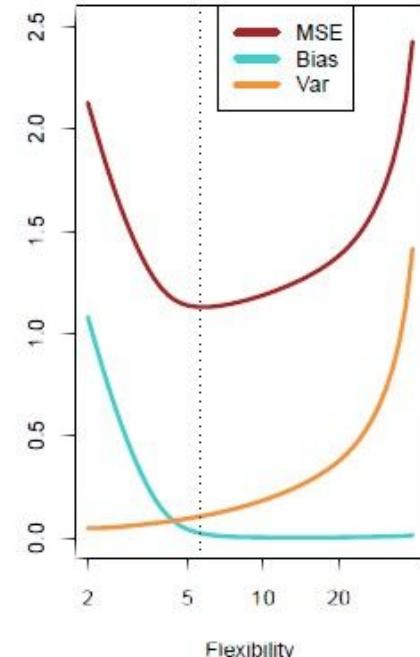
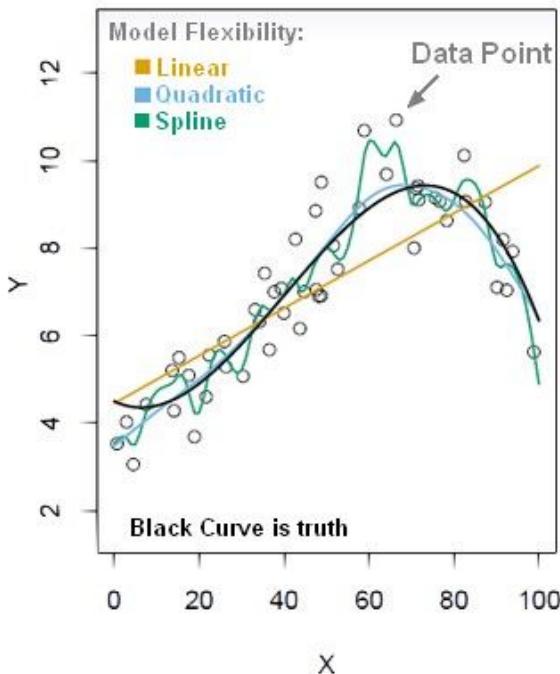


Bias Variance Trade-Off

- Doing this can cause a model to overfit to your training data and cause large errors on new data, such as the test set.
- Let's take a look at an example model on how we can see overfitting occur from a error standpoint using test data!
- We'll use a black curve with some “noise” points off of it to represent the True shape the data follows.

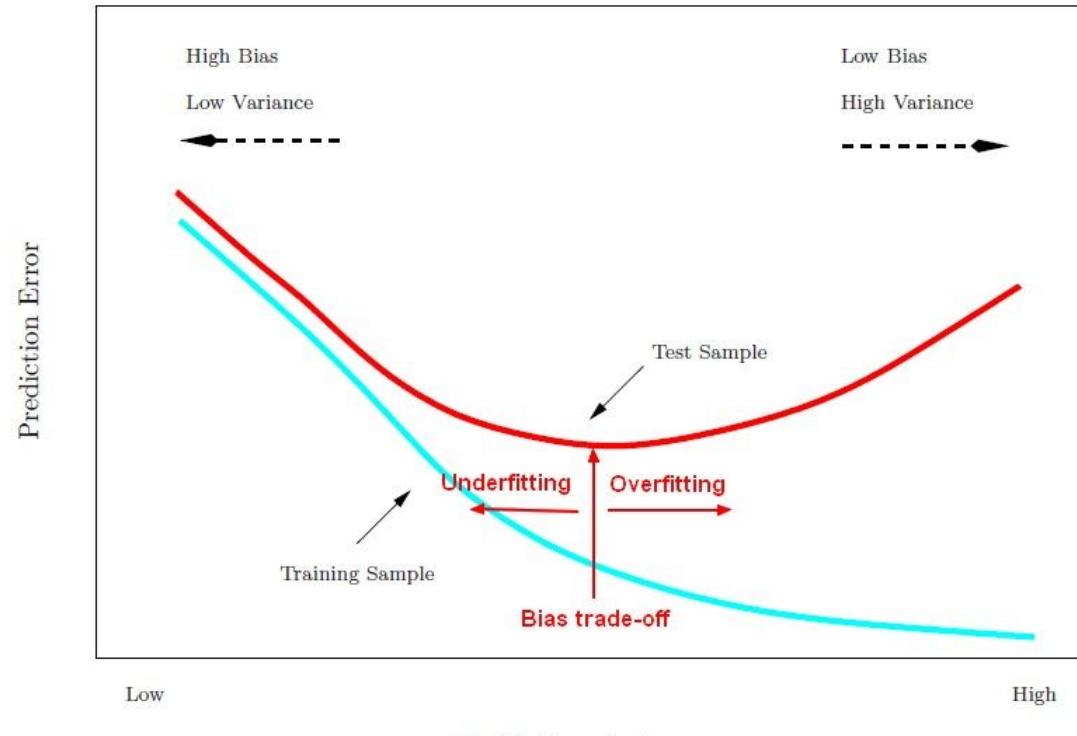


Bias Variance Trade-Off





Bias Variance Trade-Off





**Let's continue by going
through some examples and
exploring the documentation!**



Introduction to Logistic Regression



Reading Assignment

Sections 4-4.3 of
Introduction to Statistical Learning
By Gareth James, et al.



Background

- We want to learn about Logistic Regression as a method for **Classification**.
- Some examples of classification problems:
 - Spam versus “Ham” emails
 - Loan Default (yes/no)
 - Disease Diagnosis
- Above were all examples of Binary Classification



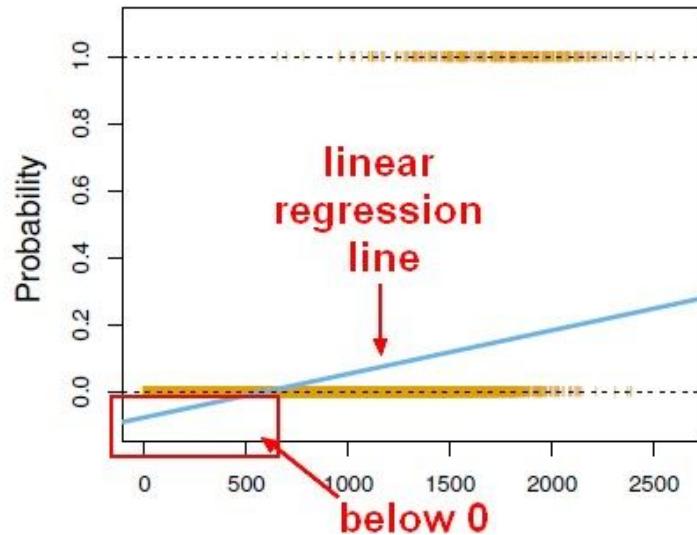
Background

- So far we've only seen regression problems where we try to predict a continuous value.
- Although the name may be confusing at first, logistic regression allows us to solve classification problems, where we are trying to predict discrete categories.
- The convention for binary classification is to have two classes 0 and 1.



Background

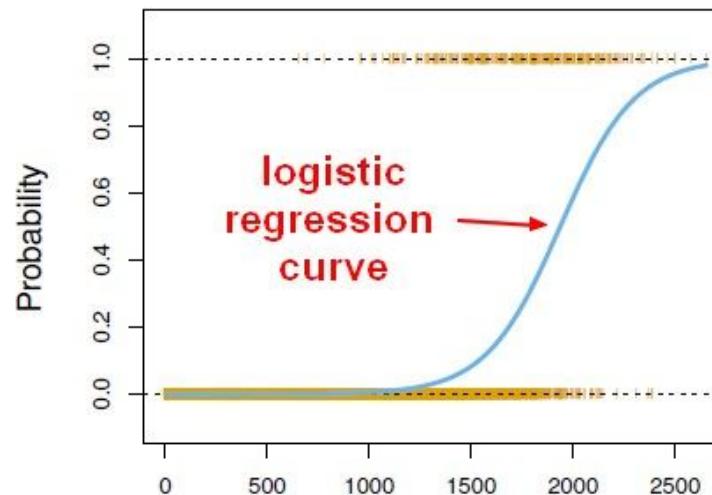
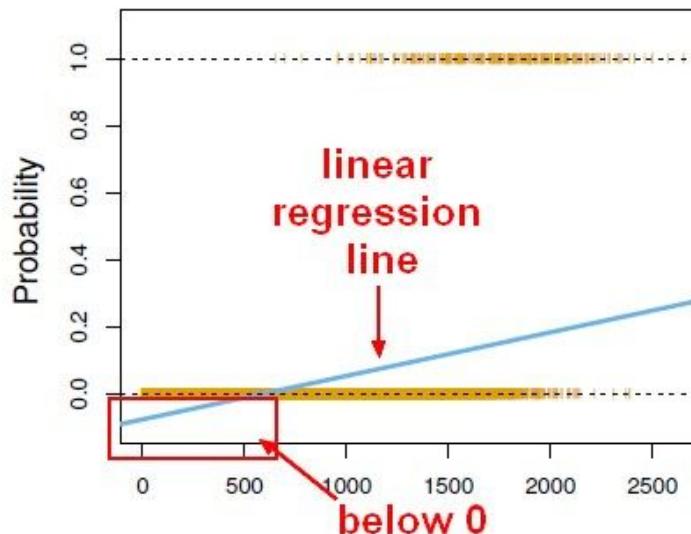
- We can't use a normal linear regression model on binary groups. It won't lead to a good fit:





Background

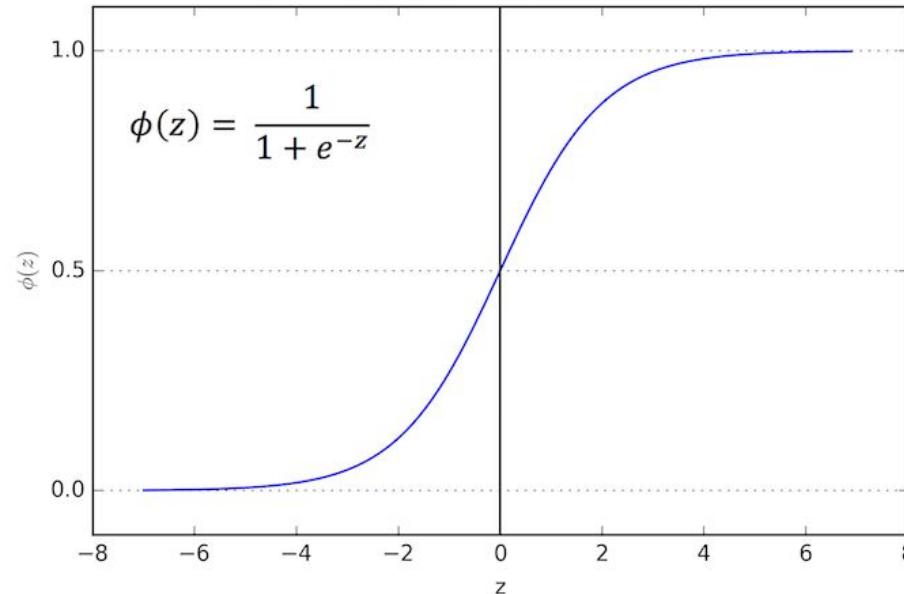
- Instead we can transform our linear regression to a logistic regression curve.





Sigmoid Function

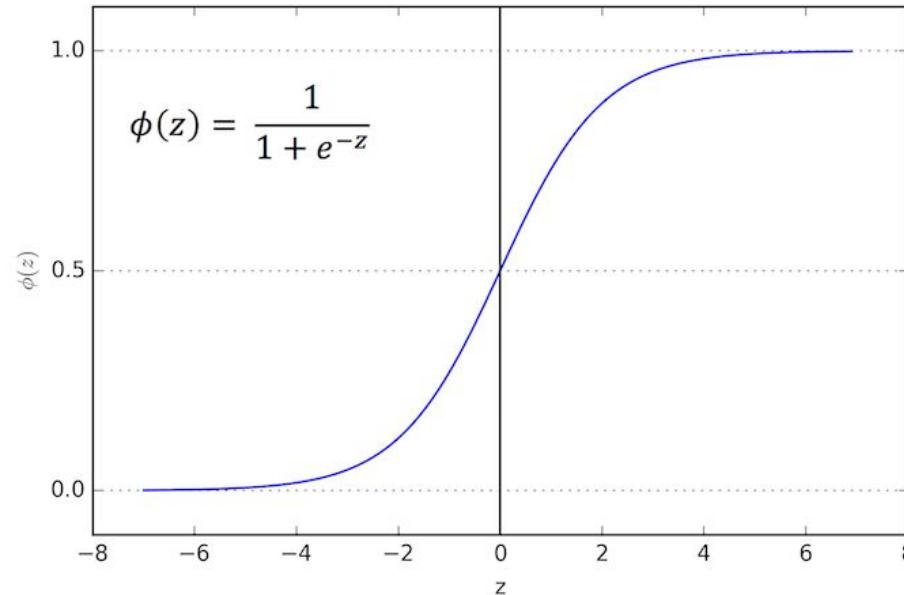
- The Sigmoid (aka Logistic) Function takes in any value and outputs it to be between 0 and 1.





Sigmoid Function

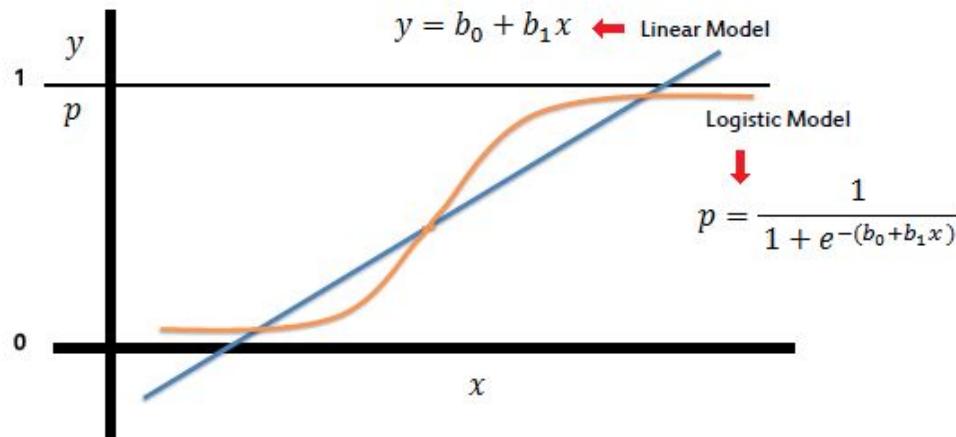
- This means we can take our Linear Regression Solution and place it into the Sigmoid Function.





Sigmoid Function

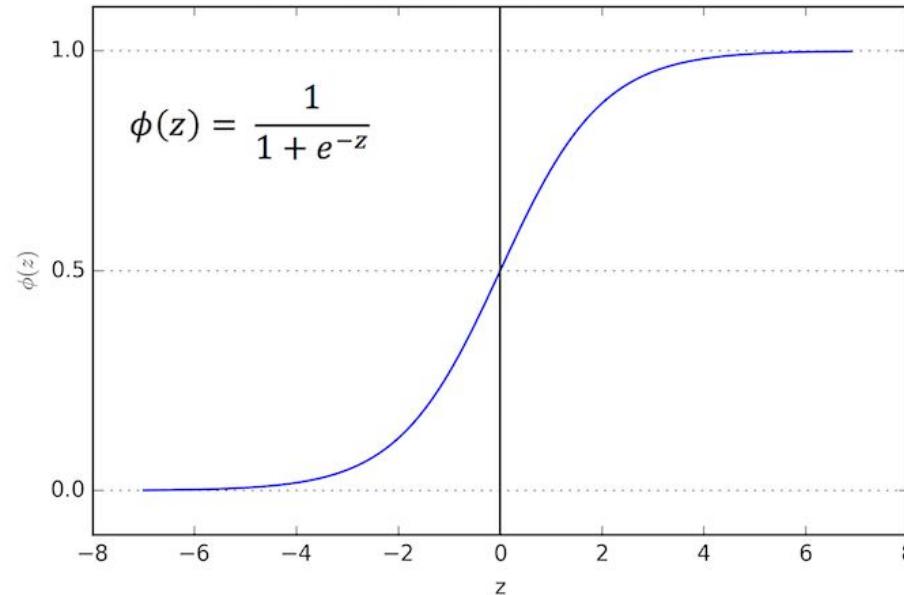
- This means we can take our Linear Regression Solution and place it into the Sigmoid Function.





Sigmoid Function

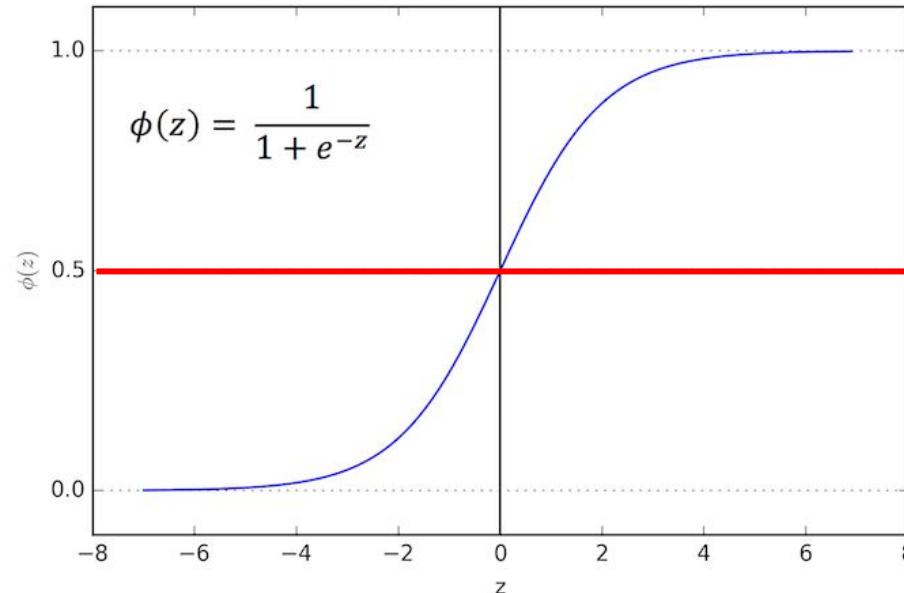
- This results in a probability from 0 to 1 of belonging in the 1 class.





Sigmoid Function

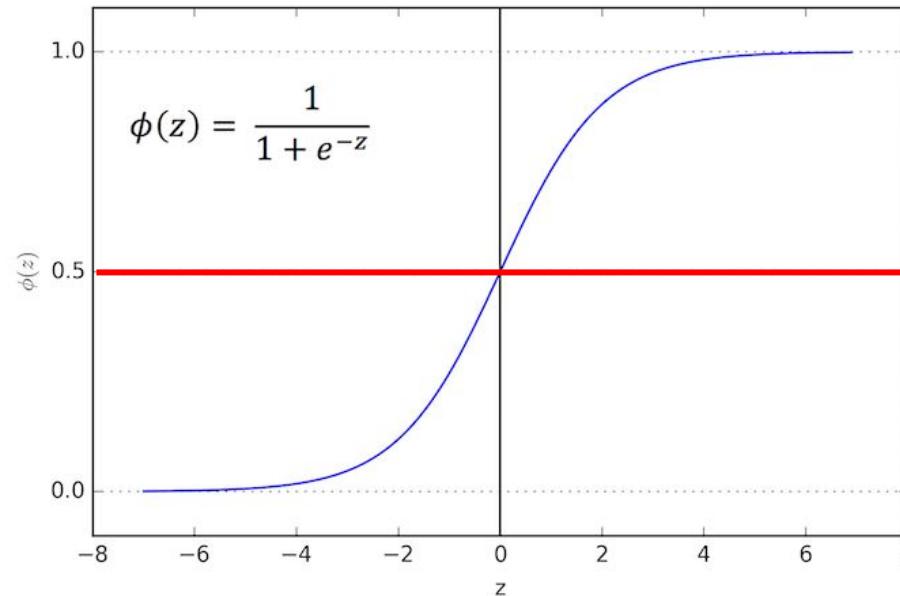
- We can set a cutoff point at 0.5, anything below it results in class 0, anything above is class 1.





Review

- We use the logistic function to output a value ranging from 0 to 1. Based off of this probability we assign a class.





Model Evaluation

- After you train a logistic regression model on some training data, you will evaluate your model's performance on some test data.
- You can use a confusion matrix to evaluate classification models.



Model Evaluation

- We can use a confusion matrix to evaluate our model.
- For example, imagine testing for disease.

n=165	Predicted:	
	NO	YES
Actual: NO	50	10
Actual: YES	5	100

Example: Test for presence of disease
NO = negative test = False = 0
YES = positive test = True = 1



Confusion Matrix

n=165	Predicted:		
	NO	YES	
Actual:			
NO	TN = 50	FP = 10	60
YES	FN = 5	TP = 100	105
	55	110	

Basic Terminology:

- True Positives (TP)
- True Negatives (TN)
- False Positives (FP)
- False Negatives (FN)



Confusion Matrix

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Accuracy:

- Overall, how often is it **correct?**
- $(TP + TN) / \text{total} = 150/165 = 0.91$



Confusion Matrix

n=165	Predicted:		
	NO	YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

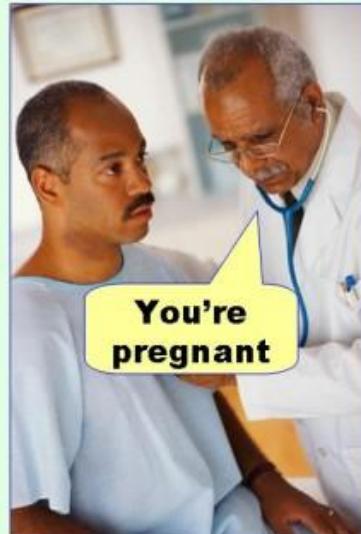
Misclassification Rate (Error Rate):

- Overall, how often is it **wrong?**
- $(FP + FN) / \text{total} = 15/165 = 0.09$



Confusion Matrix

Type I error
(false positive)



Type II error
(false negative)



LOGISTIC REGRESSION

```
In [ ]: Steps for the basic prediction:
Step1:Read the required dataset
Step2:Using heatmaps and isnan() find all the missing values
Step3:Clean the data
Step4:If less data is missing then add average values in the place
Step5:If more data is missing then drop the column
Step6:Convert the all the datas into binary of 0 and 1 for example:In gender we cre
        with Sex column and made 1 as male and female as 0
Step 7:Remove unwanted columns for the analysis
```

I REQUEST TO CLEAN THE DATAS AS MUCH AS POSSIBLE AND MAKE IT IN NUMBERS FOR THE ACCURATE PREDICTION

```
In [1]: #Importing all the required packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
In [2]: #Reading the csv file
train=pd.read_csv("titanic_train.csv")
```

```
In [3]: #Viewing the first 5 datas
train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
4	5	0	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

In [4]: `#Displaying the columns
train.columns`

Out[4]: `Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
dtype='object')`

In [5]: `train.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
 ---  --          --          --      
 0   PassengerId 891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare          891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

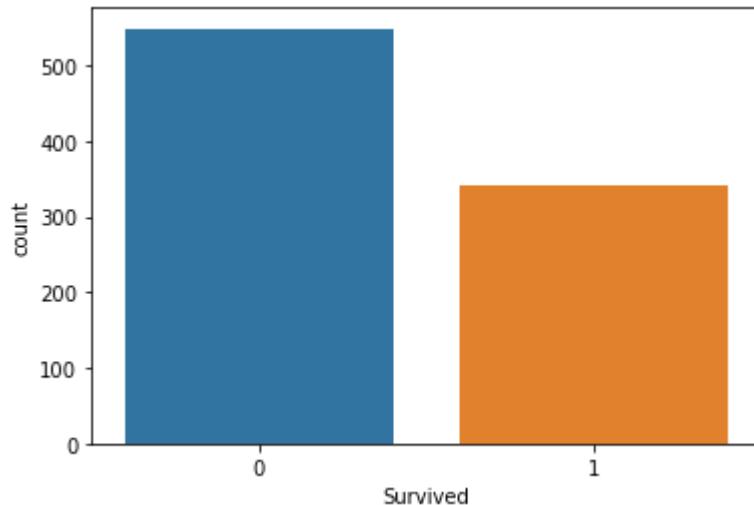
In [6]: `train.describe()`

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Data-Analysis Exploratory Plots

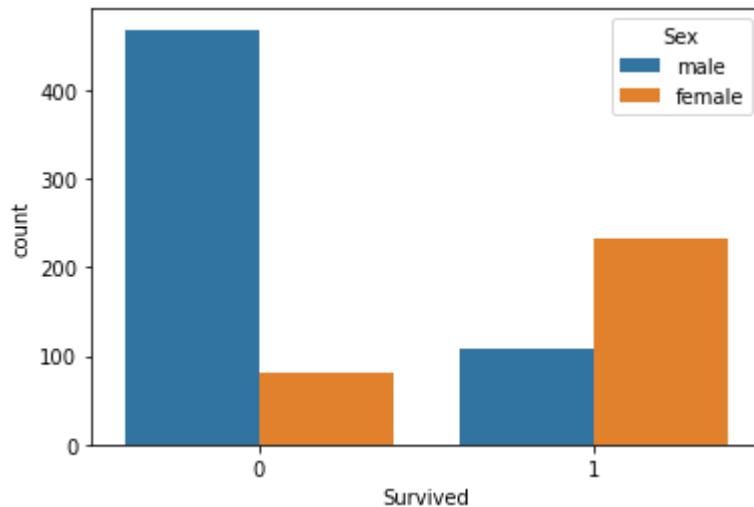
In [8]: `#He we use count plot to find how many has survived and not survived
#0 is not Survived and 1 is Survived
sns.countplot(x="Survived",data=train)`

```
Out[8]: <AxesSubplot:xlabel='Survived', ylabel='count'>
```



```
In [10]: #Plot used to differentiate survived are male or female  
sns.countplot(x="Survived",hue="Sex",data=train)
```

```
Out[10]: <AxesSubplot:xlabel='Survived', ylabel='count'>
```



```
In [12]: #Finding the details which are blank  
#For this we use isnull  
train.isnull()
```

```
Out[12]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0		False	False	False	False	False	False	False	False	False	True	False
1		False	False	False	False	False	False	False	False	False	False	False
2		False	False	False	False	False	False	False	False	False	True	False
3		False	False	False	False	False	False	False	False	False	False	False
4		False	False	False	False	False	False	False	False	False	True	False
...	
886		False	False	False	False	False	False	False	False	False	True	False

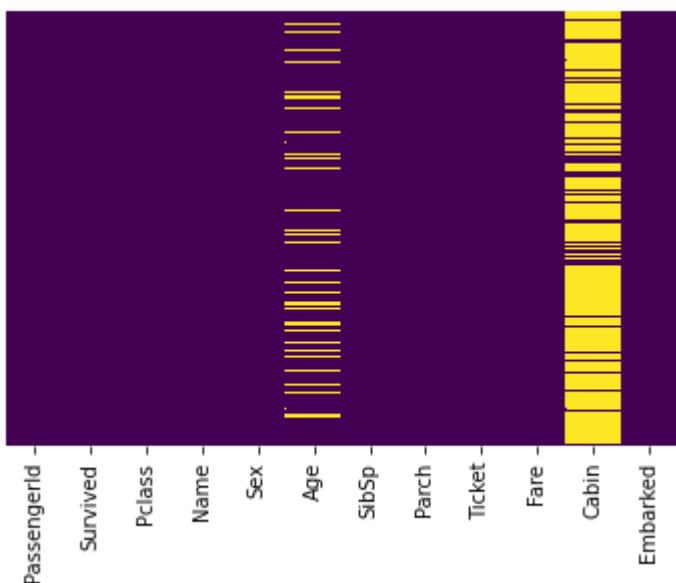
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
887		False	False	False	False	False	False	False	False	False	False	False
888		False	False	False	False	True	False	False	False	False	True	False
889		False	False	False	False	False	False	False	False	False	False	False
890		False	False	False	False	False	False	False	False	False	True	False

891 rows × 12 columns

In [11]:

```
#Here we display the details which is not filled
#Purple coloured are filled details and yellow coloured are unfilled details
#in age 20 % of details are not filled
#In cabin 90 % of the details are not filled
#Understanding with heatmap
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap="viridis")
```

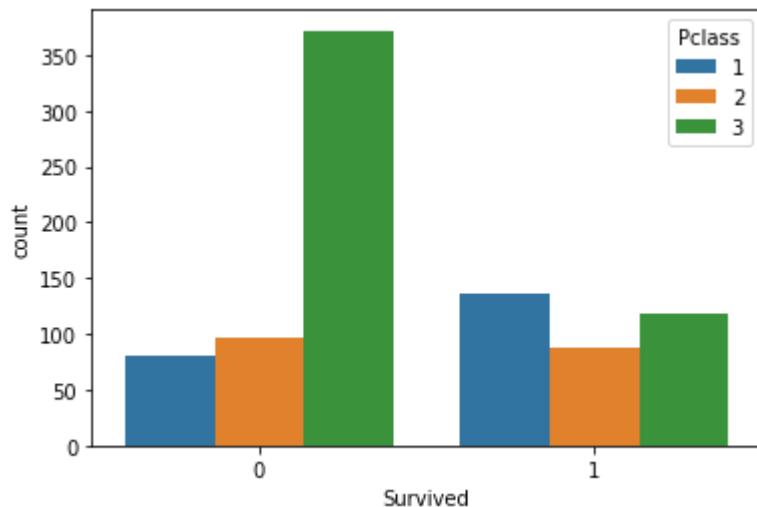
Out[11]: <AxesSubplot:>



In [17]:

```
#Plot to view how many members are survived according to the class
sns.countplot(x="Survived",hue="Pclass",data=train)
```

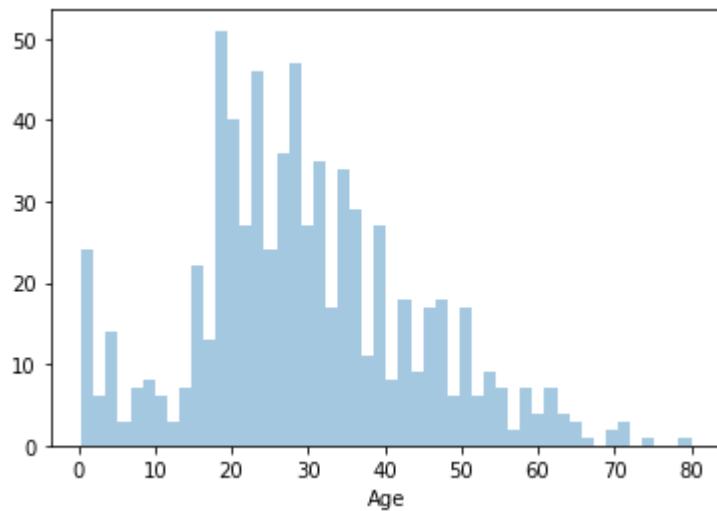
Out[17]: <AxesSubplot:xlabel='Survived', ylabel='count'>



```
In [18]: #Here dropna will filter the nan values in the table
sns.distplot(train["Age"].dropna(), kde=False, bins=50)
```

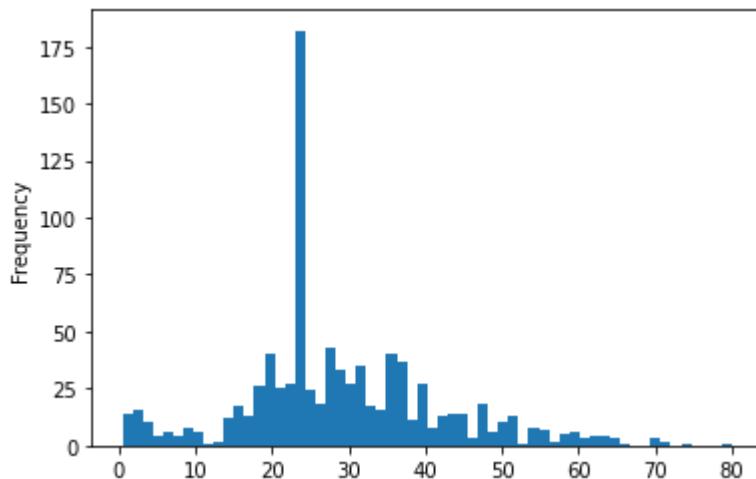
C:\Users\srena\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
 warnings.warn(msg, FutureWarning)

```
Out[18]: <AxesSubplot:xlabel='Age'>
```



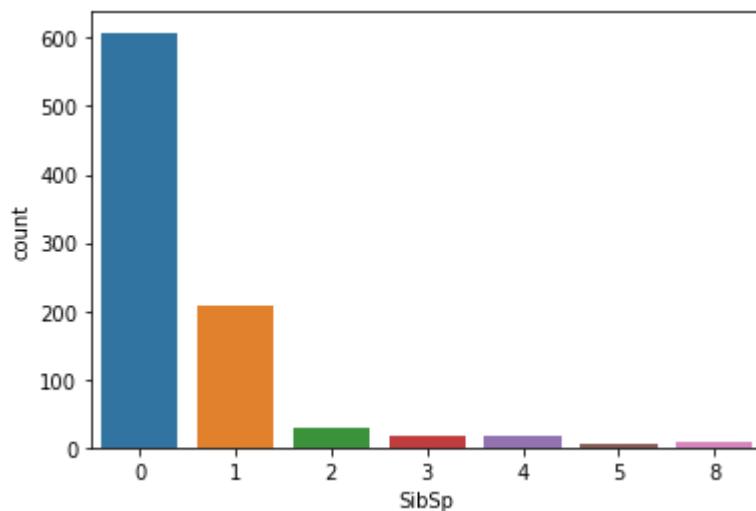
```
In [48]: #Plotting the histogram of Age
train["Age"].plot.hist(bins=60)
```

```
Out[48]: <AxesSubplot:ylabel='Frequency'>
```



```
In [21]: #Counting how many sibling or wifes are availabe per person in the dataset
sns.countplot(x="SibSp",data=train)
```

```
Out[21]: <AxesSubplot:xlabel='SibSp', ylabel='count'>
```



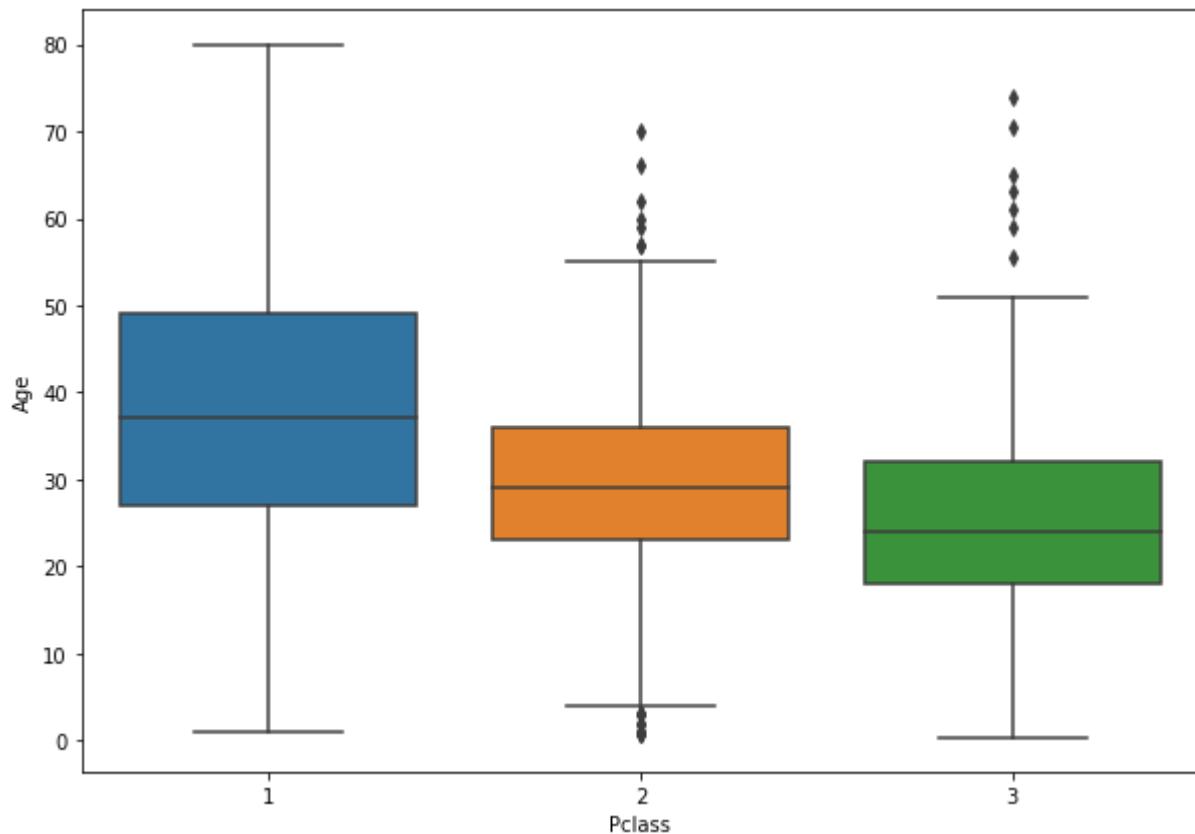
DATA CLEANING

```
In [22]: import cufflinks as cf
```

```
In [23]: cf.go_offline()
```

```
In [28]: #Here we are analysing the details the age according the age
#For Example to fill the age in class one the common age in between 30 and 50 so we take
plt.figure(figsize=(10,7))
sns.boxplot(x="Pclass",y="Age",data=train)
```

```
Out[28]: <AxesSubplot:xlabel='Pclass', ylabel='Age'>
```



```
In [30]: def impute_age(cols):
    #On applying the details the age will be column[0]
    #Pclass in the column[1]
    Age=cols[0]
    Pclass=cols[1]

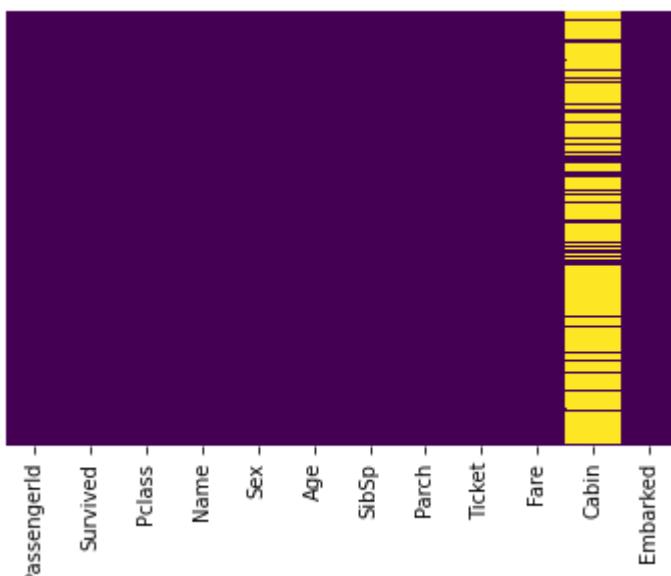
    if pd.isnull(Age):

        if Pclass==1:
            return 37
        elif Pclass==2:
            return 29
        else:
            return 24
    else:
        return Age
```

```
In [31]: #Applying the function into the train dataset
train["Age"]=train[["Age","Pclass"]].apply(impute_age, axis=1)
```

```
In [35]: #Now the age is cleared
sns.heatmap(train.isnull(), yticklabels=False, cbar=False, cmap="viridis")
```

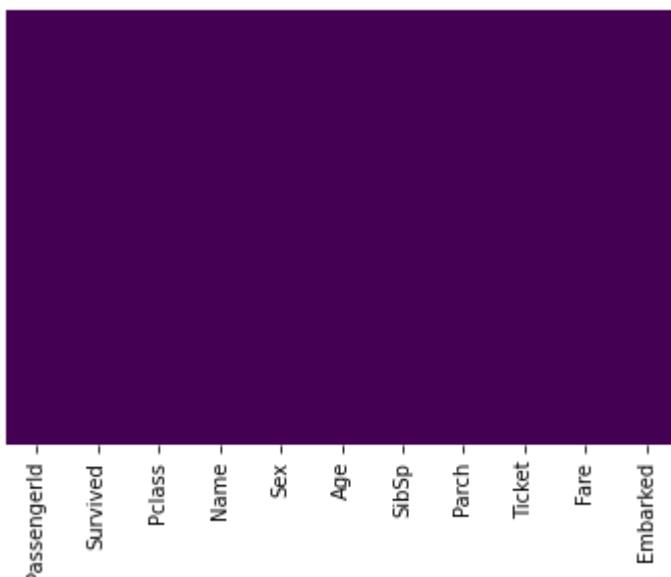
```
Out[35]: <AxesSubplot:>
```



In [36]: *#WE are droping the cabin as we dont have much details*
`train.drop("Cabin",axis=1,inplace=True)`

In [37]: *#now all the missing values are cleared*
`sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap="viridis")`

Out[37]: <AxesSubplot:>



In [40]: *#We are creating dummie values for the Sex column to convert form string to binary of 0
#Because we can't analyse the string
#We are taking the male column and droping female column
#1 means male and 0 means female*
`sex=pd.get_dummies(train["Sex"],drop_first=True)`

In [41]: *#Same we are making with embarked with binary of 0 and 1*
`embark=pd.get_dummies(train["Embarked"],drop_first=True)`

In [43]: *#Finally we are adding the sex and embark column in the train dataset using concat in a*
`train=pd.concat([train,sex,embark],axis=1)`

In [44]: `#Viewing the datas
train.head(2)`

Out[44]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	male
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C	

In [47]: `#Here "PassengerId", "Name", "Sex", "Ticket", "Embarked" are unwanted details in the database
train.drop(["PassengerId", "Name", "Sex", "Ticket", "Embarked"], axis=1, inplace=True)`

In [51]: `train.head()`

Out[51]:

	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	0	3	22.0	1	0	7.2500	1	0	1
1	1	1	38.0	1	0	71.2833	0	0	0
2	1	3	26.0	0	0	7.9250	0	0	1
3	1	1	35.0	1	0	53.1000	0	0	1
4	0	3	35.0	0	0	8.0500	1	0	1

In [50]: `#Now all the datas are cleared and ready for prediction`

In [57]: `from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression`

In [53]: `#Here all the columns other than Survived are taken as X for features
X=train.drop(["Survived"],axis=1)
#We are Taking Survived for prediction
y=train["Survived"]`

In [56]: `X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=101)`

In [59]: `logistic_model=LogisticRegression()`

In [60]: `logistic_model.fit(X_train,y_train)`

C:\Users\srena\anaconda3\lib\site-packages\sklearn\linear_model_logistic.py:762: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

Out[60]: LogisticRegression()

In [61]: prediction=logistic_model.predict(X_test)

In [62]: from sklearn.metrics import classification_report

In [63]: print(classification_report(y_test,prediction))

	precision	recall	f1-score	support
0	0.78	0.86	0.82	154
1	0.78	0.67	0.72	114
accuracy			0.78	268
macro avg	0.78	0.77	0.77	268
weighted avg	0.78	0.78	0.78	268

In [64]: from sklearn.metrics import confusion_matrix

In [65]: confusion_matrix(y_test,prediction)

Out[65]: array([[133, 21],
[38, 76]], dtype=int64)

CONFUSION MATRIX

In []: TN--->TRUE NEGATIVE
TP--->TRUE POSITIVE
FN--->FALSE NEGATIVE
FP--->FALSE POSITIVE

	Predicted(No)	Predicted(Yes)	
Actual(No)	TN=133	FP=21	154
Actual(Yes)	FN=38	TP=76	114
	171	97	

LOGISTIC REGRESSION COMPLETED

In []:



Introduction to K Nearest Neighbors



Reading Assignment

Complete Chapter 4
Introduction to Statistical Learning
By Gareth James, et al.



KNN

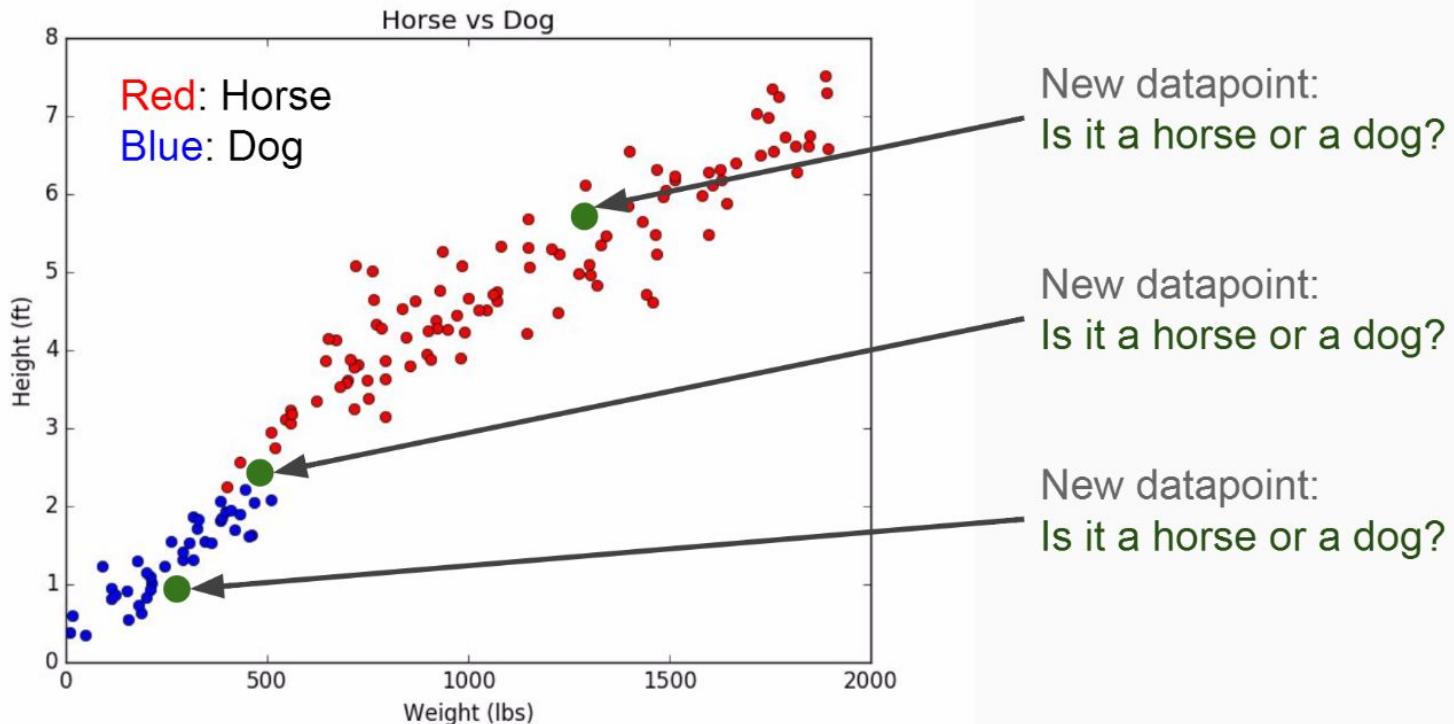
K Nearest Neighbors is a **classification** algorithm that operates on a very simple principle.

It is best shown through example!

Imagine we had some imaginary data on Dogs and Horses, with heights and weights.



KNN





KNN

Training Algorithm:

1. Store all the Data

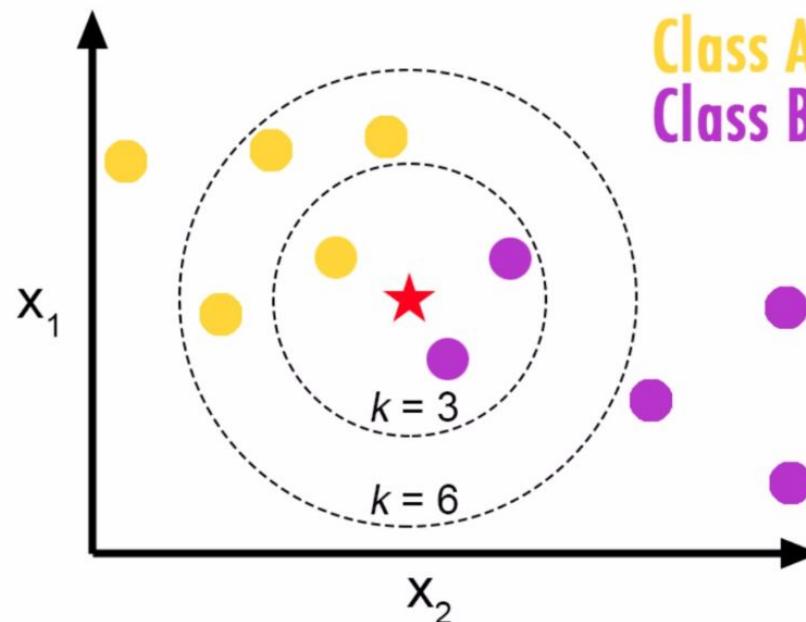
Prediction Algorithm:

1. Calculate the distance from x to all points in your data
2. Sort the points in your data by increasing distance from x
3. Predict the majority label of the “ k ” closest points



KNN

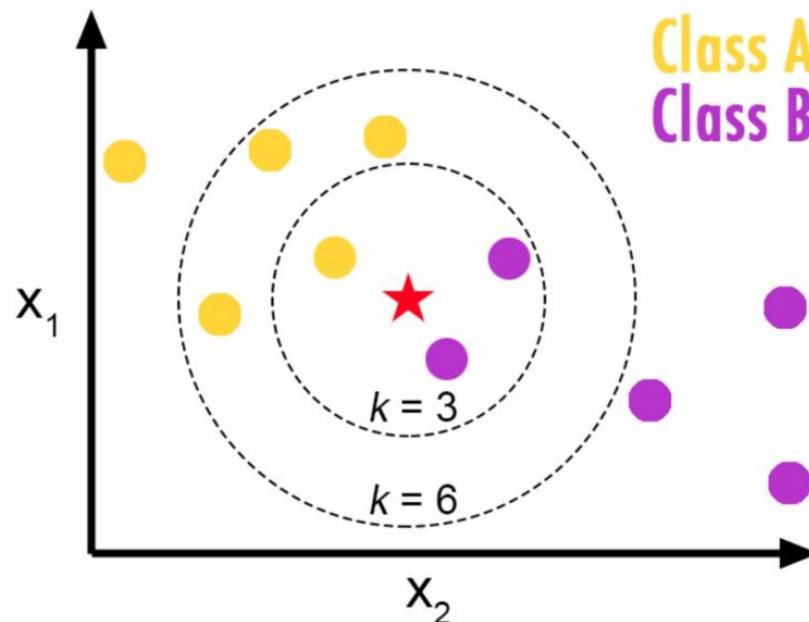
Choosing a K will affect what class a new point is assigned to:





KNN

Choosing a K will affect what class a new point is assigned to:

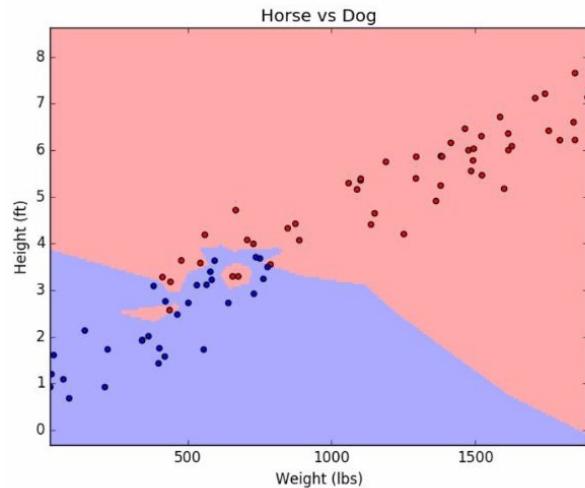




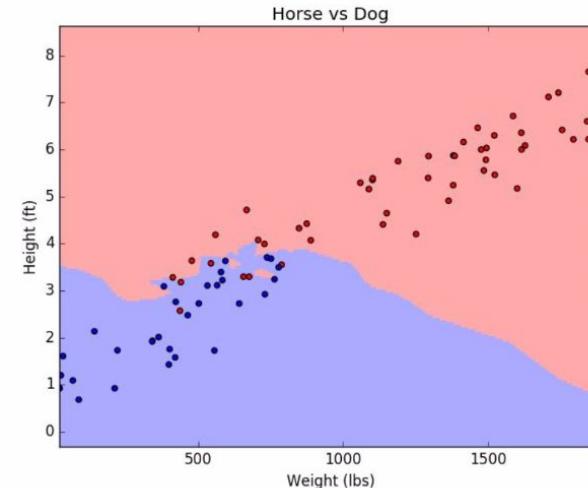
KNN

Choosing a K will affect what class a new point is assigned to:

k=1



k=5

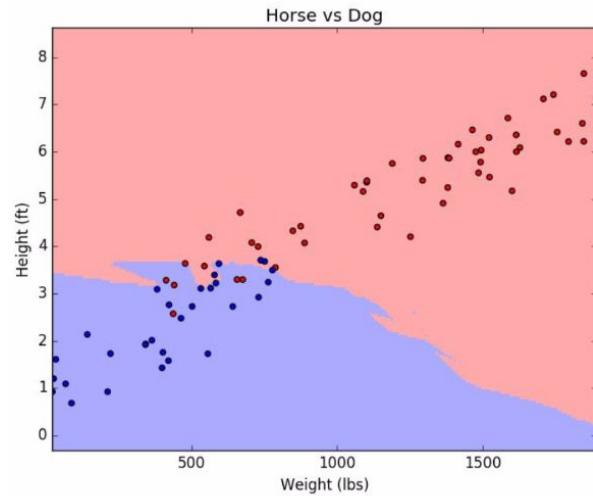




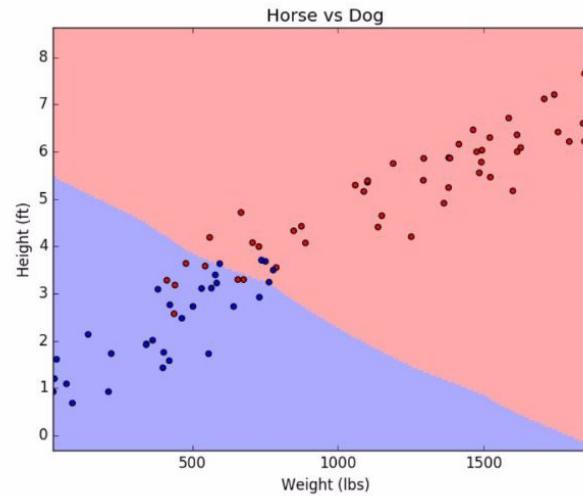
KNN

Choosing a K will affect what class a new point is assigned to:

k=10



k=50





KNN

Pros

- Very simple
- Training is trivial
- Works with any number of classes
- Easy to add more data
- Few parameters
 - K
 - Distance Metric



KNN

Cons

- High Prediction Cost (worse for large data sets)
- Not good with high dimensional data
- Categorical Features don't work well

K NEAREST NEIGHBOURS

```
In [1]: #Importing all the required datas
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
In [2]: #Reading the csv data
df=pd.read_csv("Classified Data")
```

```
In [3]: df
```

Out[3]:

	Unnamed: 0	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE
0	0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422
1	1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552
2	2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877
3	3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692
4	4	1.279491	0.949750	0.627280	0.668976	1.232537	0.703727	1.115596	0.646691	1.463812
...
995	995	1.010953	1.034006	0.853116	0.622460	1.036610	0.586240	0.746811	0.319752	1.117340
996	996	0.575529	0.955786	0.941835	0.792882	1.414277	1.269540	1.055928	0.713193	0.958684
997	997	1.135470	0.982462	0.781905	0.916738	0.901031	0.884738	0.386802	0.389584	0.919191
998	998	1.084894	0.861769	0.407158	0.665696	1.608612	0.943859	0.855806	1.061338	1.277456
999	999	0.837460	0.961184	0.417006	0.799784	0.934399	0.424762	0.778234	0.907962	1.257190

1000 rows × 12 columns

```
In [21]: df.head()
```

Out[21]:

	Unnamed: 0	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE
0	0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422
1	1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552
2	2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877
3	3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692
4	4	1.279491	0.949750	0.627280	0.668976	1.232537	0.703727	1.115596	0.646691	1.463812

```
In [22]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Unnamed: 0    1000 non-null   int64  
 1   WTT          1000 non-null   float64 
 2   PTI          1000 non-null   float64 
 3   EQW          1000 non-null   float64 
 4   SBI          1000 non-null   float64 
 5   LQE          1000 non-null   float64 
 6   QWG          1000 non-null   float64 
 7   FDJ          1000 non-null   float64 
 8   PJF          1000 non-null   float64 
 9   HQE          1000 non-null   float64 
 10  NXJ          1000 non-null   float64 
 11  TARGET CLASS 1000 non-null   int64  
dtypes: float64(10), int64(2)
memory usage: 93.9 KB
```

In [23]: df.describe()

	Unnamed: 0	WTT	PTI	EQW	SBI	LQE	QWG
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	499.500000	0.949682	1.114303	0.834127	0.682099	1.032336	0.943534
std	288.819436	0.289635	0.257085	0.291554	0.229645	0.243413	0.256121
min	0.000000	0.174412	0.441398	0.170924	0.045027	0.315307	0.262389
25%	249.750000	0.742358	0.942071	0.615451	0.515010	0.870855	0.761064
50%	499.500000	0.940475	1.118486	0.813264	0.676835	1.035824	0.941502
75%	749.250000	1.163295	1.307904	1.028340	0.834317	1.198270	1.123060
max	999.000000	1.721779	1.833757	1.722725	1.634884	1.650050	1.666902

In [24]: df.columns

Out[24]: Index(['Unnamed: 0', 'WTT', 'PTI', 'EQW', 'SBI', 'LQE', 'QWG', 'FDJ', 'PJF', 'HQE', 'NXJ', 'TARGET CLASS'],
dtype='object')

In [25]: #Importing StandardScaler from sklearn where StandardScalar is Model
from sklearn.preprocessing import StandardScaler

In [28]: #Creating an model
scaler=StandardScaler()

In [30]: #Fitting the data into the model
#WE are fitting the data other than TARGET CLASS
scaler.fit(df.drop("TARGET CLASS",axis=1))

Out[30]: StandardScaler()

```
In [32]: #Helps to fill missed data while transforming
scaler_feature=scaler.transform(df.drop("TARGET CLASS",axis=1))
```

```
In [37]: #Creating an Dataframe with the datas
#Here we are removing columns[:-1] because in previous cell we use dropna to remove TAR
#columns with TARGET CLASS is 12 columns without TARGET CLASS is 11 so due to this we t
df_feat=pd.DataFrame(scaler_feature,columns=df.columns[:-1])
```

```
In [38]: df_feat
```

Out[38]:

	Unnamed: 0	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF
0	-1.730320	-0.123542	0.185907	-0.913431	0.319629	-1.033637	-2.308375	-0.798951	-1.482368
1	-1.726856	-1.084836	-0.430348	-1.025313	0.625388	-0.444847	-1.152706	-1.129797	-0.202240
2	-1.723391	-0.788702	0.339318	0.301511	0.755873	2.031693	-0.870156	2.599818	0.285707
3	-1.719927	0.982841	1.060193	-0.621399	0.625299	0.452820	-0.267220	1.750208	1.066491
4	-1.716463	1.139275	-0.640392	-0.709819	-0.057175	0.822886	-0.936773	0.596782	-1.472352
...
995	1.716463	0.211653	-0.312490	0.065163	-0.259834	0.017567	-1.395721	-0.849486	-2.604264
996	1.719927	-1.292453	-0.616901	0.369613	0.482648	1.569891	1.273495	0.362784	-1.242110
997	1.723391	0.641777	-0.513083	-0.179205	1.022255	-0.539703	-0.229680	-2.261339	-2.362494
998	1.726856	0.467072	-0.982786	-1.465194	-0.071465	2.368666	0.001269	-0.422041	-0.036777
999	1.730320	-0.387654	-0.595894	-1.431398	0.512722	-0.402552	-2.026512	-0.726253	-0.567789

1000 rows × 11 columns



-->These above process is used to make the prediction in accurate manner

```
In [40]: from sklearn.model_selection import train_test_split
X=df_feat
y=df["TARGET CLASS"]
#Spliting the datas into training and testing data
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.4,random_state=101)
```

```
In [41]: #We are taking another model KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier
```

```
In [49]: #Creating an model with k=1
knn=KNeighborsClassifier(n_neighbors=1)
```

```
In [50]: #Fitting the model with datas
knn.fit(X_train,y_train)
```

```
Out[50]: KNeighborsClassifier(n_neighbors=1)
```

```
In [51]: prediction=knn.predict(X_test)
```

```
In [52]: from sklearn.metrics import classification_report,confusion_matrix
```

```
In [53]: print(classification_report(y_test,prediction))
```

	precision	recall	f1-score	support
0	0.90	0.93	0.91	204
1	0.92	0.89	0.91	196
accuracy			0.91	400
macro avg	0.91	0.91	0.91	400
weighted avg	0.91	0.91	0.91	400

```
In [48]: print(confusion_matrix(y_test,prediction))
```

```
[[190 14]
 [ 15 181]]
```

Choosing a K Value

Let's go ahead and use the elbow method to pick a good K Value:

```
In [54]: #Choosing the k value with less error_rate for accurate prediction
```

```
error_rate = []

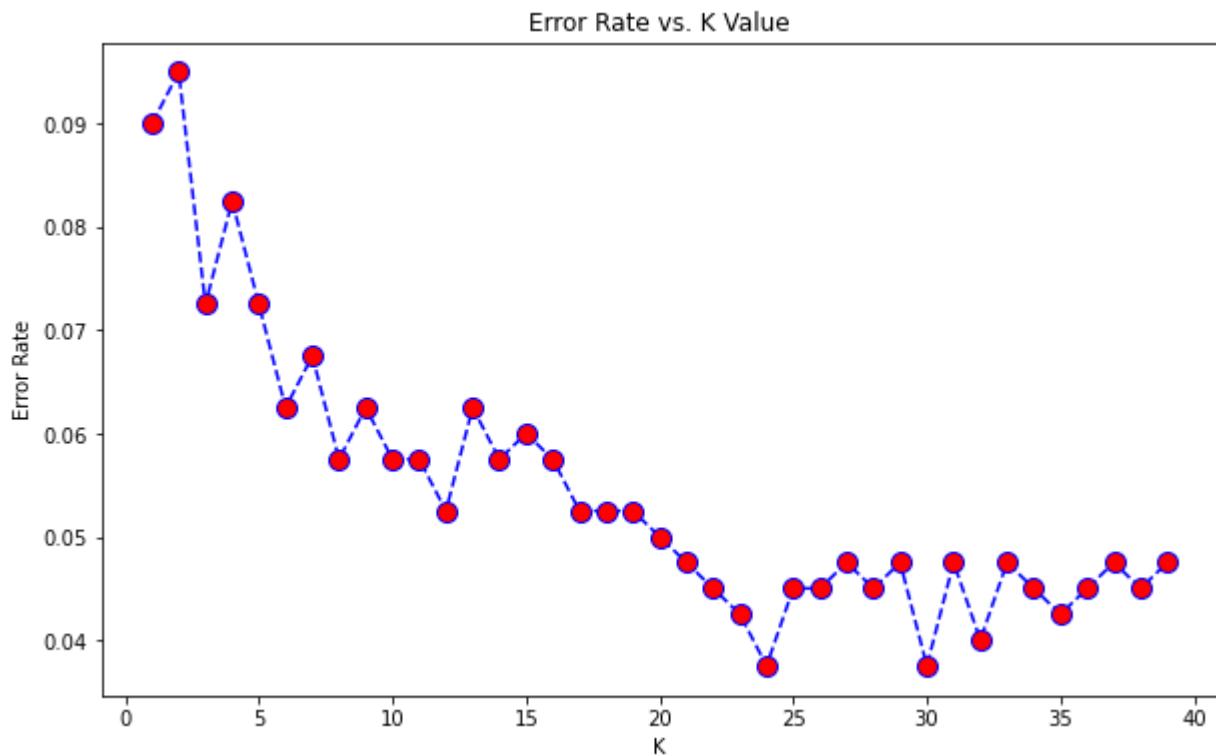
# Will take some time
#For Loop in range of 1 to 40 as k
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    #Appending the error_rate where the formula is mean(predicted_value not equal to y_
    error_rate.append(np.mean(pred_i != y_test))
```

```
In [55]: #Displaying the datas in visualization plots
```

```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

```
Out[55]: Text(0, 0.5, 'Error Rate')
```



Here we can see that after around K>23 the error rate just tends to hover around 0.06-0.05 Let's retrain the model with that and check the classification report!

```
In [56]: # FIRST A QUICK COMPARISON TO OUR ORIGINAL K=1
knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=1')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

WITH K=1

```
[[189 15]
 [ 21 175]]
```

	precision	recall	f1-score	support
0	0.90	0.93	0.91	204
1	0.92	0.89	0.91	196
accuracy			0.91	400
macro avg	0.91	0.91	0.91	400
weighted avg	0.91	0.91	0.91	400

```
In [57]: # NOW WITH K=23
knn = KNeighborsClassifier(n_neighbors=23)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)
```

```
print('WITH K=23')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

WITH K=23

```
[[195  9]
 [ 8 188]]
```

	precision	recall	f1-score	support
0	0.96	0.96	0.96	204
1	0.95	0.96	0.96	196
accuracy			0.96	400
macro avg	0.96	0.96	0.96	400
weighted avg	0.96	0.96	0.96	400

In []: *#Therefore we get accurate prediction value*



Introduction to Tree Methods



Reading Assignment

Chapter 8 of
Introduction to Statistical Learning
By Gareth James, et al.



Tree Methods

Let's start off with a thought experiment to give some motivation behind using a decision tree method.



Tree Methods

Imagine that I play Tennis every Saturday and I always invite a friend to come with me.

Sometimes my friend shows up, sometimes not.

For him it depends on a variety of factors, such as: weather, temperature, humidity, wind etc..

I start keeping track of these features and whether or not he showed up to play with me.



Tree Methods

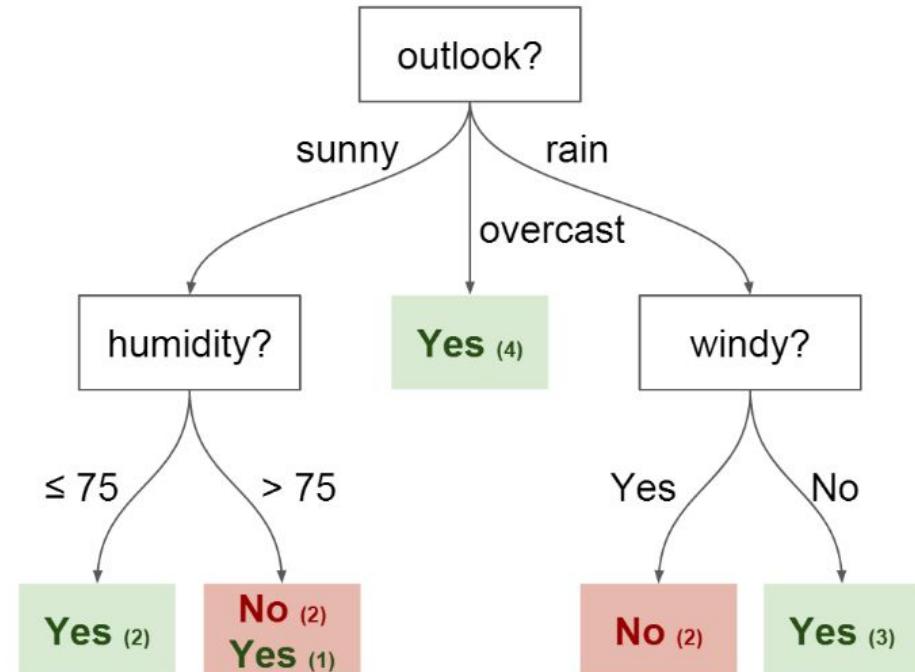
Temperature	Outlook	Humidity	Windy	Played?
Mild	Sunny	80	No	Yes
Hot	Sunny	75	Yes	No
Hot	Overcast	77	No	Yes
Cool	Rain	70	No	Yes
Cool	Overcast	72	Yes	Yes
Mild	Sunny	77	No	No
Cool	Sunny	70	No	Yes
Mild	Rain	69	No	Yes
Mild	Sunny	65	Yes	Yes
Mild	Overcast	77	Yes	Yes
Hot	Overcast	74	No	Yes
Mild	Rain	77	Yes	No
Cool	Rain	73	Yes	No
Mild	Rain	78	No	Yes



Tree Methods

I want to use this data to predict whether or not he will show up to play.

An intuitive way to do this is through a Decision Tree

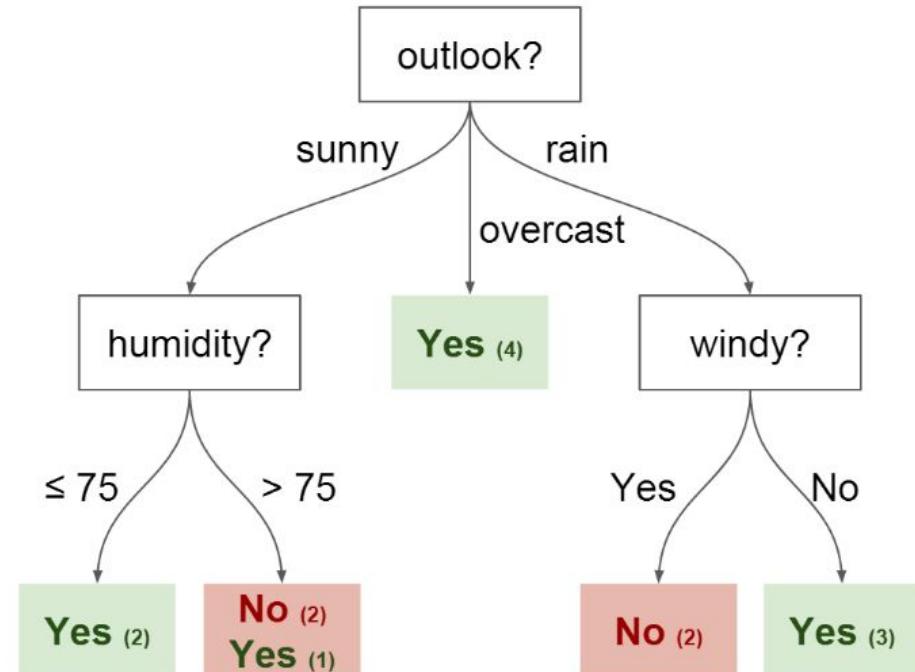




Tree Methods

In this tree we have:

- Nodes
 - Split for the value of a certain attribute
- Edges
 - Outcome of a split to next node

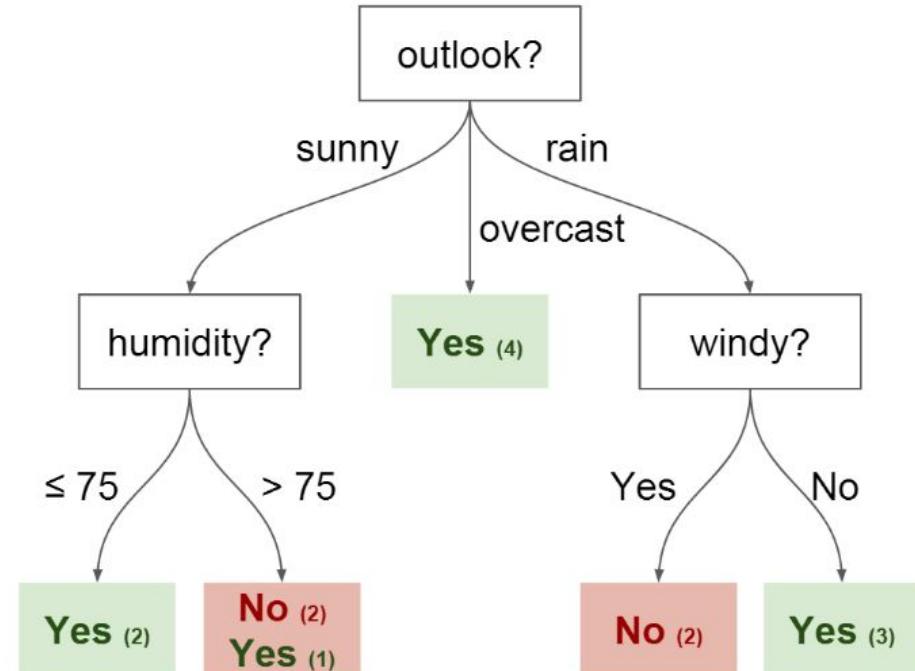




Tree Methods

In this tree we have:

- Root
 - The node that performs the first split
- Leaves
 - Terminal nodes that predict the outcome





Intuition Behind Splits

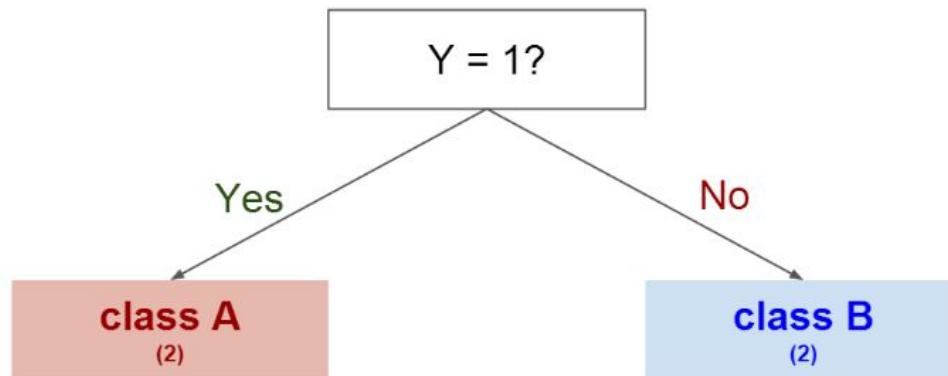
Imaginary Data with 3 features (X,Y, and Z) with two possible classes.

X	Y	Z	Class
1	1	1	A
1	1	0	A
0	0	1	B
1	0	0	B



Intuition Behind Splits

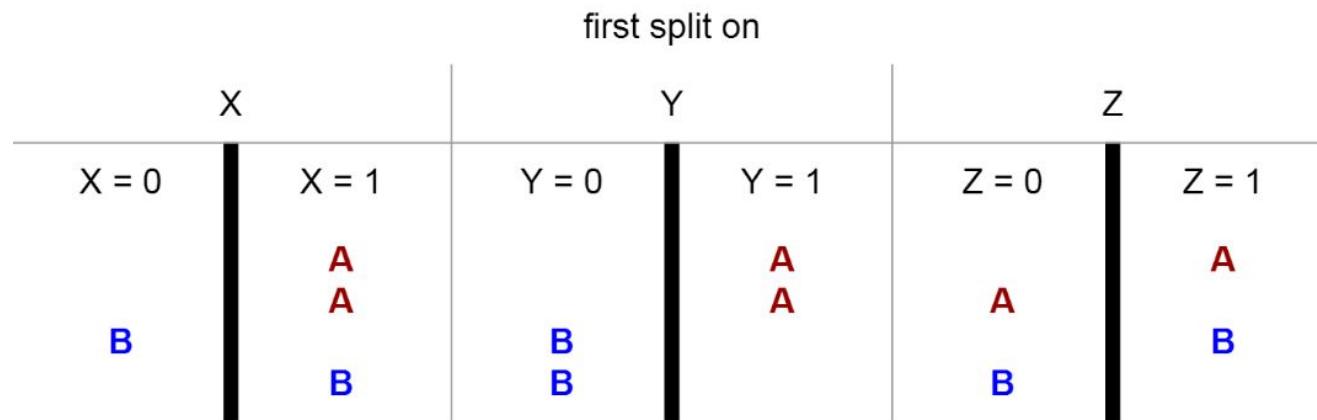
Splitting on Y gives us a clear separation between classes





Intuition Behind Splits

We could have also tried splitting on other features first:





Intuition Behind Splits

Entropy and Information Gain are the Mathematical Methods of choosing the best split. Refer to reading assignment.

Entropy:

$$H(S) = - \sum_i p_i(S) \log_2 p_i(S)$$

Information Gain:

$$IG(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{S} H(S_v)$$



Random Forests

To improve performance, we can use many trees with a random sample of features chosen as the split.

- A new random sample of features is chosen for **every single tree at every single split.**
- For **classification**, m is typically chosen to be the square root of p .



Random Forests

What's the point?

- Suppose there is **one very strong feature** in the data set. When using “bagged” trees, most of the trees will use that feature as the top split, resulting in an ensemble of similar trees that are **highly correlated**.



Random Forests

What's the point?

- Averaging highly correlated quantities does not significantly reduce variance.
- By randomly leaving out candidate features from each split, **Random Forests "decorrelates" the trees**, such that the averaging process can reduce the variance of the resulting model.

DECESION TREES AND RANDOM FOREST

```
In [1]: #Importing all the required packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [2]: #Reading the data
data=pd.read_csv("kyphosis.csv")
```

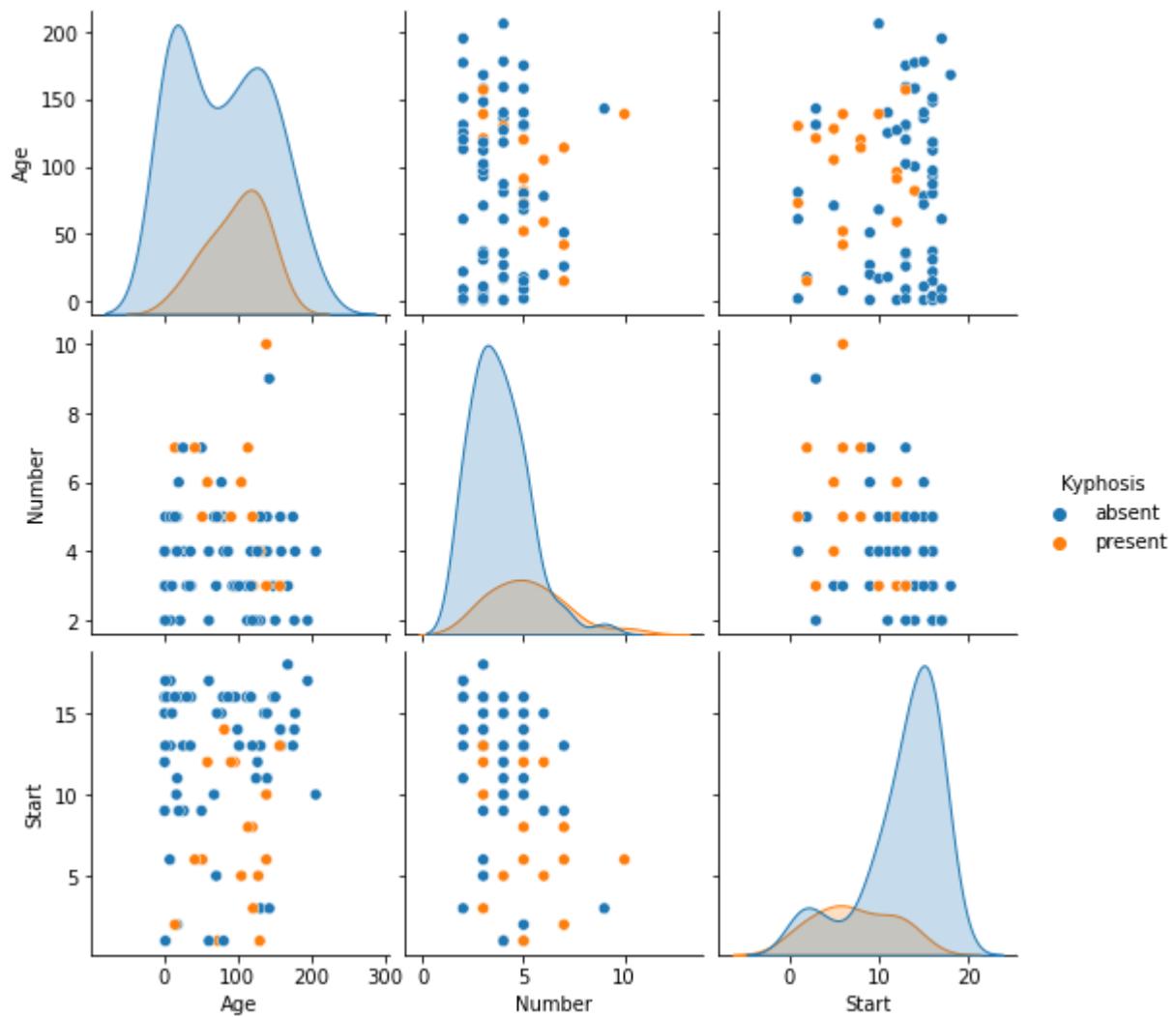
```
In [3]: data.head()
```

```
Out[3]:
```

	Kyphosis	Age	Number	Start
0	absent	71	3	5
1	absent	158	3	14
2	present	128	4	5
3	absent	2	5	1
4	absent	1	4	15

```
In [6]: #Creating an pairplot
sns.pairplot(data,hue="Kyphosis")
```

```
Out[6]: <seaborn.axisgrid.PairGrid at 0x2dab31d28b0>
```



```
In [8]: #Dividing the data into X and y where X are features and y are target
X=data.drop(["Kyphosis"],axis=1)
y=data["Kyphosis"]
```

```
In [9]: from sklearn.model_selection import train_test_split
```

```
In [10]: #Splitting the package
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3)
```

Decision Tree

```
In [28]: from sklearn.tree import DecisionTreeClassifier
```

```
In [12]: #Creating an model
dt=DecisionTreeClassifier()
```

```
In [13]: #Fitting the model into the data
dt.fit(X_train,y_train)
```

```
Out[13]: DecisionTreeClassifier()
```

```
In [14]: #Predicting the values
prediction=dt.predict(X_test)
```

```
In [15]: from sklearn.metrics import classification_report,confusion_matrix
```

```
In [17]: print(classification_report(y_test,prediction))
print("\n")
print(confusion_matrix(prediction,y_test))
```

	precision	recall	f1-score	support
absent	0.81	0.65	0.72	20
present	0.22	0.40	0.29	5
accuracy			0.60	25
macro avg	0.52	0.53	0.50	25
weighted avg	0.69	0.60	0.63	25

```
[[13  3]
 [ 7  2]]
```

Random Forest

```
In [19]: from sklearn.ensemble import RandomForestClassifier
```

```
In [24]: #Creating an model for prediction
rf=RandomForestClassifier(n_estimators=100)
```

```
In [25]: rf.fit(X_train,y_train)
```

```
Out[25]: RandomForestClassifier()
```

```
In [26]: prediction1=rf.predict(X_test)
```

```
In [27]: print(classification_report(y_test,prediction1))
print("\n")
print(confusion_matrix(y_test,prediction1))
```

	precision	recall	f1-score	support
absent	0.81	0.65	0.72	20
present	0.22	0.40	0.29	5
accuracy			0.60	25
macro avg	0.52	0.53	0.50	25
weighted avg	0.69	0.60	0.63	25

```
[[13  7]
 [ 3  2]]
```

```
In [ ]:
```

SUPPORT VECTOR MACHINES (SVM)

Support vector machines is used to find best line to separate the two plotted data in plots i.e finding best place to put the line in the plot. The line is called as Hyperplane

```
In [35]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Get the Data

We'll use the built in breast cancer dataset from Scikit Learn. We can get with the load function:

```
In [52]: #Getting the data from the datasets in the sklearn
from sklearn.datasets import load_breast_cancer
```

```
In [54]: #Loading the dataset in the variable called cancer
cancer = load_breast_cancer()
```

The data set is presented in a dictionary form:

```
In [55]: #Keys of the dataset
cancer.keys()
```

```
Out[55]: dict_keys(['DESCR', 'target', 'data', 'target_names', 'feature_names'])
```

We can grab information and arrays out of this dictionary to set up our data frame and understanding of the features:

```
In [4]: print(cancer['DESCR'])
```

```
Breast Cancer Wisconsin (Diagnostic) Database
```

```
Notes
```

```
-----
```

```
Data Set Characteristics:
```

```
:Number of Instances: 569
```

```
:Number of Attributes: 30 numeric, predictive attributes and the class
```

```
:Attribute Information:
```

- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness (perimeter² / area - 1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

The mean, standard error, and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

- class:
 - WDBC-Malignant
 - WDBC-Benign

:Summary Statistics:

	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54
perimeter (worst):	50.41	251.2
area (worst):	185.2	4254.0
smoothness (worst):	0.071	0.223
compactness (worst):	0.027	1.058
concavity (worst):	0.0	1.252
concave points (worst):	0.0	0.291
symmetry (worst):	0.156	0.664
fractal dimension (worst):	0.055	0.208

:Missing Attribute Values: None

:Class Distribution: 212 - Malignant, 357 - Benign

:Creator: Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

:Donor: Nick Street

:Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
<https://goo.gl/U2Uwz2>

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. A few of the images can be found at
<http://www.cs.wisc.edu/~street/images/>

Separating plane described above was obtained using Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree Construction Via Linear Programming." Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society,

pp. 97-101, 1992], a classification method which uses linear programming to construct a decision tree. Relevant features were selected using an exhaustive search in the space of 1-4 features and 1-3 separating planes.

The actual linear program used to obtain the separating plane in the 3-dimensional space is that described in:
 [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

```
ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/
```

References

- W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on Electronic Imaging: Science and Technology, volume 1905, pages 861-870, San Jose, CA, 1993.
- O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. Operations Research, 43(4), pages 570-577, July-August 1995.
- W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) 163-171.

```
In [56]: #List of features in the dataset
cancer['feature_names']
```

```
Out[56]: array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension error',
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimension'],
      dtype='|<U23')
```

Set up DataFrame

```
In [12]: #Creating an dataframe by taking the only the data in the dataset and naming the column
df_feat = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
df_feat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
mean radius           569 non-null float64
mean texture          569 non-null float64
mean perimeter        569 non-null float64
mean area              569 non-null float64
mean smoothness        569 non-null float64
mean compactness       569 non-null float64
mean concavity         569 non-null float64
mean concave points   569 non-null float64
mean symmetry          569 non-null float64
mean fractal dimension 569 non-null float64
```

```
radius error          569 non-null float64
texture error         569 non-null float64
perimeter error       569 non-null float64
area error            569 non-null float64
smoothness error      569 non-null float64
compactness error     569 non-null float64
concavity error        569 non-null float64
concave points error   569 non-null float64
symmetry error         569 non-null float64
fractal dimension error 569 non-null float64
worst radius           569 non-null float64
worst texture          569 non-null float64
worst perimeter         569 non-null float64
worst area              569 non-null float64
worst smoothness         569 non-null float64
worst compactness        569 non-null float64
worst concavity          569 non-null float64
worst concave points     569 non-null float64
worst symmetry           569 non-null float64
worst fractal dimension    569 non-null float64
dtypes: float64(30)
memory usage: 133.4 KB
```

```
In [14]: #List of targets in the datasets  
cancer['target']
```

```
In [32]: #Creating an df target  
df_target = pd.DataFrame(cancer[ 'target' ],columns=['Cancer'])
```

Now let's actually check out the dataframe!

In [8]: df.head()

```
Out[8]:    mean      mean  
radius  texture  perimeter  area  smoothness  compactness  concavity  concavity  concave  
         points  symmetry  fr  
         diment
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fr dimen
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.05
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05

5 rows × 30 columns

Exploratory Data Analysis

We'll skip the Data Viz part for this lecture since there are so many features that are hard to interpret if you don't have domain knowledge of cancer or tumor cells. In your project you will have more to visualize for the data.

Train Test Split

```
In [57]: from sklearn.model_selection import train_test_split
```

```
In [58]: #Instead of X and y variable we used df_feat,df_target respectively
X_train, X_test, y_train, y_test = train_test_split(df_feat,df_target, test_size=0.30,
```

Train the Support Vector Classifier

```
In [59]: #For support vector classifier we take it from svm module as svc
from sklearn.svm import SVC
```

```
In [60]: #creating an model svc
model = SVC()
```

```
In [61]: #Fitting the model into the data
model.fit(X_train,y_train)
```

```
Out[61]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
      decision_function_shape='ovo', degree=3, gamma='auto', kernel='rbf',
      max_iter=-1, probability=False, random_state=None, shrinking=True,
      tol=0.001, verbose=False)
```

Predictions and Evaluations

Now let's predict using the trained model.

```
In [27]: #Predicting the value of the models
predictions = model.predict(X_test)
```

```
In [45]: from sklearn.metrics import classification_report,confusion_matrix
```

```
In [46]: #Printing the confusion matrix reports
print(confusion_matrix(y_test,predictions))
```

```
[[ 0  66]
 [ 0 105]]
```

```
In [62]: #Printing the classification report
print(classification_report(y_test,predictions))
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	66
1	0.61	1.00	0.76	105
avg / total	0.38	0.61	0.47	171

/Users/marci/anaconda/lib/python3.5/site-packages/sklearn/metrics/classification.py:107
4: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples.

'precision', 'predicted', average, warn_for)

Woah! Notice that we are classifying everything into a single class! This means our model needs to have its parameters adjusted (it may also help to normalize the data).

We can search for parameters using a GridSearch!

```
In [ ]: #Here er noticed that the 0 class has 0 prediction due to no availability of the 0 precision
#To solve it we use grid search
```

Gridsearch

Finding the right parameters (like what C or gamma values to use) is a tricky task! But luckily, we can be a little lazy and just try a bunch of combinations and see what works best! This idea of creating a 'grid' of parameters and just trying out all the possible combinations is called a GridSearch, this method is common enough that Scikit-learn has this functionality built in with GridSearchCV! The CV stands for cross-validation which is the

GridSearchCV takes a dictionary that describes the parameters that should be tried and a model to train. The grid of parameters is defined as a dictionary, where the keys are the parameters and the values are the settings to be tested.

```
In [63]: #In grid search the C and gamma values are important for the prediction
#So we create some parameters by giving some C and gamma values which check with each other
param_grid = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001], 'kernel':
```

```
In [64]: #Importing GridSearchCV form model selection
from sklearn.model_selection import GridSearchCV
```

One of the great things about GridSearchCV is that it is a meta-estimator. It takes an estimator like SVC, and creates a new estimator, that behaves exactly the same - in this case, like a classifier. You should add refit=True and choose verbose to whatever number you want, higher the number, the more verbose (verbose just means the text output describing the process).

```
In [65]: #In GridSearchCV we give some parameters  
#Here verbose is important the higher the number more verbose taken  
#Its work is to [c]ontrols the verbosity: the higher, the more messages  
#Making refit is true that helps to refit the models in the datas  
#Now the grid is taken as model after making some changes  
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=3)
```

What fit does is a bit more involved than usual. First, it runs the same loop with cross-validation, to find the best parameter combination. Once it has the best combination, it runs fit again on all data passed to fit (without cross-validation), to build a single new model using the best parameter setting.

```
In [40]: # May take awhile!
#Now it compares all the details given as paramets in the param_grid
grid.fit(X_train,y_train)
```

Fitting 3 folds for each of 25 candidates, totalling 75 fits

```
[CV] gamma=1, C=0.1, kernel=rbf .....  
[CV] ..... gamma=1, C=0.1, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=1, C=0.1, kernel=rbf .....  
[CV] ..... gamma=1, C=0.1, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=1, C=0.1, kernel=rbf .....  
[CV] ..... gamma=1, C=0.1, kernel=rbf, score=0.636364 - 0.0s  
[CV] gamma=0.1, C=0.1, kernel=rbf .....  
[CV] ..... gamma=0.1, C=0.1, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=0.1, C=0.1, kernel=rbf .....  
[CV] ..... gamma=0.1, C=0.1, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=0.1, C=0.1, kernel=rbf .....  
[CV] ..... gamma=0.1, C=0.1, kernel=rbf, score=0.636364 - 0.0s  
[CV] gamma=0.01, C=0.1, kernel=rbf .....  
[CV] ..... gamma=0.01, C=0.1, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=0.01, C=0.1, kernel=rbf .....  
[CV] ..... gamma=0.01, C=0.1, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=0.01, C=0.1, kernel=rbf .....  
[CV] ..... gamma=0.01, C=0.1, kernel=rbf, score=0.636364 - 0.0s  
[CV] gamma=0.001, C=0.1, kernel=rbf .....  
[CV] ..... gamma=0.001, C=0.1, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=0.001, C=0.1, kernel=rbf .....  
[CV] ..... gamma=0.001, C=0.1, kernel=rbf, score=0.636364 - 0.0s  
[CV] gamma=0.0001, C=0.1, kernel=rbf .....  
[CV] ..... gamma=0.0001, C=0.1, kernel=rbf, score=0.902256 - 0.0s  
[CV] gamma=0.0001, C=0.1, kernel=rbf .....  
[CV] ..... gamma=0.0001, C=0.1, kernel=rbf, score=0.962406 - 0.0s  
[CV] gamma=0.0001, C=0.1, kernel=rbf .....  
[CV] ..... gamma=0.0001, C=0.1, kernel=rbf, score=0.916667 - 0.0s  
[CV] gamma=1, C=1, kernel=rbf .....  
[CV] ..... gamma=1, C=1, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=1, C=1, kernel=rbf .....  
[CV] ..... gamma=1, C=1, kernel=rbf, score=0.636364 - 0.0s  
[CV] gamma=0.1, C=1, kernel=rbf .....  
[CV] ..... gamma=0.1, C=1, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=0.1, C=1, kernel=rbf .....  
[CV] ..... gamma=0.1, C=1, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=0.1, C=1, kernel=rbf .....  
[CV] ..... gamma=0.1, C=1, kernel=rbf, score=0.636364 - 0.0s  
[CV] gamma=0.01, C=1, kernel=rbf .....  
[CV] ..... gamma=0.01, C=1, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=0.01, C=1, kernel=rbf .....  
[CV] ..... gamma=0.01, C=1, kernel=rbf, score=0.631579 - 0.0s
```

```
[CV] gamma=0.01, C=1, kernel=rbf .....  
[CV] ..... gamma=0.01, C=1, kernel=rbf, score=0.636364 - 0.0s  
[CV] gamma=0.001, C=1, kernel=rbf .....  
[CV] ..... gamma=0.001, C=1, kernel=rbf, score=0.902256 - 0.0s  
[CV] gamma=0.001, C=1, kernel=rbf .....  
[CV] ..... gamma=0.001, C=1, kernel=rbf, score=0.939850 - 0.0s  
[CV] gamma=0.001, C=1, kernel=rbf .....  
[CV] ..... gamma=0.001, C=1, kernel=rbf, score=0.954545 - 0.0s  
[CV] gamma=0.0001, C=1, kernel=rbf .....  
[CV] ..... gamma=0.0001, C=1, kernel=rbf, score=0.939850 - 0.0s  
[CV] gamma=0.0001, C=1, kernel=rbf .....  
[CV] ..... gamma=0.0001, C=1, kernel=rbf, score=0.969925 - 0.0s  
[CV] gamma=0.0001, C=1, kernel=rbf .....  
[CV] ..... gamma=0.0001, C=1, kernel=rbf, score=0.946970 - 0.0s  
[CV] gamma=1, C=10, kernel=rbf .....  
[CV] ..... gamma=1, C=10, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=1, C=10, kernel=rbf .....  
[CV] ..... gamma=1, C=10, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=1, C=10, kernel=rbf .....  
[CV] ..... gamma=1, C=10, kernel=rbf, score=0.636364 - 0.0s  
[CV] gamma=0.1, C=10, kernel=rbf .....  
[CV] ..... gamma=0.1, C=10, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=0.1, C=10, kernel=rbf .....  
[CV] ..... gamma=0.1, C=10, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=0.1, C=10, kernel=rbf .....  
[CV] ..... gamma=0.1, C=10, kernel=rbf, score=0.636364 - 0.0s  
[CV] gamma=0.01, C=10, kernel=rbf .....  
[CV] ..... gamma=0.01, C=10, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=0.01, C=10, kernel=rbf .....  
[CV] ..... gamma=0.01, C=10, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=0.01, C=10, kernel=rbf .....  
[CV] ..... gamma=0.01, C=10, kernel=rbf, score=0.636364 - 0.0s  
[CV] gamma=0.001, C=10, kernel=rbf .....  
[CV] ..... gamma=0.001, C=10, kernel=rbf, score=0.894737 - 0.0s  
[CV] gamma=0.001, C=10, kernel=rbf .....  
[CV] ..... gamma=0.001, C=10, kernel=rbf, score=0.932331 - 0.0s  
[CV] gamma=0.001, C=10, kernel=rbf .....  
[CV] ..... gamma=0.001, C=10, kernel=rbf, score=0.916667 - 0.0s  
[CV] gamma=0.0001, C=10, kernel=rbf .....  
[CV] ..... gamma=0.0001, C=10, kernel=rbf, score=0.932331 - 0.0s  
[CV] gamma=0.0001, C=10, kernel=rbf .....  
[CV] ..... gamma=0.0001, C=10, kernel=rbf, score=0.969925 - 0.0s  
[CV] gamma=0.0001, C=10, kernel=rbf .....  
[CV] ..... gamma=0.0001, C=10, kernel=rbf, score=0.962121 - 0.0s  
[CV] gamma=1, C=100, kernel=rbf .....  
[CV] ..... gamma=1, C=100, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=1, C=100, kernel=rbf .....  
[CV] ..... gamma=1, C=100, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=1, C=100, kernel=rbf .....  
[CV] ..... gamma=1, C=100, kernel=rbf, score=0.636364 - 0.0s  
[CV] gamma=0.1, C=100, kernel=rbf .....  
[CV] ..... gamma=0.1, C=100, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=0.1, C=100, kernel=rbf .....  
[CV] ..... gamma=0.1, C=100, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=0.1, C=100, kernel=rbf .....  
[CV] ..... gamma=0.1, C=100, kernel=rbf, score=0.636364 - 0.0s  
[CV] gamma=0.01, C=100, kernel=rbf .....  
[CV] ..... gamma=0.01, C=100, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=0.01, C=100, kernel=rbf .....  
[CV] ..... gamma=0.01, C=100, kernel=rbf, score=0.631579 - 0.0s  
[CV] gamma=0.01, C=100, kernel=rbf .....  
[CV] ..... gamma=0.01, C=100, kernel=rbf, score=0.636364 - 0.0s  
[CV] gamma=0.001, C=100, kernel=rbf .....  
[CV] ..... gamma=0.001, C=100, kernel=rbf, score=0.894737 - 0.0s
```

```
[CV] ..... gamma=0.001, C=100, kernel=rbf, score=0.932331 - 0.0s
[CV] gamma=0.001, C=100, kernel=rbf .....
[CV] ..... gamma=0.001, C=100, kernel=rbf, score=0.916667 - 0.0s
[CV] gamma=0.0001, C=100, kernel=rbf .....
[CV] ..... gamma=0.0001, C=100, kernel=rbf, score=0.917293 - 0.0s
[CV] gamma=0.0001, C=100, kernel=rbf .....
[CV] ..... gamma=0.0001, C=100, kernel=rbf, score=0.977444 - 0.0s
[CV] gamma=0.0001, C=100, kernel=rbf .....
[CV] ..... gamma=0.0001, C=100, kernel=rbf, score=0.939394 - 0.0s
[CV] gamma=1, C=1000, kernel=rbf .....
[CV] ..... gamma=1, C=1000, kernel=rbf, score=0.631579 - 0.0s
[CV] gamma=1, C=1000, kernel=rbf .....
[CV] ..... gamma=1, C=1000, kernel=rbf, score=0.631579 - 0.0s
[CV] gamma=1, C=1000, kernel=rbf .....
[CV] ..... gamma=1, C=1000, kernel=rbf, score=0.636364 - 0.0s
[CV] gamma=0.1, C=1000, kernel=rbf .....
[CV] ..... gamma=0.1, C=1000, kernel=rbf, score=0.631579 - 0.0s
[CV] gamma=0.1, C=1000, kernel=rbf .....
[CV] ..... gamma=0.1, C=1000, kernel=rbf, score=0.631579 - 0.0s
[CV] gamma=0.1, C=1000, kernel=rbf .....
[CV] ..... gamma=0.1, C=1000, kernel=rbf, score=0.636364 - 0.0s
[CV] gamma=0.01, C=1000, kernel=rbf .....
[CV] ..... gamma=0.01, C=1000, kernel=rbf, score=0.631579 - 0.0s
[CV] gamma=0.01, C=1000, kernel=rbf .....
[CV] ..... gamma=0.01, C=1000, kernel=rbf, score=0.636364 - 0.0s
[CV] gamma=0.001, C=1000, kernel=rbf .....
[CV] ..... gamma=0.001, C=1000, kernel=rbf, score=0.894737 - 0.0s
[CV] gamma=0.001, C=1000, kernel=rbf .....
[CV] ..... gamma=0.001, C=1000, kernel=rbf, score=0.932331 - 0.0s
[CV] gamma=0.001, C=1000, kernel=rbf .....
[CV] ..... gamma=0.001, C=1000, kernel=rbf, score=0.916667 - 0.0s

[Parallel(n_jobs=1)]: Done 31 tasks | elapsed: 0.3s
[Parallel(n_jobs=1)]: Done 75 out of 75 | elapsed: 0.8s finished
[CV] gamma=0.0001, C=1000, kernel=rbf .....
[CV] ..... gamma=0.0001, C=1000, kernel=rbf, score=0.909774 - 0.0s
[CV] gamma=0.0001, C=1000, kernel=rbf .....
[CV] ..... gamma=0.0001, C=1000, kernel=rbf, score=0.969925 - 0.0s
[CV] gamma=0.0001, C=1000, kernel=rbf .....
[CV] ..... gamma=0.0001, C=1000, kernel=rbf, score=0.931818 - 0.0s
```

```
Out[40]: GridSearchCV(cv=None, error_score='raise',
          estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
          decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False),
          fit_params={}, iid=True, n_jobs=1,
          param_grid={'gamma': [1, 0.1, 0.01, 0.001, 0.0001], 'C': [0.1, 1, 10, 100, 1000],
          'kernel': ['rbf']},
          pre_dispatch='2*n_jobs', refit=True, scoring=None, verbose=3)
```

You can inspect the best parameters found by GridSearchCV in the `best_params` attribute, and the best estimator in the `best_estimator_` attribute:

```
In [41]: #From that we select best_params_
#We get all the details which is best
grid.best_params_
```

```
Out[41]: {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
```

```
In [ ]: grid.best_estimator_
```

Then you can re-run predictions on this grid object just like you would with a normal model.

```
In [48]: grid_predictions = grid.predict(X_test)
```

```
In [49]: print(confusion_matrix(y_test,grid_predictions))
```

```
[[ 60  6]
 [ 3 102]]
```

```
In [50]: print(classification_report(y_test,grid_predictions))
```

	precision	recall	f1-score	support
0	0.95	0.91	0.93	66
1	0.94	0.97	0.96	105
avg / total	0.95	0.95	0.95	171

```
In [33]: # Now we can get accurate prediction
```

In this process we are taking the model SVC() but we did not get the perfect prediction So we took a GridSearchCV which helps to get the best C and gamma values due to which we get accuracy prediction

```
In [ ]:
```

Example:2

```
In [34]: # The Iris Setosa
from IPython.display import Image
url = 'http://upload.wikimedia.org/wikipedia/commons/5/56/Kosaciec_szczecinkowaty_Iris_
Image(url,width=300, height=300)
```

Out[34]:



```
In [2]: # The Iris Versicolor
from IPython.display import Image
url = 'http://upload.wikimedia.org/wikipedia/commons/4/41/Iris_versicolor_3.jpg'
Image(url,width=300, height=300)
```

Out[2]:



```
In [3]: # The Iris Virginica
from IPython.display import Image
url = 'http://upload.wikimedia.org/wikipedia/commons/9/9f/Iris_virginica.jpg'
Image(url,width=300, height=300)
```

Out[3]:



The iris dataset contains measurements for 150 iris flowers from three different species.

The three classes in the Iris dataset:

Iris-setosa (n=50)
Iris-versicolor (n=50)
Iris-virginica (n=50)

The four features of the Iris dataset:

sepal length in cm
sepal width in cm
petal length in cm
petal width in cm

Get the data

Use seaborn to get the iris data by using: `iris = sns.load_dataset('iris')`

```
In [6]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [7]: iris=sns.load_dataset("iris")
```

Let's visualize the data and get you started!

Exploratory Data Analysis

Time to put your data viz skills to the test! Try to recreate the following plots, make sure to import the libraries you'll need!

Import some libraries you think you'll need.

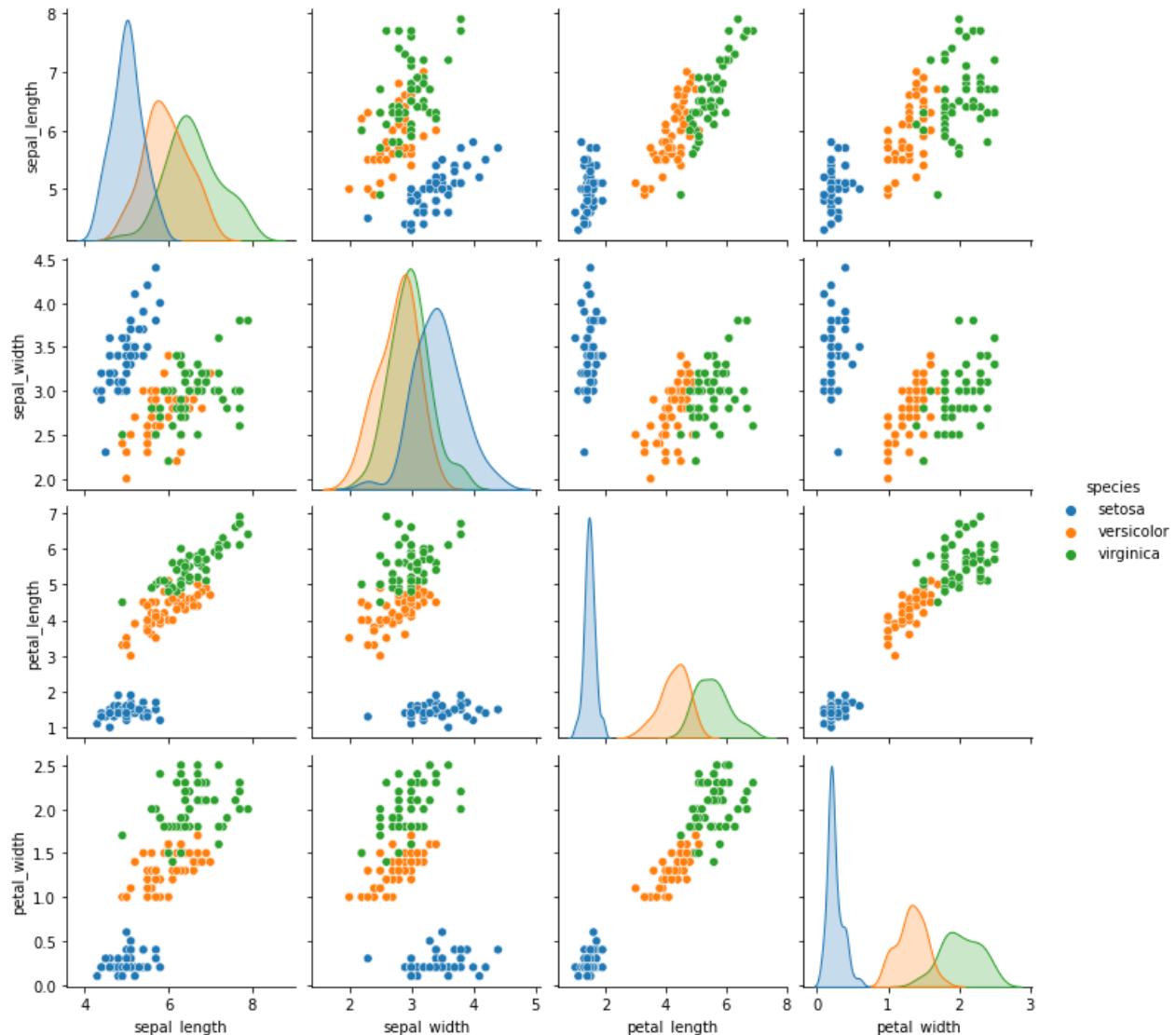
In [10]: `iris.head()`

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Create a pairplot of the data set. Which flower species seems to be the most separable?

In [12]: `sns.pairplot(hue="species", data=iris)`

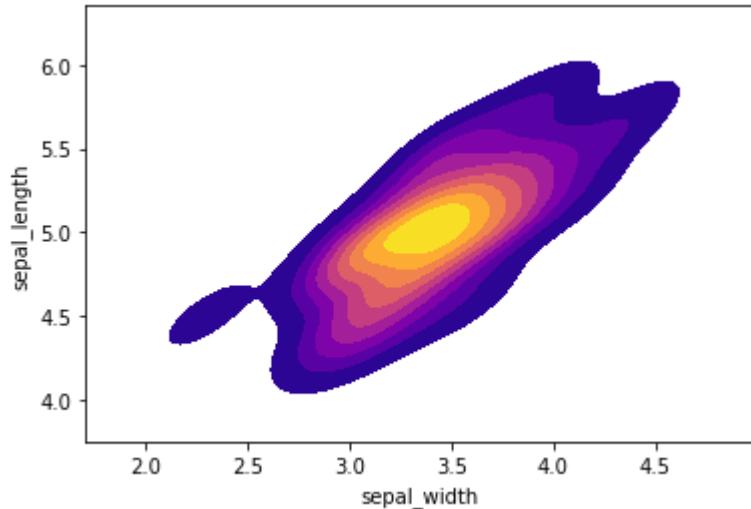
Out[12]: <seaborn.axisgrid.PairGrid at 0x164275012b0>



Create a kde plot of sepal_length versus sepal width for setosa species of flower.

```
In [13]: setosa = iris[iris['species']=='setosa']
sns.kdeplot( setosa['sepal_width'], setosa['sepal_length'],
cmap="plasma", shade=True, shade_lowest=False)
```

```
C:\Users\srena\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
    warnings.warn(
C:\Users\srena\anaconda3\lib\site-packages\seaborn\distributions.py:1678: UserWarning: `shade_lowest` is now deprecated in favor of `thresh`. Setting `thresh=0.05`, but please update your code.
    warnings.warn(msg, UserWarning)
Out[13]: <AxesSubplot:xlabel='sepal_width', ylabel='sepal_length'>
```



Train Test Split

Split your data into a training set and a testing set.

```
In [14]: X=iris.drop("species",axis=1)
y=iris["species"]
```

```
In [15]: from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.4)
```

Train a Model

Now its time to train a Support Vector Machine Classifier.

Call the SVC() model from sklearn and fit the model to the training data.

```
In [16]: from sklearn.svm import SVC
```

```
In [17]: vm=SVC()
```

```
In [18]: vm.fit(X_train,y_train)
```

```
Out[18]: SVC()
```

Model Evaluation

Now get predictions from the model and create a confusion matrix and a classification report.

```
In [19]: prediction=vm.predict(X_test)
```

```
In [21]: from sklearn.metrics import classification_report,confusion_matrix
```

```
In [22]: print(confusion_matrix(y_test,prediction))
```

```
[[16  0  0]
 [ 0 20  2]
 [ 0  0 22]]
```

```
In [23]: print(classification_report(y_test,prediction))
```

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	16
versicolor	1.00	0.91	0.95	22
virginica	0.92	1.00	0.96	22
accuracy			0.97	60
macro avg	0.97	0.97	0.97	60
weighted avg	0.97	0.97	0.97	60

Wow! You should have noticed that your model was pretty good! Let's see if we can tune the parameters to try to get even better (unlikely, and you probably would be satisfied with these results in real life because the data set is quite small, but I just want you to practice using GridSearch).

We would use this grid search if we did not get perfect prediction

Gridsearch Practice

Import GridsearchCV from SciKit Learn.

```
In [24]: from sklearn.model_selection import GridSearchCV
```

Create a dictionary called param_grid and fill out some parameters for C and gamma.

```
In [25]: param_grid={'C':[0.1,1,100,10,1000],'gamma':[1,0.1,0.01,0.001]}
```

Create a GridSearchCV object and fit it to the training data.

```
In [26]: grid=GridSearchCV(SVC(),param_grid,refit=True,verbose=3)
```

Now take that grid model and create some predictions using the test set and create classification reports and confusion matrices for them. Were you able to improve?

```
In [28]: grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[CV] C=0.1, gamma=1 .....[CV] ..... C=0.1, gamma=1, score=0.889, total= 0.0s
[CV] C=0.1, gamma=1 .....[CV] ..... C=0.1, gamma=1, score=0.944, total= 0.0s
```

```
[CV] C=0.1, gamma=1 ..... C=0.1, gamma=1, score=1.000, total= 0.0s
[CV] ..... C=0.1, gamma=1 ..... C=0.1, gamma=1, score=0.889, total= 0.0s
[CV] C=0.1, gamma=1 ..... C=0.1, gamma=1, score=1.000, total= 0.0s
[CV] ..... C=0.1, gamma=1 ..... C=0.1, gamma=1, score=1.000, total= 0.0s
[CV] C=0.1, gamma=0.1 ..... C=0.1, gamma=0.1, score=0.778, total= 0.0s
[CV] ..... C=0.1, gamma=0.1 ..... C=0.1, gamma=0.1, score=0.944, total= 0.0s
[CV] C=0.1, gamma=0.1 ..... C=0.1, gamma=0.1, score=0.889, total= 0.0s
[CV] C=0.1, gamma=0.1 ..... C=0.1, gamma=0.1, score=0.944, total= 0.0s
[CV] C=0.1, gamma=0.1 ..... C=0.1, gamma=0.1, score=0.889, total= 0.0s
[CV] C=0.1, gamma=0.01 ..... C=0.1, gamma=0.01, score=0.444, total= 0.0s
[CV] ..... C=0.1, gamma=0.01 ..... C=0.1, gamma=0.01, score=0.389, total= 0.0s
[CV] C=0.1, gamma=0.01 ..... C=0.1, gamma=0.01, score=0.333, total= 0.0s
[CV] C=0.1, gamma=0.01 ..... [Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1 out of 1 | elapsed: 0.0s remaining: 0.0s
[Parallel(n_jobs=1)]: Done 2 out of 2 | elapsed: 0.0s remaining: 0.0s
[CV] ..... C=0.1, gamma=0.01, score=0.389, total= 0.0s
[CV] C=0.1, gamma=0.01 ..... C=0.1, gamma=0.01, score=0.389, total= 0.0s
[CV] C=0.1, gamma=0.001 ..... C=0.1, gamma=0.001, score=0.389, total= 0.0s
[CV] ..... C=0.1, gamma=0.001, score=0.389, total= 0.0s
[CV] C=0.1, gamma=0.001 ..... C=0.1, gamma=0.001, score=0.389, total= 0.0s
[CV] ..... C=0.1, gamma=0.001, score=0.389, total= 0.0s
[CV] C=0.1, gamma=0.001 ..... C=0.1, gamma=0.001, score=0.389, total= 0.0s
[CV] ..... C=0.1, gamma=0.001, score=0.389, total= 0.0s
[CV] C=0.1, gamma=0.001 ..... C=0.1, gamma=0.001, score=0.389, total= 0.0s
[CV] ..... C=0.1, gamma=0.001, score=0.389, total= 0.0s
[CV] C=0.1, gamma=0.001 ..... C=0.1, gamma=0.001, score=0.389, total= 0.0s
[CV] ..... C=0.1, gamma=0.001, score=0.389, total= 0.0s
[CV] C=0.1, gamma=0.001 ..... C=0.1, gamma=0.001, score=0.389, total= 0.0s
[CV] ..... C=0.1, gamma=0.001, score=0.389, total= 0.0s
[CV] C=1, gamma=1 ..... C=1, gamma=1, score=1.000, total= 0.0s
[CV] ..... C=1, gamma=1 ..... C=1, gamma=1, score=0.944, total= 0.0s
[CV] C=1, gamma=1 ..... C=1, gamma=1, score=1.000, total= 0.0s
[CV] ..... C=1, gamma=1 ..... C=1, gamma=1, score=0.889, total= 0.0s
[CV] C=1, gamma=1 ..... C=1, gamma=1, score=1.000, total= 0.0s
[CV] ..... C=1, gamma=0.1 ..... C=1, gamma=0.1, score=0.944, total= 0.0s
[CV] C=1, gamma=0.1 ..... C=1, gamma=0.1, score=0.944, total= 0.0s
[CV] ..... C=1, gamma=0.1 ..... C=1, gamma=0.1, score=1.000, total= 0.0s
[CV] C=1, gamma=0.1 ..... C=1, gamma=0.1, score=0.944, total= 0.0s
[CV] ..... C=1, gamma=0.1 ..... C=1, gamma=0.1, score=1.000, total= 0.0s
[CV] C=1, gamma=0.1 ..... C=1, gamma=0.1, score=0.889, total= 0.0s
[CV] ..... C=1, gamma=0.1 ..... C=1, gamma=0.1, score=1.000, total= 0.0s
[CV] C=1, gamma=0.1 ..... C=1, gamma=0.1, score=0.889, total= 0.0s
[CV] ..... C=1, gamma=0.1 ..... C=1, gamma=0.1, score=1.000, total= 0.0s
[CV] C=1, gamma=0.01 ..... C=1, gamma=0.01, score=0.833, total= 0.0s
[CV] ..... C=1, gamma=0.01 ..... C=1, gamma=0.01, score=0.944, total= 0.0s
[CV] C=1, gamma=0.01 ..... C=1, gamma=0.01, score=0.889, total= 0.0s
[CV] ..... C=1, gamma=0.01 ..... C=1, gamma=0.01, score=0.944, total= 0.0s
```

```
[CV] C=1, gamma=0.01 .....  
[CV] ..... C=1, gamma=0.01, score=0.889, total= 0.0s  
[CV] C=1, gamma=0.01 .....  
[CV] ..... C=1, gamma=0.01, score=1.000, total= 0.0s  
[CV] C=1, gamma=0.001 .....  
[CV] ..... C=1, gamma=0.001, score=0.500, total= 0.0s  
[CV] C=1, gamma=0.001 .....  
[CV] ..... C=1, gamma=0.001, score=0.444, total= 0.0s  
[CV] C=1, gamma=0.001 .....  
[CV] ..... C=1, gamma=0.001, score=0.444, total= 0.0s  
[CV] C=1, gamma=0.001 .....  
[CV] ..... C=1, gamma=0.001, score=0.389, total= 0.0s  
[CV] C=1, gamma=0.001 .....  
[CV] ..... C=1, gamma=0.001, score=0.389, total= 0.0s  
[CV] C=100, gamma=1 .....  
[CV] ..... C=100, gamma=1, score=0.944, total= 0.0s  
[CV] C=100, gamma=1 .....  
[CV] ..... C=100, gamma=1, score=0.944, total= 0.0s  
[CV] C=100, gamma=1 .....  
[CV] ..... C=100, gamma=1, score=0.889, total= 0.0s  
[CV] C=100, gamma=1 .....  
[CV] ..... C=100, gamma=1, score=0.944, total= 0.0s  
[CV] C=100, gamma=1 .....  
[CV] ..... C=100, gamma=1, score=0.944, total= 0.0s  
[CV] C=100, gamma=1 .....  
[CV] ..... C=100, gamma=1, score=0.944, total= 0.0s  
[CV] C=100, gamma=0.1 .....  
[CV] ..... C=100, gamma=0.1, score=1.000, total= 0.0s  
[CV] C=100, gamma=0.1 .....  
[CV] ..... C=100, gamma=0.1, score=0.944, total= 0.0s  
[CV] C=100, gamma=0.1 .....  
[CV] ..... C=100, gamma=0.1, score=0.944, total= 0.0s  
[CV] C=100, gamma=0.1 .....  
[CV] ..... C=100, gamma=0.1, score=0.944, total= 0.0s  
[CV] C=100, gamma=0.1 .....  
[CV] ..... C=100, gamma=0.1, score=0.944, total= 0.0s  
[CV] C=100, gamma=0.01 .....  
[CV] ..... C=100, gamma=0.01, score=1.000, total= 0.0s  
[CV] C=100, gamma=0.01 .....  
[CV] ..... C=100, gamma=0.01, score=0.944, total= 0.0s  
[CV] C=100, gamma=0.01 .....  
[CV] ..... C=100, gamma=0.01, score=0.944, total= 0.0s  
[CV] C=100, gamma=0.01 .....  
[CV] ..... C=100, gamma=0.01, score=0.944, total= 0.0s  
[CV] C=100, gamma=0.01 .....  
[CV] ..... C=100, gamma=0.01, score=0.944, total= 0.0s  
[CV] C=100, gamma=0.01 .....  
[CV] ..... C=100, gamma=0.01, score=0.944, total= 0.0s  
[CV] C=100, gamma=0.01 .....  
[CV] ..... C=100, gamma=0.01, score=0.944, total= 0.0s  
[CV] C=100, gamma=0.01 .....  
[CV] ..... C=100, gamma=0.01, score=0.944, total= 0.0s  
[CV] C=10, gamma=1 .....  
[CV] ..... C=10, gamma=1, score=1.000, total= 0.0s  
[CV] C=10, gamma=1 .....  
[CV] ..... C=10, gamma=1, score=0.944, total= 0.0s  
[CV] C=10, gamma=1 .....  
[CV] ..... C=10, gamma=1, score=0.944, total= 0.0s  
[CV] C=10, gamma=1 .....  
[CV] ..... C=10, gamma=1, score=0.889, total= 0.0s  
[CV] C=10, gamma=1 .....  
[CV] ..... C=10, gamma=1, score=0.944, total= 0.0s  
[CV] C=10, gamma=0.1 .....  
[CV] ..... C=10, gamma=0.1, score=0.889, total= 0.0s
```



```
[CV] C=1000, gamma=0.001 .....  

[CV] ..... C=1000, gamma=0.001, score=0.944, total= 0.0s  

[CV] C=1000, gamma=0.001 .....  

[CV] ..... C=1000, gamma=0.001, score=1.000, total= 0.0s  

[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed: 0.8s finished  

Out[28]: GridSearchCV(estimator=SVC(),  

                      param_grid={'C': [0.1, 1, 100, 10, 1000],  

                                   'gamma': [1, 0.1, 0.01, 0.001]},  

                      verbose=3)
```

In [29]: `grid.best_params_`

Out[29]: `{'C': 100, 'gamma': 0.01}`

In [30]: `grid.best_estimator_`

Out[30]: `SVC(C=100, gamma=0.01)`

In [31]: `prediction=grid.predict(X_test)`

In [32]: `print(classification_report(y_test,prediction))`

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	16
versicolor	1.00	0.91	0.95	22
virginica	0.92	1.00	0.96	22
accuracy			0.97	60
macro avg	0.97	0.97	0.97	60
weighted avg	0.97	0.97	0.97	60

In [33]: `print(confusion_matrix(y_test,prediction))`

```
[[16  0  0]
 [ 0 20  2]
 [ 0  0 22]]
```

You should have done about the same or exactly the same, this makes sense, there is basically just one point that is too noisy to grab, which makes sense, we don't want to have an overfit model that would be able to grab that.



Introduction to K Means Clustering



Reading Assignment

Chapter 10 of
Introduction to Statistical Learning
By Gareth James, et al.



K Means Clustering

K Means Clustering is an unsupervised learning algorithm that will attempt to group similar clusters together in your data.

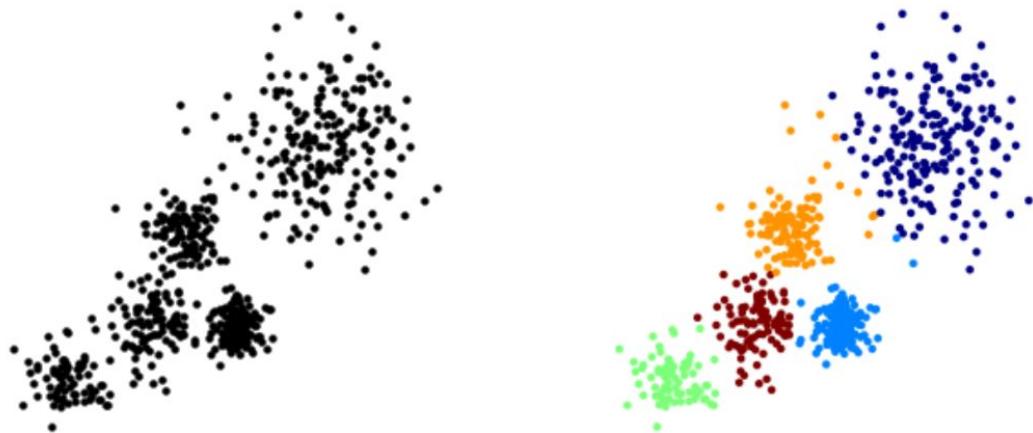
So what does a typical clustering problem look like?

- Cluster Similar Documents
- Cluster Customers based on Features
- Market Segmentation
- Identify similar physical groups



K Means Clustering

- The overall goal is to divide data into distinct groups such that observations within each group are similar





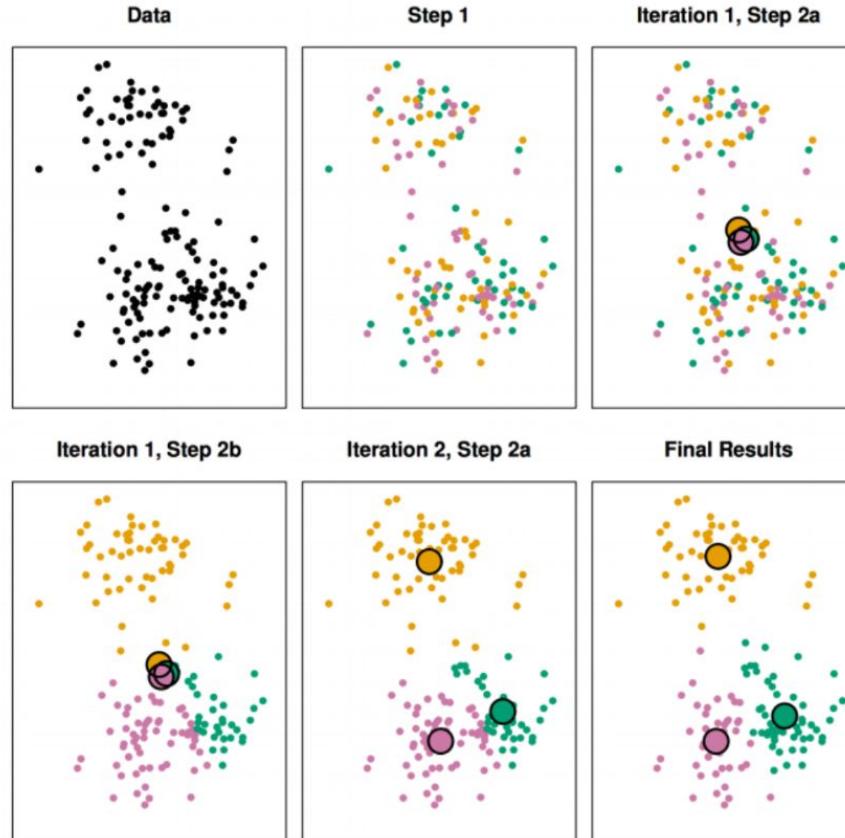
K Means Clustering

The K Means Algorithm

- Choose a number of Clusters “K”
- Randomly assign each point to a cluster
- Until clusters stop changing, repeat the following:
 - For each cluster, compute the cluster centroid by taking the mean vector of points in the cluster
 - Assign each data point to the cluster for which the centroid is the closest

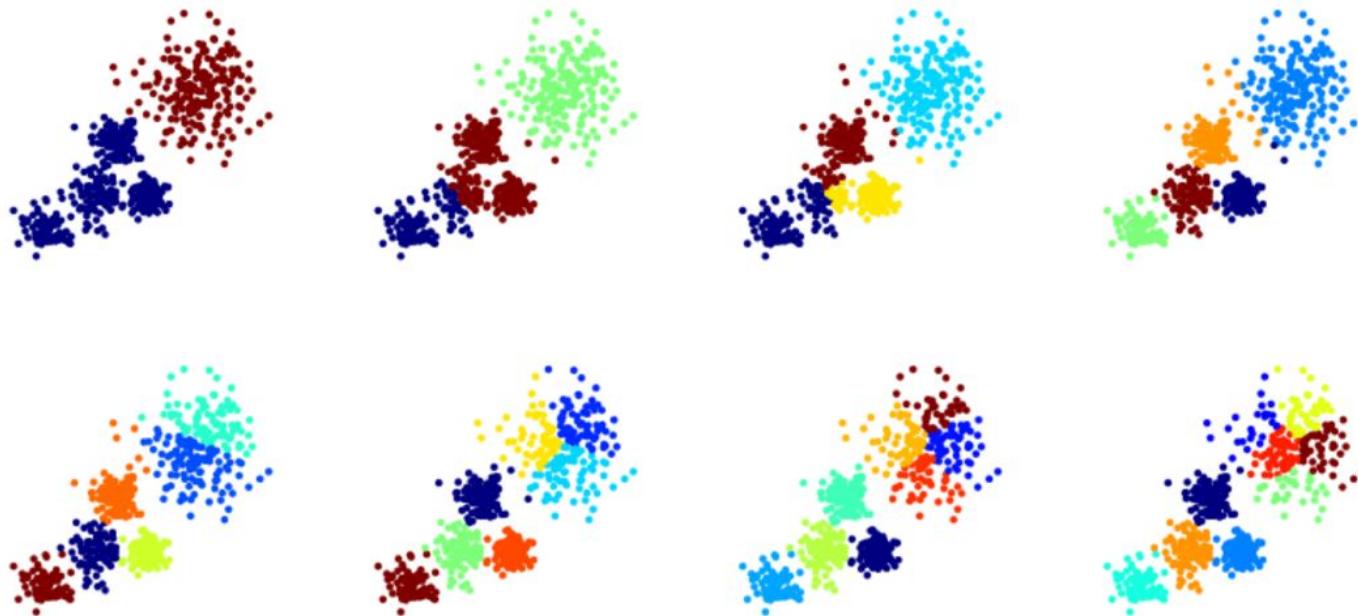


K Means Clustering





Choosing a K Value





Choosing a K Value

- There is no easy answer for choosing a “best” K value
- One way is the elbow method

First of all, compute the sum of squared error (SSE) for some values of k (for example 2, 4, 6, 8, etc.).

The SSE is defined as the sum of the squared distance between each member of the cluster and its centroid.



Choosing a K Value

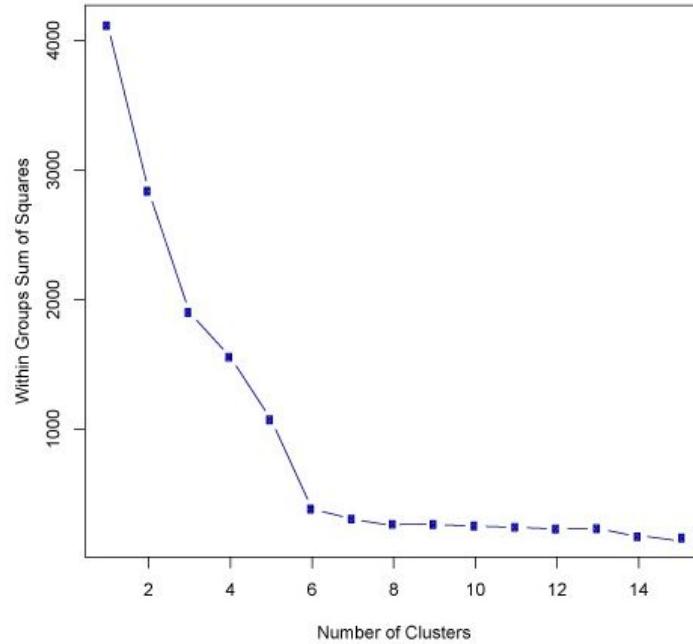
If you plot k against the SSE, you will see that *the error decreases as k gets larger*; this is because when the number of clusters increases, they should be smaller, so distortion is also smaller.

The idea of the elbow method is to choose the k at which the SSE decreases abruptly.

This produces an "elbow effect" in the graph, as you can see in the following picture:



Choosing a K Value



K Means Clustering with Python

Method Used

K Means Clustering is an unsupervised learning algorithm that tries to cluster data based on their similarity. Unsupervised learning means that there is no outcome to be predicted, and the algorithm just tries to find patterns in the data. In k means clustering, we have to specify the number of clusters we want the data to be grouped into. The algorithm randomly assigns each observation to a cluster, and finds the centroid of each cluster. Then, the algorithm iterates through two steps: Reassign data points to the cluster whose centroid is closest. Calculate new centroid of each cluster. These two steps are repeated till the within cluster variation cannot be reduced any further. The within cluster variation is calculated as the sum of the euclidean distance between the data points and their respective cluster centroids.

Simple concept is we have a data which is splitted into different colours and we predict some data with k means clustering So, it will predict and gives output and we compare the original and predicted plots and finalize

Import Libraries

```
In [22]: #Importing all required libraries
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Create some Data

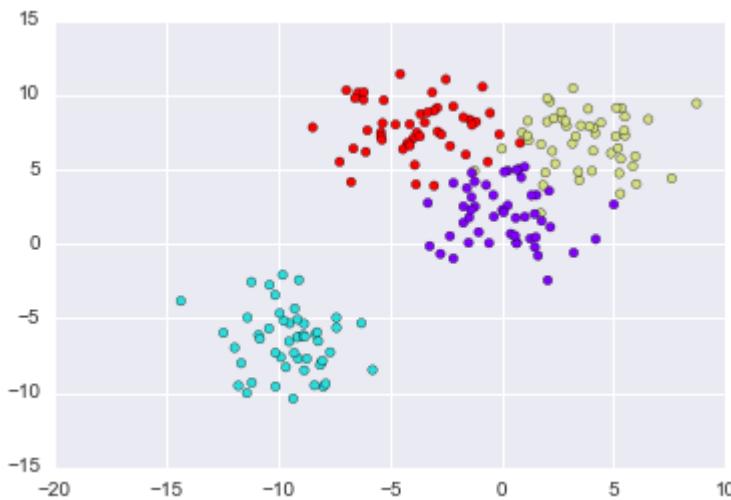
```
In [23]: #Importing the datas from sklearn
from sklearn.datasets import make_blobs
```

```
In [42]: # Create Data
data = make_blobs(n_samples=200, n_features=2,
                  centers=4, cluster_std=1.8, random_state=101)
```

Visualize Data

```
In [43]: #visualizing the data with 4 centers
plt.scatter(data[0][:,0],data[0][:,1],c=data[1],cmap='rainbow')
```

```
Out[43]: <matplotlib.collections.PathCollection at 0x11ab34d30>
```



Creating the Clusters

```
In [48]: from sklearn.cluster import KMeans
```

```
In [49]: #Creating the model with 4 clusters
kmeans = KMeans(n_clusters=4)
```

```
In [50]: #Fitting the first set of data for prediction
kmeans.fit(data[0])
```

```
Out[50]: KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=4, n_init=10,
n_jobs=1, precompute_distances='auto', random_state=None, tol=0.0001,
verbose=0)
```

```
In [51]: #Getting centers of the kmeans
kmeans.cluster_centers_
```

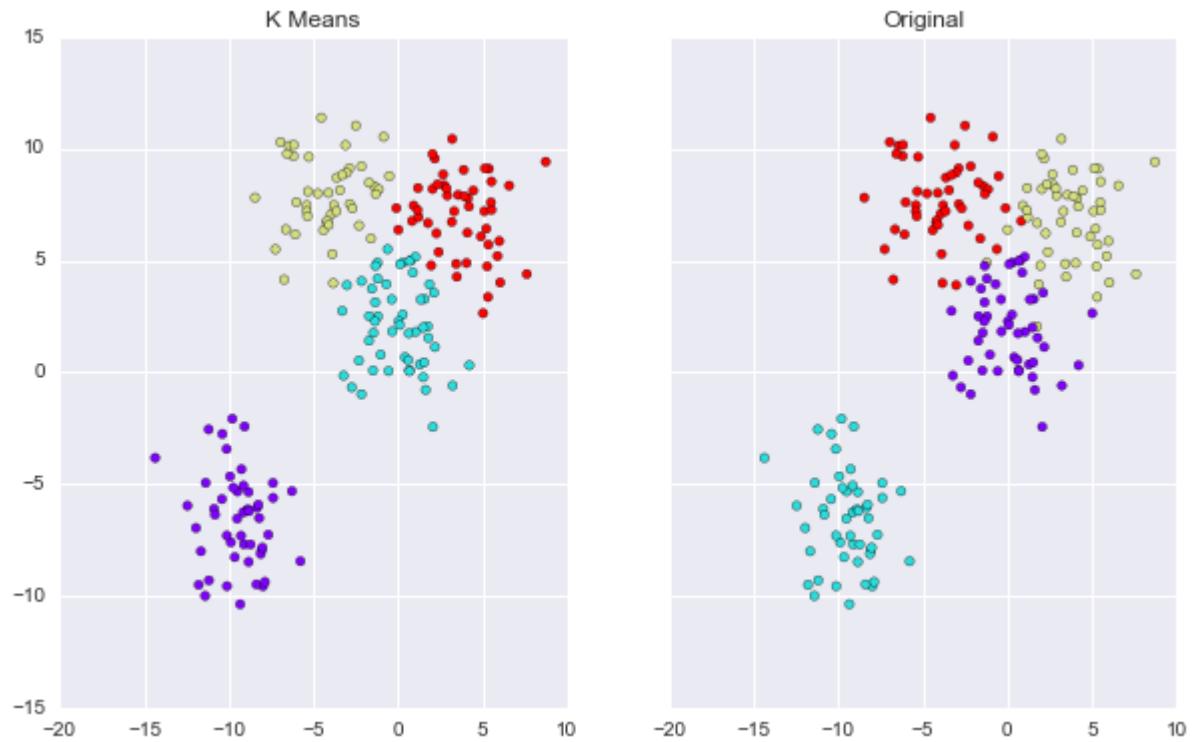
```
Out[51]: array([[-4.13591321,  7.95389851],
 [-9.46941837, -6.56081545],
 [-0.0123077 ,  2.13407664],
 [ 3.71749226,  7.01388735]])
```

```
In [55]: #Getting the labels
kmeans.labels_
```

```
Out[55]: array([2, 3, 1, 3, 3, 0, 3, 1, 3, 1, 2, 1, 3, 3, 2, 1, 3, 1, 0, 2, 0, 1, 1,
 0, 2, 0, 0, 1, 3, 3, 2, 0, 3, 1, 1, 2, 0, 0, 0, 1, 0, 2, 2, 2, 1, 3,
 2, 1, 0, 1, 1, 2, 3, 1, 0, 2, 1, 1, 2, 3, 0, 3, 0, 2, 3, 1, 0, 3, 3,
 0, 3, 1, 0, 1, 0, 3, 3, 1, 2, 1, 1, 0, 3, 0, 1, 1, 1, 2, 1, 0, 0, 0,
 0, 1, 1, 0, 3, 2, 0, 3, 1, 0, 1, 1, 3, 1, 0, 3, 0, 0, 3, 2, 2, 3, 0,
 3, 2, 2, 3, 2, 1, 2, 1, 2, 1, 3, 2, 1, 0, 2, 2, 2, 1, 0, 0, 2, 3, 2,
 3, 1, 0, 3, 0, 2, 2, 3, 1, 0, 2, 2, 2, 1, 3, 1, 2, 3, 3, 1, 1, 3,
 1, 1, 2, 0, 2, 1, 3, 2, 1, 3, 1, 2, 3, 1, 2, 3, 3, 0, 3, 2, 0, 0, 2,
 0, 0, 0, 0, 1, 0, 3, 3, 2, 0, 1, 3, 3, 0, 1], dtype=int32)
```

```
In [69]: #Finally comparing original datas ans k means clustering model predicted datas to know
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True, figsize=(10,6))
ax1.set_title('K Means')
ax1.scatter(data[0][:,0], data[0][:,1], c=kmeans.labels_, cmap='rainbow')
ax2.set_title("Original")
ax2.scatter(data[0][:,0], data[0][:,1], c=data[1], cmap='rainbow')
```

Out[69]: <matplotlib.collections.PathCollection at 0x11da01be0>



You should note, the colors are meaningless in reference between the two plots.



Principal Component Analysis



Background

Section 10.2 of
Introduction to Statistical Learning
By Gareth James, et al.



Background

- Let's discuss the basic idea behind principal component analysis.
- It is an unsupervised statistical technique used to examine the interrelations among a set of variables in order to identify the underlying structure of those variables.
- It is also known sometimes as a general **factor analysis**.



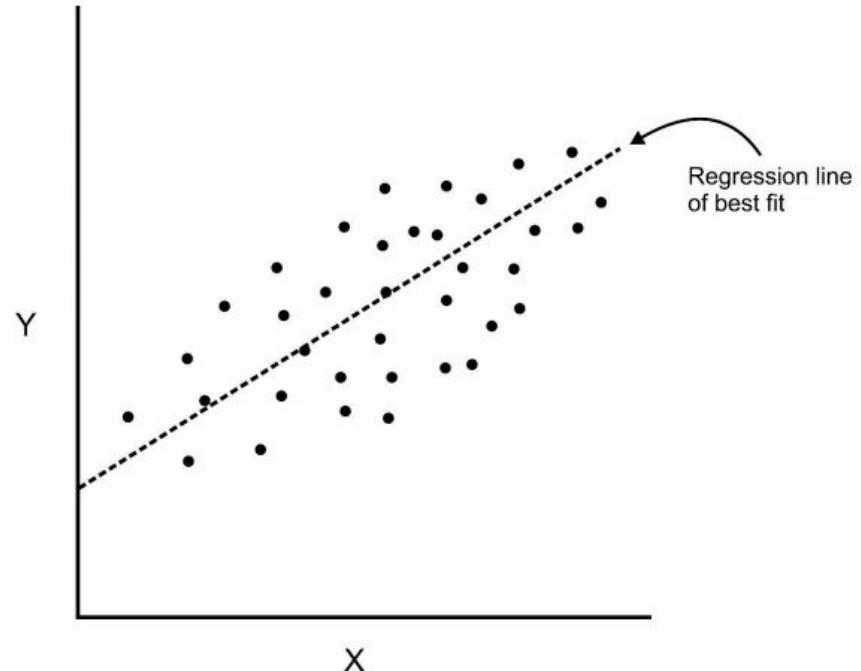
Background

- Where regression determines a line of best fit to a data set, factor analysis determines several orthogonal lines of best fit to the data set.
- Orthogonal means “at right angles”.
 - Actually the lines are perpendicular to each other in n -dimensional space.
- n -Dimensional Space is the variable sample space.
 - There are as many dimensions as there are variables, so in a data set with 4 variables the sample space is 4-dimensional.



Background

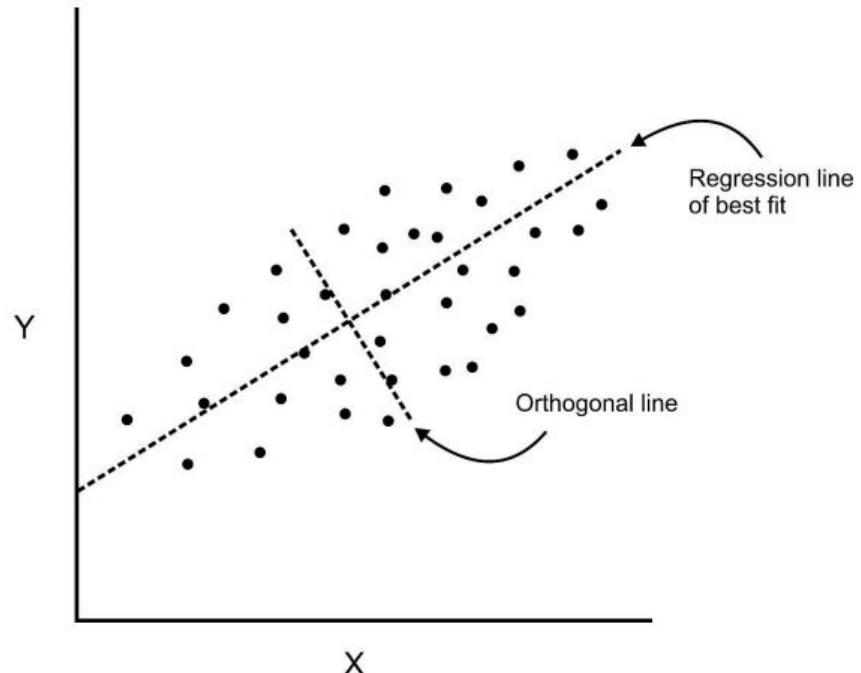
- Here we have some data plotted along two features, x and y.





Background

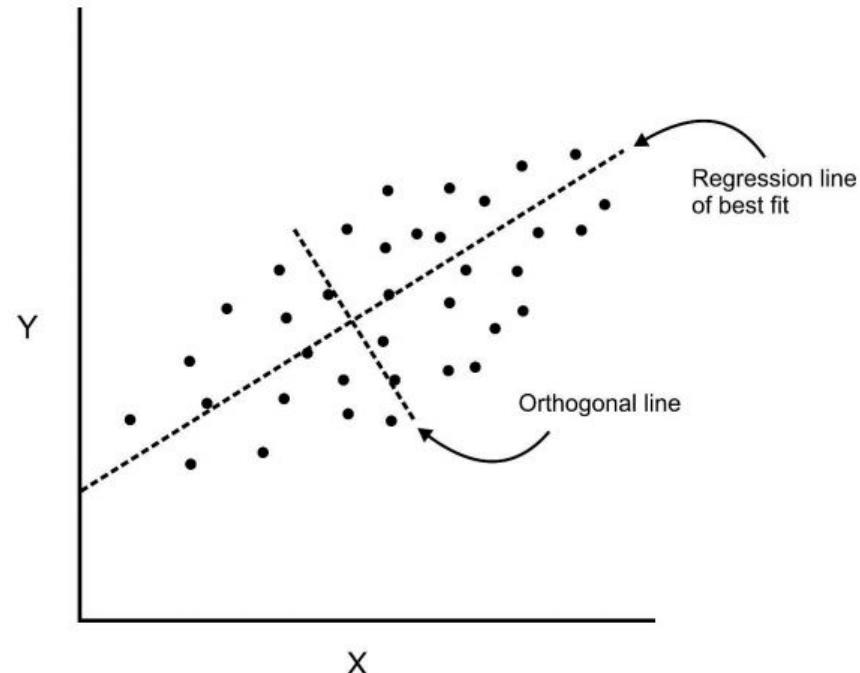
- We can add an orthogonal line.
- Now we can begin to understand the components!





Background

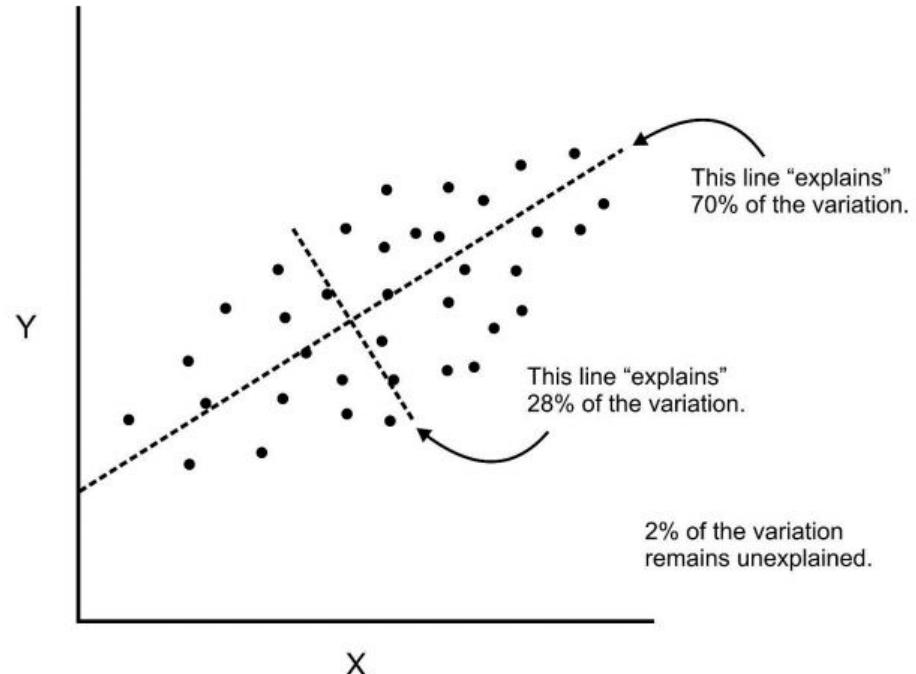
- Components are a linear transformation that chooses a variable system for the data set such that the greatest variance of the data set comes to lie on the first axis





Background

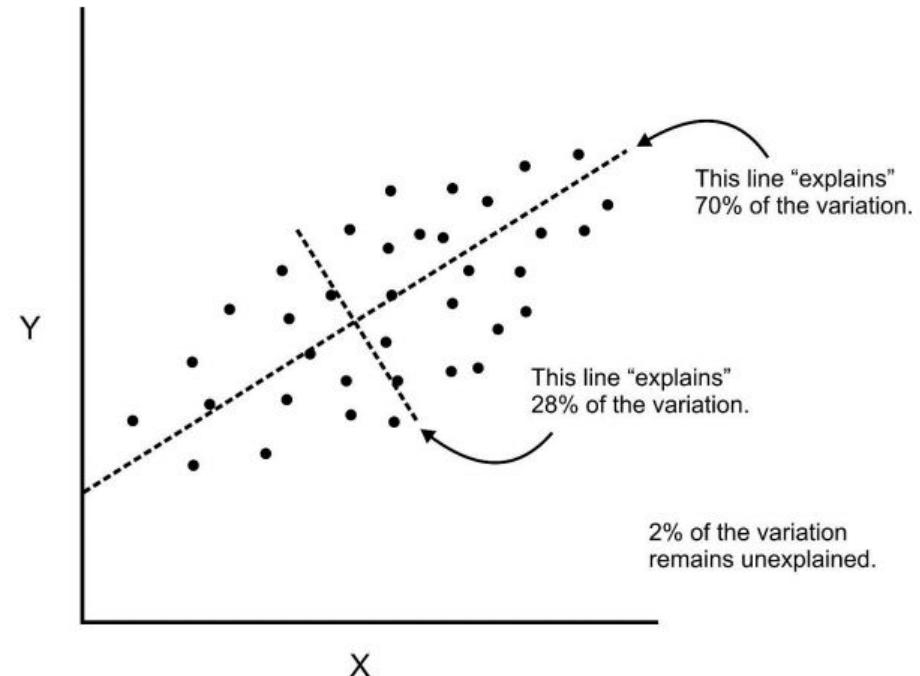
- The second greatest variance on the second axis, and so on ...
- This process allows us to reduce the number of variables used in an analysis.





Background

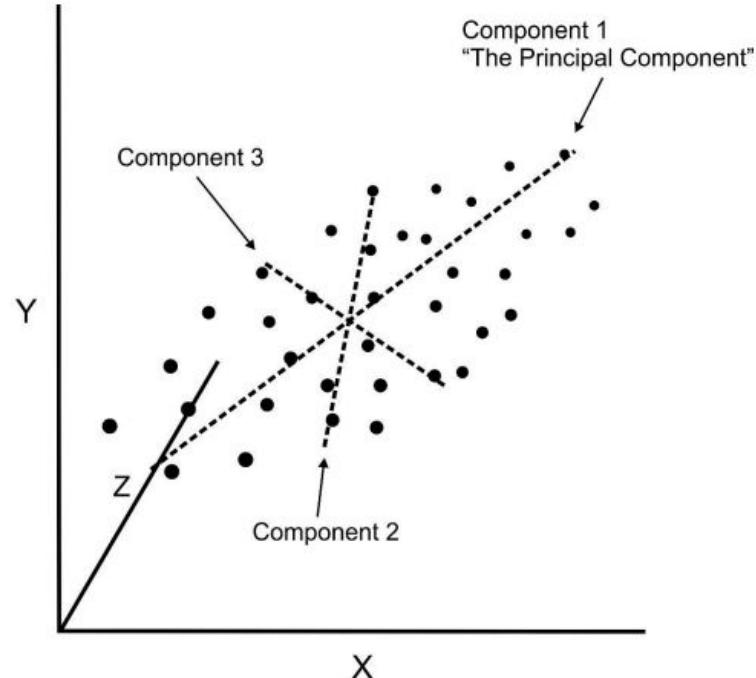
- Note that components are uncorrelated, since in the sample space they are orthogonal to each other.





Background

- We can continue this analysis into higher dimensions





Background

- If we use this technique on a data set with a large number of variables, we can compress the amount of explained variation to just a few components.
- The most challenging part of PCA is interpreting the components.



Background

- For our work with Python, we'll walk through an example of how to perform PCA with scikit learn.
- We usually want to standardize our data by some scale for PCA, so we'll cover how to do this as well.
- Since this algorithm is used usually for analysis of data and not a fully deployable model, there won't be a portfolio project for this topic.



Example with Python

Principal Component Analysis

Let's discuss PCA! Since this isn't exactly a full machine learning algorithm, but instead an unsupervised learning algorithm, we will just have a lecture on this topic, but no full machine learning project (although we will walk through the cancer set with PCA).

PCA Review

Make sure to watch the video lecture and theory presentation for a full overview of PCA! Remember that PCA is just a transformation of your data and attempts to find out what features explain the most variance in your data. For example:



Libraries

```
In [21]: #Importing all required modules
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
```

The Data

Let's work with the cancer data set again since it had so many features.

```
In [22]: #Importing the dataset from sklearn
from sklearn.datasets import load_breast_cancer
```

```
In [23]: cancer = load_breast_cancer()
```

```
In [24]: cancer.keys()
```

```
Out[24]: dict_keys(['DESCR', 'data', 'feature_names', 'target_names', 'target'])
```

```
In [25]: print(cancer['DESCR'])
```

Breast Cancer Wisconsin (Diagnostic) Database

Notes

Data Set Characteristics:

:Number of Instances: 569

:Number of Attributes: 30 numeric, predictive attributes and the class

:Attribute Information:

- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter

- area
- smoothness (local variation in radius lengths)
- compactness (perimeter² / area - 1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

The mean, standard error, and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

- class:
 - WDBC-Malignant
 - WDBC-Benign

:Summary Statistics:

	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54
perimeter (worst):	50.41	251.2
area (worst):	185.2	4254.0
smoothness (worst):	0.071	0.223
compactness (worst):	0.027	1.058
concavity (worst):	0.0	1.252
concave points (worst):	0.0	0.291
symmetry (worst):	0.156	0.664
fractal dimension (worst):	0.055	0.208

:Missing Attribute Values: None

:Class Distribution: 212 - Malignant, 357 - Benign

:Creator: Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

:Donor: Nick Street

:Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.

<https://goo.gl/U2Uwz2>

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. A few of the images can be found at <http://www.cs.wisc.edu/~street/images/>

Separating plane described above was obtained using Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree Construction Via Linear Programming." Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society, pp. 97-101, 1992], a classification method which uses linear programming to construct a decision tree. Relevant features were selected using an exhaustive search in the space of 1-4 features and 1-3 separating planes.

The actual linear program used to obtain the separating plane in the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

```
ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/
```

References

- W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on Electronic Imaging: Science and Technology, volume 1905, pages 861-870, San Jose, CA, 1993.
- O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. Operations Research, 43(4), pages 570-577, July-August 1995.
- W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) 163-171.

In [26]: `#Creating an dataframe of data
df = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
#['DESCR', 'data', 'feature_names', 'target_names', 'target']`

In [27]: `df.head()`

Out[27]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fr dimen
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.05
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05

5 rows × 30 columns

PCA Visualization

As we've noticed before it is difficult to visualize high dimensional data, we can use PCA to find the first two principal components, and visualize the data in this new, two-dimensional space, with a single scatter-plot. Before we do this though, we'll need to scale our data so that each feature has a single unit variance.

```
In [30]: #First we need to import the Standard Scale  
from sklearn.preprocessing import StandardScaler
```

```
In [32]: #Creating an model and fitting into the data  
scaler = StandardScaler()  
scaler.fit(df)
```

```
Out[32]: StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
In [33]: #The idea behind StandardScaler is that it will transform your data such that its  
#distribution will have a mean value 0 and standard deviation of 1. In case of multivar  
#this is done feature-wise (in other words independently for each column of the data).  
scaled_data = scaler.transform(df)
```

PCA with Scikit Learn uses a very similar process to other preprocessing functions that come with SciKit Learn. We instantiate a PCA object, find the principal components using the fit method, then apply the rotation and dimensionality reduction by calling transform().

We can also specify how many components we want to keep when creating the PCA object.

```
In [34]: from sklearn.decomposition import PCA
```

```
In [35]: #Creating an model with only main 2 columns  
pca = PCA(n_components=2)
```

```
In [36]: #Fitting into the data  
pca.fit(scaled_data)
```

```
Out[36]: PCA(copy=True, n_components=2, whiten=False)
```

Now we can transform this data to its first 2 principal components.

```
In [37]: #Transform the data  
x_pca = pca.transform(scaled_data)
```

```
In [38]: #Shape before PCA model  
scaled_data.shape
```

```
Out[38]: (569, 30)
```

```
In [39]: #Shape After PCA model  
x_pca.shape
```

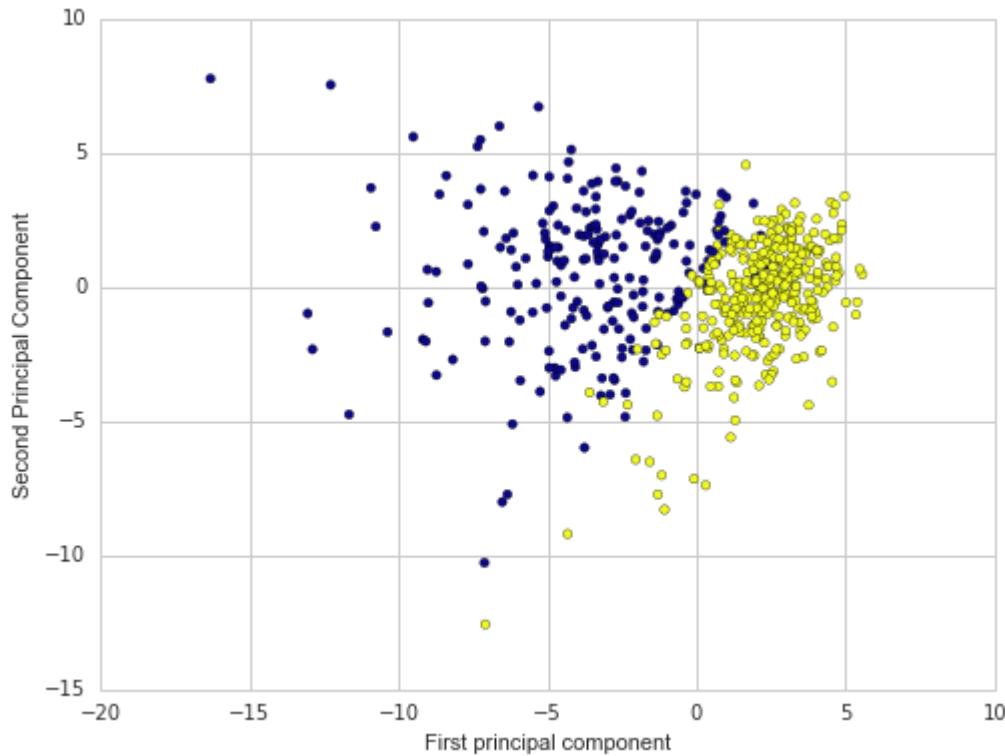
```
Out[39]: (569, 2)
```

In []: #As we See in this the columns before was 30 this pca comperess it into 2 columns for p

Great! We've reduced 30 dimensions to just 2! Let's plot these two dimensions out!

In [52]: #Printing the first and second principal components i,e :first and second columns respe
plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,1],c=cancer['target'],cmap='plasma')
plt.xlabel('First principal component')
plt.ylabel('Second Principal Component')

Out[52]: <matplotlib.text at 0x11eb56908>



Clearly by using these two components we can easily separate these two classes.

Interpreting the components

Unfortunately, with this great power of dimensionality reduction, comes the cost of being able to easily understand what these components represent.

The components correspond to combinations of the original features, the components themselves are stored as an attribute of the fitted PCA object:

In [55]: pca.components_

```
Out[55]: array([[-0.21890244, -0.10372458, -0.22753729, -0.22099499, -0.14258969,
 -0.23928535, -0.25840048, -0.26085376, -0.13816696, -0.06436335,
 -0.20597878, -0.01742803, -0.21132592, -0.20286964, -0.01453145,
 -0.17039345, -0.15358979, -0.1834174 , -0.04249842, -0.10256832,
 -0.22799663, -0.10446933, -0.23663968, -0.22487053, -0.12795256,
 -0.21009588, -0.22876753, -0.25088597, -0.12290456, -0.13178394],
 [ 0.23385713,  0.05970609,  0.21518136,  0.23107671, -0.18611302,
 -0.15189161, -0.06016536,  0.0347675 , -0.19034877, -0.36657547,
```

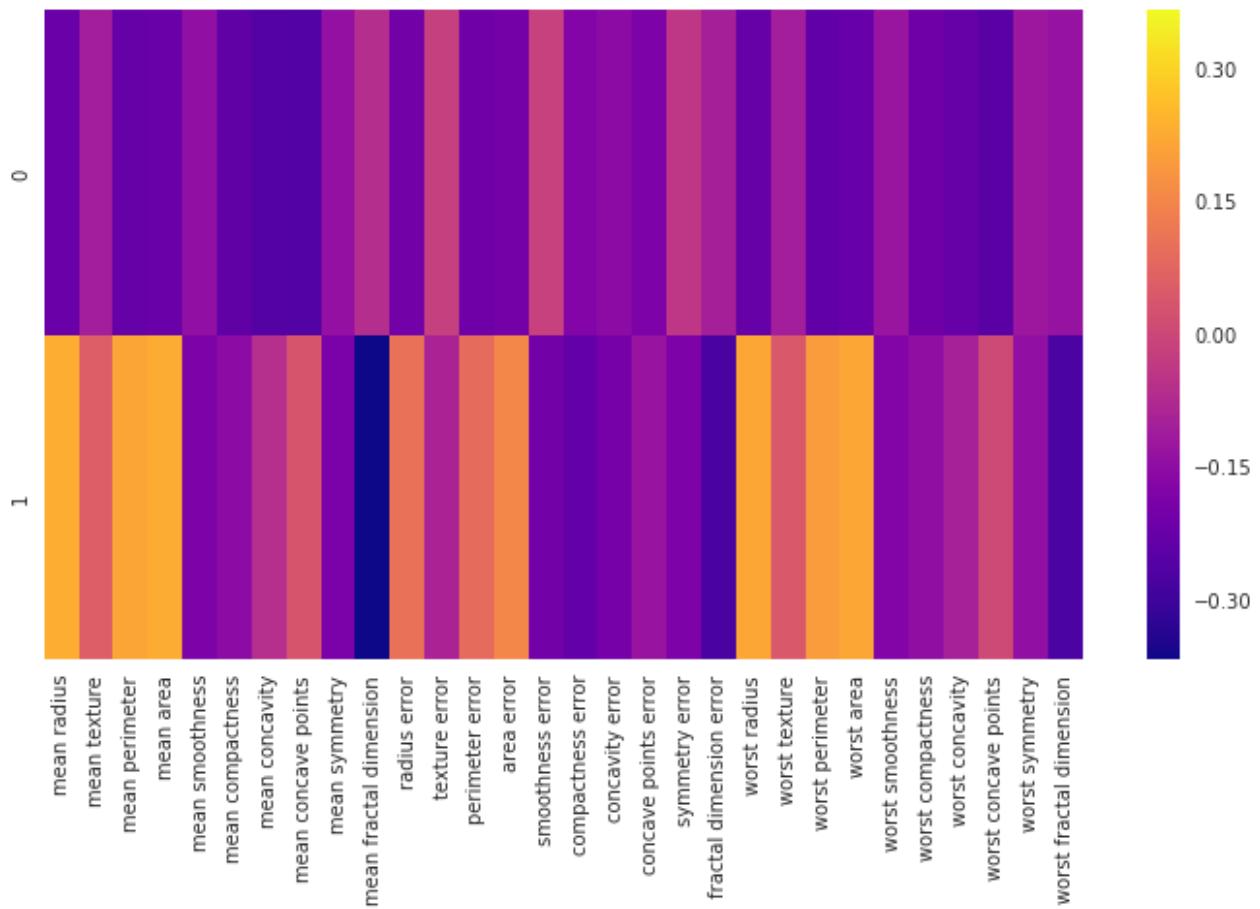
```
0.10555215, -0.08997968,  0.08945723,  0.15229263, -0.20443045,
-0.2327159 , -0.19720728, -0.13032156, -0.183848 , -0.28009203,
0.21986638,  0.0454673 ,  0.19987843,  0.21935186, -0.17230435,
-0.14359317, -0.09796411,  0.00825724, -0.14188335, -0.27533947]])
```

In this numpy matrix array, each row represents a principal component, and each column relates back to the original features. we can visualize this relationship with a heatmap:

```
In [56]: #Creating an datafram of pca components as it was in array
df_comp = pd.DataFrame(pca.components_,columns=cancer['feature_names'])
```

```
In [57]: plt.figure(figsize=(12,6))
#Here we create an heat map to find corolation
sns.heatmap(df_comp,cmap='plasma')
#If the colour is yellow it has more cooleration
#If the colour is purple it has less coloration
#It is compared according to it
```

```
Out[57]: <matplotlib.axes._subplots.AxesSubplot at 0x11d546f98>
```



This heatmap and the color bar basically represent the correlation between the various feature and the principal component itself.

Conclusion

Hopefully this information is useful to you when dealing with high dimensional data!

Recommender Systems with Python

Welcome to the code notebook for Recommender Systems with Python. In this lecture we will develop basic recommendation systems using Python and pandas. There is another notebook: *Advanced Recommender Systems with Python*. That notebook goes into more detail with the same data set.

In this notebook, we will focus on providing a basic recommendation system by suggesting items that are most similar to a particular item, in this case, movies. Keep in mind, this is not a true robust recommendation system, to describe it more accurately, it just tells you what movies/items are most similar to your movie choice.

There is no project for this topic, instead you have the option to work through the advanced lecture version of this notebook (totally optional!).

Let's get started!

Import Libraries

```
In [136...]: #Importing all the required packages
import numpy as np
import pandas as pd
```

Get the Data

```
In [137...]: # Creating columns names according to the names
column_names = ['user_id', 'item_id', 'rating', 'timestamp']
#Reading all the packages
df = pd.read_csv('u.data', sep='\t', names=column_names)
```

```
In [138...]: #Viewing all the details
df.head()
```

```
Out[138...]:
```

	user_id	item_id	rating	timestamp
0	0	50	5	881250949
1	0	172	5	881250949
2	0	133	1	881250949
3	196	242	3	881250949
4	186	302	3	891717742

Now let's get the movie titles:

```
In [139...]: #Reading the movie titles
```

```
movie_titles = pd.read_csv("Movie_Id_Titles")
movie_titles.head()
```

Out[139...]

	item_id	title
0	1	Toy Story (1995)
1	2	GoldenEye (1995)
2	3	Four Rooms (1995)
3	4	Get Shorty (1995)
4	5	Copycat (1995)

We can merge them together:

In [140...]

```
#Merging the movie_titles and column_names according to the item_id
#As both dataframe has common user_id
df = pd.merge(df, movie_titles, on='item_id')
df.head()
```

Out[140...]

	user_id	item_id	rating	timestamp	title
0	0	50	5	881250949	Star Wars (1977)
1	290	50	5	880473582	Star Wars (1977)
2	79	50	4	891271545	Star Wars (1977)
3	2	50	5	888552084	Star Wars (1977)
4	8	50	5	879362124	Star Wars (1977)

EDA

Let's explore the data a bit and get a look at some of the best rated movies.

Visualization Imports

In [160...]

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('white')
%matplotlib inline
```

Let's create a ratings dataframe with average rating and number of ratings:

In [142...]

```
#Grouping all the title and rating and finding mean of the data and sorting it
df.groupby('title')['rating'].mean().sort_values(ascending=False).head()
```

Out[142...]

title	rating
Marlene Dietrich: Shadow and Light (1996)	5.0
Prefontaine (1997)	5.0
Santa with Muscles (1996)	5.0
Star Kid (1997)	5.0
Someone Else's America (1995)	5.0
Name: rating, dtype: float64	

```
In [143...]: #Grouping the data and finding count
#This helps to find how many numbers of people watching the data
df.groupby('title')['rating'].count().sort_values(ascending=False).head()
```

```
Out[143...]: title
Star Wars (1977)      584
Contact (1997)        509
Fargo (1996)          508
Return of the Jedi (1983) 507
Liar Liar (1997)       485
Name: rating, dtype: int64
```

```
In [144...]: #Creating an dataframe and grouping it and finding the mean
ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
ratings.head()
```

title	rating
'Til There Was You (1997)	2.333333
1-900 (1994)	2.600000
101 Dalmatians (1996)	2.908257
12 Angry Men (1957)	4.344000
187 (1997)	3.024390

Now set the number of ratings column:

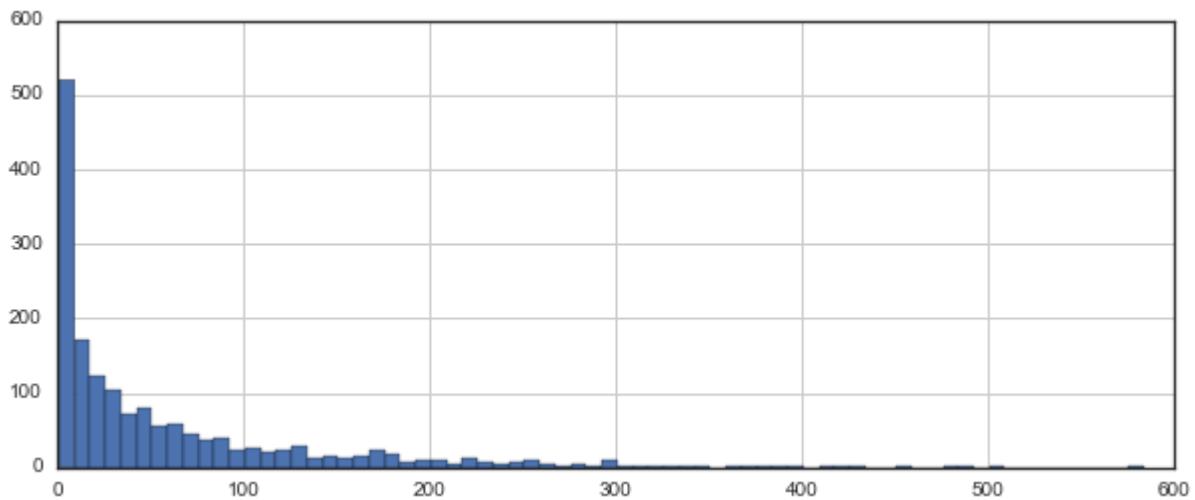
```
In [159...]: #Creating an variable no of ratings to get the count
#no of ratings
ratings['num of ratings'] = pd.DataFrame(df.groupby('title')['rating'].count())
ratings.head()
```

title	rating	num of ratings
'Til There Was You (1997)	2.333333	9
1-900 (1994)	2.600000	5
101 Dalmatians (1996)	2.908257	109
12 Angry Men (1957)	4.344000	125
187 (1997)	3.024390	41

Now a few histograms:

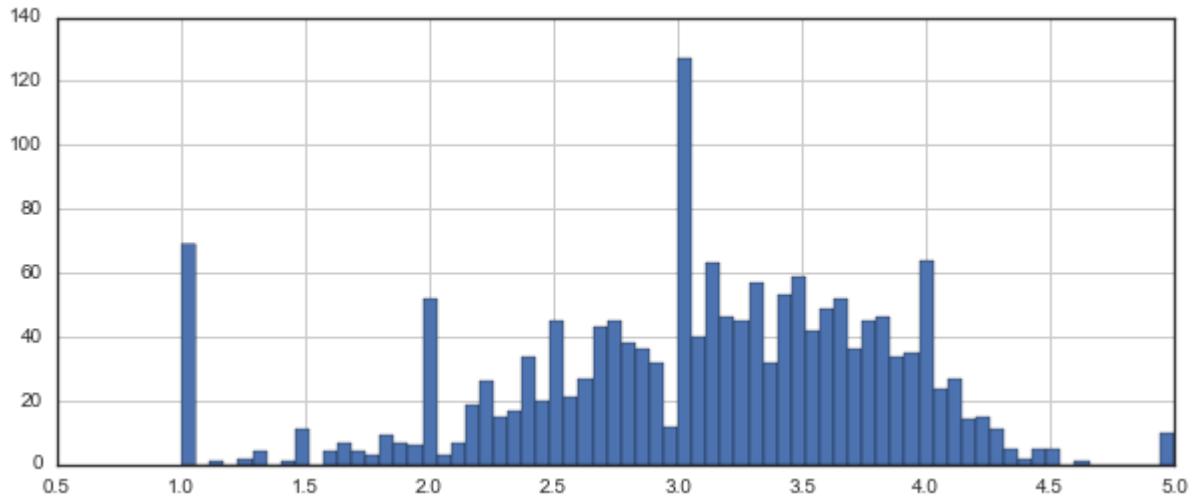
```
In [146...]: #Creating an histogram plot for ratings i,e each movie the count
plt.figure(figsize=(10,4))
ratings['num of ratings'].hist(bins=70)
```

```
Out[146...]: <matplotlib.axes._subplots.AxesSubplot at 0x1258f8780>
```



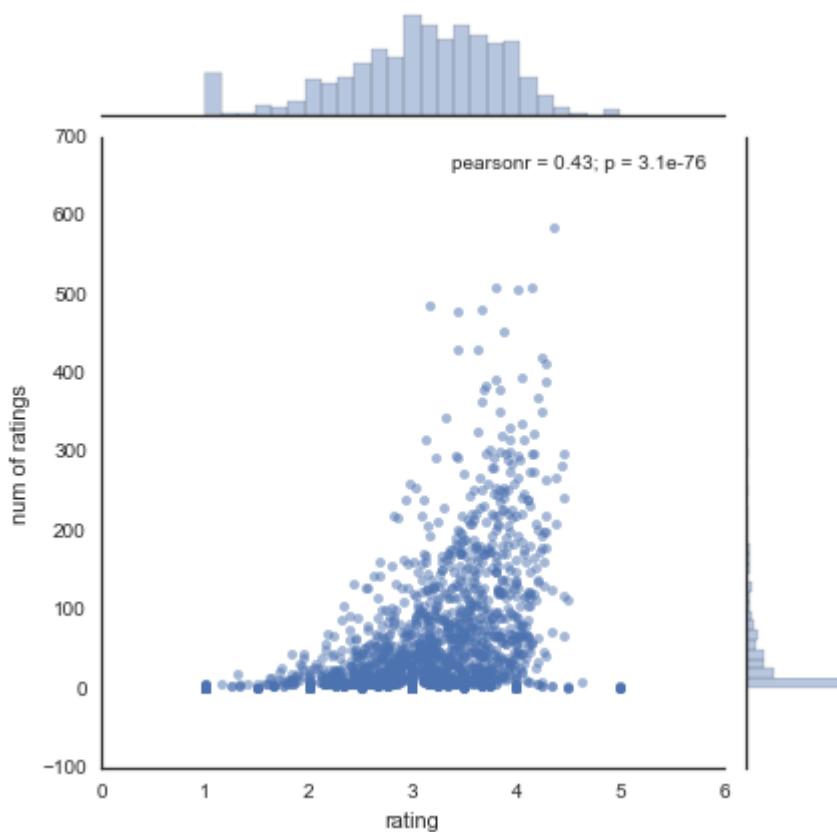
```
In [147...]: #Viewing the rating from 1.0 to 5.0  
plt.figure(figsize=(10,4))  
ratings['rating'].hist(bins=70)
```

```
Out[147...]: <matplotlib.axes._subplots.AxesSubplot at 0x125d12908>
```



```
In [148...]: #Creating an jointplot  
sns.jointplot(x='rating',y='num of ratings',data=ratings,alpha=0.5)
```

```
Out[148...]: <seaborn.axisgrid.JointGrid at 0x126005320>
```



Okay! Now that we have a general idea of what the data looks like, let's move on to creating a simple recommendation system:

Recommend Similar Movies

Now let's create a matrix that has the user ids on one access and the movie title on another axis. Each cell will then consist of the rating the user gave to that movie. Note there will be a lot of NaN values, because most people have not seen most of the movies.

```
In [149]: moviemat = df.pivot_table(index='user_id', columns='title', values='rating')
moviemat.head()
```

Out[149...]

	'Til There Was You (1997)	1-900 (1994)	101 Dalmatians (1996)	12 Angry Men (1957)	187 (1997)	2 Days in the Valley (1996)	20,000 Leagues Under the Sea (1954)	2001: A Space Odyssey (1968)	3 Ninjas: High Noon At Mega Mountain (1998)	39 Steps, The (1935)	...
user_id											
0	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
1	NaN	NaN		2.0	5.0	NaN	NaN	3.0	4.0	NaN	NaN
2	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN
3	NaN	NaN		NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN		NaN	NaN	NaN	NaN	NaN	NaN	NaN	...

5 rows × 1664 columns

Most rated movie:

```
In [150...]: ratings.sort_values('num of ratings', ascending=False).head(10)
```

```
Out[150...]:
```

	rating	num of ratings
title		
Star Wars (1977)	4.359589	584
Contact (1997)	3.803536	509
Fargo (1996)	4.155512	508
Return of the Jedi (1983)	4.007890	507
Liar Liar (1997)	3.156701	485
English Patient, The (1996)	3.656965	481
Scream (1996)	3.441423	478
Toy Story (1995)	3.878319	452
Air Force One (1997)	3.631090	431
Independence Day (ID4) (1996)	3.438228	429

Let's choose two movies: starwars, a sci-fi movie. And Liar Liar, a comedy.

```
In [161...]: ratings.head()
```

```
Out[161...]:
```

	rating	num of ratings
title		
'Til There Was You (1997)	2.333333	9
1-900 (1994)	2.600000	5
101 Dalmatians (1996)	2.908257	109
12 Angry Men (1957)	4.344000	125
187 (1997)	3.024390	41

Now let's grab the user ratings for those two movies:

```
In [162...]: starwars_user_ratings = moviemat['Star Wars (1977)']
liarliar_user_ratings = moviemat['Liar Liar (1997)']
starwars_user_ratings.head()
```

```
Out[162...]: user_id
0    5.0
1    5.0
2    5.0
3    NaN
4    5.0
Name: Star Wars (1977), dtype: float64
```

We can then use corrwith() method to get correlations between two pandas series:

```
In [163...]: similar_to_starwars = moviemat.corrwith(starwars_user_ratings)
similar_to_liarliar = moviemat.corrwith(liarliar_user_ratings)
```

```
/Users/marci/anaconda/lib/python3.5/site-packages/numpy/lib/function_base.py:2487: RuntimeWarning: Degrees of freedom <= 0 for slice
    warnings.warn("Degrees of freedom <= 0 for slice", RuntimeWarning)
```

Let's clean this by removing NaN values and using a DataFrame instead of a series:

```
In [164...]: corr_starwars = pd.DataFrame(similar_to_starwars,columns=['Correlation'])
corr_starwars.dropna(inplace=True)
corr_starwars.head()
```

Out[164...]:

Correlation	
	title
'Til There Was You (1997)	0.872872
1-900 (1994)	-0.645497
101 Dalmatians (1996)	0.211132
12 Angry Men (1957)	0.184289
187 (1997)	0.027398

Now if we sort the dataframe by correlation, we should get the most similar movies, however note that we get some results that don't really make sense. This is because there are a lot of movies only watched once by users who also watched star wars (it was the most popular movie).

```
In [155...]: corr_starwars.sort_values('Correlation', ascending=False).head(10)
```

Out[155...]:

Correlation	
	title
	Commandments (1997) 1.0
	Cosi (1996) 1.0
	No Escape (1994) 1.0
	Stripes (1981) 1.0
	Man of the Year (1995) 1.0
	Hollow Reed (1996) 1.0
	Beans of Egypt, Maine, The (1994) 1.0
	Good Man in Africa, A (1994) 1.0
Old Lady Who Walked in the Sea, The (Vieille qui marchait dans la mer, La) (1991)	1.0
Outlaw, The (1943)	1.0

Let's fix this by filtering out movies that have less than 100 reviews (this value was chosen based off the histogram from earlier).

```
In [165...]: corr_starwars = corr_starwars.join(ratings['num of ratings'])
corr_starwars.head()
```

Out[165...]

Correlation num of ratings

title	Correlation	num of ratings
'Til There Was You (1997)	0.872872	9
1-900 (1994)	-0.645497	5
101 Dalmatians (1996)	0.211132	109
12 Angry Men (1957)	0.184289	125
187 (1997)	0.027398	41

Now sort the values and notice how the titles make a lot more sense:

In [157...]: corr_starwars[corr_starwars['num of ratings']>100].sort_values('Correlation', ascending=

Out[157...]

Correlation num of ratings

title	Correlation	num of ratings
Star Wars (1977)	1.000000	584
Empire Strikes Back, The (1980)	0.748353	368
Return of the Jedi (1983)	0.672556	507
Raiders of the Lost Ark (1981)	0.536117	420
Austin Powers: International Man of Mystery (1997)	0.377433	130

Now the same for the comedy Liar Liar:

In [158...]: corr_liarliar = pd.DataFrame(similar_to_liarliar, columns=['Correlation'])
corr_liarliar.dropna(inplace=True)
corr_liarliar = corr_liarliar.join(ratings['num of ratings'])
corr_liarliar[corr_liarliar['num of ratings']>100].sort_values('Correlation', ascending=

Out[158...]

Correlation num of ratings

title	Correlation	num of ratings
Liar Liar (1997)	1.000000	485
Batman Forever (1995)	0.516968	114
Mask, The (1994)	0.484650	129
Down Periscope (1996)	0.472681	101
Con Air (1997)	0.469828	137

Advanced Recommender Systems with Python

Welcome to the code notebook for creating Advanced Recommender Systems with Python. This is an optional lecture notebook for you to check out. Currently there is no video for this lecture because of the level of mathematics used and the heavy use of SciPy here.

Recommendation Systems usually rely on larger data sets and specifically need to be organized in a particular fashion. Because of this, we won't have a project to go along with this topic, instead we will have a more intensive walkthrough process on creating a recommendation system with Python with the same Movie Lens Data Set.

Note: The actual mathematics behind recommender systems is pretty heavy in Linear Algebra.

Methods Used

Two most common types of recommender systems are **Content-Based** and **Collaborative Filtering (CF)**.

- Collaborative filtering produces recommendations based on the knowledge of users' attitude to items, that is it uses the "wisdom of the crowd" to recommend items.
- Content-based recommender systems focus on the attributes of the items and give you recommendations based on the similarity between them.

Collaborative Filtering

In general, Collaborative filtering (CF) is more commonly used than content-based systems because it usually gives better results and is relatively easy to understand (from an overall implementation perspective). The algorithm has the ability to do feature learning on its own, which means that it can start to learn for itself what features to use.

CF can be divided into **Memory-Based Collaborative Filtering** and **Model-Based Collaborative filtering**.

In this tutorial, we will implement Model-Based CF by using singular value decomposition (SVD) and Memory-Based CF by computing cosine similarity.

The Data

We will use famous MovieLens dataset, which is one of the most common datasets used when implementing and testing recommender engines. It contains 100k movie ratings from 943 users and a selection of 1682 movies.

You can download the dataset [here](#) or just use the u.data file that is already included in this folder.

Getting Started

Let's import some libraries we will need:

```
In [2]: import numpy as np
import pandas as pd
```

We can then read in the **u.data** file, which contains the full dataset. You can read a brief description of the dataset [here](#).

Note how we specify the separator argument for a Tab separated file.

```
In [3]: column_names = ['user_id', 'item_id', 'rating', 'timestamp']
df = pd.read_csv('u.data', sep='\t', names=column_names)
```

Let's take a quick look at the data.

```
In [4]: df.head()
```

```
Out[4]:   user_id  item_id  rating  timestamp
0         0       50      5  881250949
1         0      172      5  881250949
2         0      133      1  881250949
3        196      242      3  881250949
4        186      302      3  891717742
```

Note how we only have the item_id, not the movie name. We can use the Movie_ID_Titles csv file to grab the movie names and merge it with this dataframe:

```
In [15]: movie_titles = pd.read_csv("Movie_Id_Titles")
movie_titles.head()
```

```
Out[15]:    item_id          title
0         1  Toy Story (1995)
1         2  GoldenEye (1995)
2         3  Four Rooms (1995)
3         4  Get Shorty (1995)
4         5  Copycat (1995)
```

Then merge the dataframes:

```
In [16]: df = pd.merge(df, movie_titles, on='item_id')
df.head()
```

```
Out[16]:   user_id  item_id  rating  timestamp          title
0         0       50      5  881250949  Star Wars (1977)
```

	user_id	item_id	rating	timestamp	title
1	290	50	5	880473582	Star Wars (1977)
2	79	50	4	891271545	Star Wars (1977)
3	2	50	5	888552084	Star Wars (1977)
4	8	50	5	879362124	Star Wars (1977)

Now let's take a quick look at the number of unique users and movies.

```
In [26]: n_users = df.user_id.nunique()
n_items = df.item_id.nunique()

print('Num. of Users: '+ str(n_users))
print('Num of Movies: '+str(n_items))

Num. of Users: 944
Num of Movies: 1682
```

Train Test Split

Recommendation Systems by their very nature are very difficult to evaluate, but we will still show you how to evaluate them in this tutorial. In order to do this, we'll split our data into two sets. However, we won't do our classic X_train,X_test,y_train,y_test split. Instead we can actually just segment the data into two sets of data:

```
In [27]: from sklearn.cross_validation import train_test_split
train_data, test_data = train_test_split(df, test_size=0.25)
```

Memory-Based Collaborative Filtering

Memory-Based Collaborative Filtering approaches can be divided into two main sections: **user-item filtering** and **item-item filtering**.

A *user-item filtering* will take a particular user, find users that are similar to that user based on similarity of ratings, and recommend items that those similar users liked.

In contrast, *item-item filtering* will take an item, find users who liked that item, and find other items that those users or similar users also liked. It takes items and outputs other items as recommendations.

- *Item-Item Collaborative Filtering*: "Users who liked this item also liked ..."
- *User-Item Collaborative Filtering*: "Users who are similar to you also liked ..."

In both cases, you create a user-item matrix which built from the entire dataset.

Since we have split the data into testing and training we will need to create two [943 x 1682] matrices (all users by all movies).

The training matrix contains 75% of the ratings and the testing matrix contains 25% of the ratings.

Example of user-item matrix:  blog8

After you have built the user-item matrix you calculate the similarity and create a similarity matrix.

The similarity values between items in *Item-Item Collaborative Filtering* are measured by observing all the users who have rated both items.



For *User-Item Collaborative Filtering* the similarity values between users are measured by observing all the items that are rated by both users.



A distance metric commonly used in recommender systems is *cosine similarity*, where the ratings are seen as vectors in n -dimensional space and the similarity is calculated based on the angle between these vectors. Cosine similarity for users a and m can be calculated using the formula below, where you take dot product of the user vector \mathbf{u}_k and the user vector \mathbf{u}_a and divide it by multiplication of the Euclidean lengths of the vectors.

$$s_u^{\cos}(u_k, u_a) = \frac{\mathbf{u}_k \cdot \mathbf{u}_a}{\|\mathbf{u}_k\| \|\mathbf{u}_a\|} = \frac{\sum x_{k,m} x_{a,m}}{\sqrt{\sum x_{k,m}^2 \sum x_{a,m}^2}}$$

To calculate similarity between items m and b you use the formula:

$$s_u^{\cos}(i_m, i_b) = \frac{i_m \cdot i_b}{\|i_m\| \|i_b\|} = \frac{\sum x_{a,m} x_{a,b}}{\sqrt{\sum x_{a,m}^2 \sum x_{a,b}^2}}$$

Your first step will be to create the user-item matrix. Since you have both testing and training data you need to create two matrices.

```
In [28]: #Create two user-item matrices, one for training and another for testing
train_data_matrix = np.zeros((n_users, n_items))
for line in train_data.itertuples():
    train_data_matrix[line[1]-1, line[2]-1] = line[3]

test_data_matrix = np.zeros((n_users, n_items))
for line in test_data.itertuples():
    test_data_matrix[line[1]-1, line[2]-1] = line[3]
```

You can use the `pairwise_distances` function from sklearn to calculate the cosine similarity. Note, the output will range from 0 to 1 since the ratings are all positive.

```
In [29]: from sklearn.metrics.pairwise import pairwise_distances
user_similarity = pairwise_distances(train_data_matrix, metric='cosine')
item_similarity = pairwise_distances(train_data_matrix.T, metric='cosine')
```

Next step is to make predictions. You have already created similarity matrices: `user_similarity` and `item_similarity` and therefore you can make a prediction by applying following formula for user-based CF:

$$\hat{x}_{k,m} = \bar{x}_k + \frac{\sum_{u_a} sim_u(u_k, u_a)(x_{a,m} - \bar{x}_{u_a})}{\sum_{u_a} |sim_u(u_k, u_a)|}$$

You can look at the similarity between users k and a as weights that are multiplied by the ratings of a similar user a (corrected for the average rating of that user). You will need to normalize it so that the ratings stay between 1 and 5 and, as a final step, sum the average ratings for the user that you are trying to predict.

The idea here is that some users may tend always to give high or low ratings to all movies. The relative difference in the ratings that these users give is more important than the absolute values. To give an example: suppose, user k gives 4 stars to his favourite movies and 3 stars to all other good movies. Suppose now that another user t rates movies that he/she likes with 5 stars, and the movies he/she fell asleep over with 3 stars. These two users could have a very similar taste but treat the rating system differently.

When making a prediction for item-based CF you don't need to correct for users average rating since query user itself is used to do predictions.

$$\hat{x}_{k,m} = \frac{\sum_{i_b} sim_i(i_m, i_b)(x_{k,b})}{\sum_{i_b} |sim_i(i_m, i_b)|}$$

```
In [30]: def predict(ratings, similarity, type='user'):
    if type == 'user':
        mean_user_rating = ratings.mean(axis=1)
        #You use np.newaxis so that mean_user_rating has same format as ratings
        ratings_diff = (ratings - mean_user_rating[:, np.newaxis])
        pred = mean_user_rating[:, np.newaxis] + similarity.dot(ratings_diff) / np.array([np.abs(similarity).sum(axis=1)])
    elif type == 'item':
        pred = ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)])
    return pred
```



```
In [31]: item_prediction = predict(train_data_matrix, item_similarity, type='item')
user_prediction = predict(train_data_matrix, user_similarity, type='user')
```

Evaluation

There are many evaluation metrics but one of the most popular metric used to evaluate accuracy of

$$RMSE = \sqrt{\frac{1}{N} \sum (x_i - \hat{x}_i)^2}$$

predicted ratings is *Root Mean Squared Error (RMSE)*.

You can use the `mean_square_error` (MSE) function from `sklearn`, where the RMSE is just the square root of MSE. To read more about different evaluation metrics you can take a look at [this article](#).

Since you only want to consider predicted ratings that are in the test dataset, you filter out all other elements in the prediction matrix with `prediction[ground_truth.nonzero()]`.

```
In [32]: from sklearn.metrics import mean_squared_error
from math import sqrt
def rmse(prediction, ground_truth):
    prediction = prediction[ground_truth.nonzero()].flatten()
    ground_truth = ground_truth[ground_truth.nonzero()].flatten()
    return sqrt(mean_squared_error(prediction, ground_truth))
```

```
In [33]: print('User-based CF RMSE: ' + str(rmse(user_prediction, test_data_matrix)))
print('Item-based CF RMSE: ' + str(rmse(item_prediction, test_data_matrix)))
```

User-based CF RMSE: 3.135451660158989
 Item-based CF RMSE: 3.4593766647252515

Memory-based algorithms are easy to implement and produce reasonable prediction quality. The drawback of memory-based CF is that it doesn't scale to real-world scenarios and doesn't address the well-known cold-start problem, that is when new user or new item enters the system. Model-based CF methods are scalable and can deal with higher sparsity level than memory-based models, but also suffer when new users or items that don't have any ratings enter the system. I would like to thank Ethan Rosenthal for his [post](#) about Memory-Based Collaborative Filtering.

Model-based Collaborative Filtering

Model-based Collaborative Filtering is based on **matrix factorization (MF)** which has received greater exposure, mainly as an unsupervised learning method for latent variable decomposition and dimensionality reduction. Matrix factorization is widely used for recommender systems where it can deal better with scalability and sparsity than Memory-based CF. The goal of MF is to learn the latent preferences of users and the latent attributes of items from known ratings (learn features that describe the characteristics of ratings) to then predict the unknown ratings through the dot product of the latent features of users and items. When you have a very sparse matrix, with a lot of dimensions, by doing matrix factorization you can restructure the user-item matrix into low-rank structure, and you can represent the matrix by the multiplication of two low-rank matrices, where the rows contain the latent vector. You fit this matrix to approximate your original matrix, as closely as possible, by multiplying the low-rank matrices together, which fills in the entries missing in the original matrix.

Let's calculate the sparsity level of MovieLens dataset:

```
In [34]: sparsity=round(1.0-len(df)/float(n_users*n_items),3)
print('The sparsity level of MovieLens100K is ' + str(sparsity*100) + '%')
```

The sparsity level of MovieLens100K is 93.7%

To give an example of the learned latent preferences of the users and items: let's say for the MovieLens dataset you have the following information: (*user id, age, location, gender, movie id, director, actor, language, year, rating*). By applying matrix factorization the model learns that important user features are *age group (under 10, 10-18, 18-30, 30-90)*, *location* and *gender*, and for movie features it learns that *decade, director* and *actor* are most important. Now if you look into the information you have stored, there is no such feature as the *decade*, but the model can learn on its own. The important aspect is that the CF model only uses data (*user_id, movie_id, rating*) to learn

the latent features. If there is little data available model-based CF model will predict poorly, since it will be more difficult to learn the latent features.

Models that use both ratings and content features are called **Hybrid Recommender Systems** where both Collaborative Filtering and Content-based Models are combined. Hybrid recommender systems usually show higher accuracy than Collaborative Filtering or Content-based Models on their own: they are capable to address the cold-start problem better since if you don't have any ratings for a user or an item you could use the metadata from the user or item to make a prediction.

SVD

A well-known matrix factorization method is **Singular value decomposition (SVD)**. Collaborative Filtering can be formulated by approximating a matrix X by using singular value decomposition. The winning team at the Netflix Prize competition used SVD matrix factorization models to produce product recommendations, for more information I recommend to read articles: [Netflix Recommendations: Beyond the 5 stars](#) and [Netflix Prize and SVD](#). The general equation can be expressed as follows: $X = USV^T$

Given $m \times n$ matrix X :

- U is an $(m \times r)$ orthogonal matrix
- S is an $(r \times r)$ diagonal matrix with non-negative real numbers on the diagonal
- V^T is an $(r \times n)$ orthogonal matrix

Elements on the diagonal in S are known as *singular values of X* .

Matrix X can be factorized to U , S and V . The U matrix represents the feature vectors corresponding to the users in the hidden feature space and the V matrix represents the feature vectors corresponding to the items in the hidden feature space. 

Now you can make a prediction by taking dot product of U , S and V^T .



```
In [35]: import scipy.sparse as sp
from scipy.sparse.linalg import svds

#get SVD components from train matrix. Choose k.
u, s, vt = svds(train_data_matrix, k = 20)
s_diag_matrix=np.diag(s)
X_pred = np.dot(np.dot(u, s_diag_matrix), vt)
print('User-based CF MSE: ' + str(rmse(X_pred, test_data_matrix)))
```

User-based CF MSE: 2.727093975231784

Carelessly addressing only the relatively few known entries is highly prone to overfitting. SVD can be very slow and computationally expensive. More recent work minimizes the squared error by applying alternating least square or stochastic gradient descent and uses regularization terms to prevent overfitting. Alternating least square and stochastic gradient descent methods for CF will be covered in the next tutorials.

Review:

- We have covered how to implement simple **Collaborative Filtering** methods, both memory-based CF and model-based CF.
- **Memory-based models** are based on similarity between items or users, where we use cosine-similarity.
- **Model-based CF** is based on matrix factorization where we use SVD to factorize the matrix.
- Building recommender systems that perform well in cold-start scenarios (where little data is available on new users and items) remains a challenge. The standard collaborative filtering method performs poorly in such settings.

Looking for more?

If you want to tackle your own recommendation system analysis, check out these data sets. Note: The files are quite large in most cases, not all the links may stay up to host the data, but the majority of them still work. Or just Google for your own data set!

Movies Recommendation:

MovieLens - Movie Recommendation Data Sets <http://www.grouplens.org/node/73>

Yahoo! - Movie, Music, and Images Ratings Data Sets
<http://webscope.sandbox.yahoo.com/catalog.php?datatype=r>

Jester - Movie Ratings Data Sets (Collaborative Filtering Dataset)
<http://www.ieor.berkeley.edu/~goldberg/jester-data/>

Cornell University - Movie-review data for use in sentiment-analysis experiments
<http://www.cs.cornell.edu/people/pabo/movie-review-data/>

Music Recommendation:

Last.fm - Music Recommendation Data Sets
<http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/index.html>

Yahoo! - Movie, Music, and Images Ratings Data Sets
<http://webscope.sandbox.yahoo.com/catalog.php?datatype=r>

Audioscrobbler - Music Recommendation Data Sets http://www-etud.iro.umontreal.ca/~bergstrj/audioscrobbler_data.html

Amazon - Audio CD recommendations <http://131.193.40.52/data/>

Books Recommendation:

Institut für Informatik, Universität Freiburg - Book Ratings Data Sets <http://www.informatik.uni-freiburg.de/~cziegler/BX/> Food Recommendation:

Chicago Entree - Food Ratings Data Sets
<http://archive.ics.uci.edu/ml/datasets/Entree+Chicago+Recommendation+Data+Merchandise>
Recommendation:

Healthcare Recommendation:

Nursing Home - Provider Ratings Data Set <http://data.medicare.gov/dataset/Nursing-Home-Compare-Provider-Ratings/mufm-vy8d>

Hospital Ratings - Survey of Patients Hospital Experiences <http://data.medicare.gov/dataset/Survey-of-Patients-Hospital-Experiences-HCAHPS-/rj76-22dk>

Dating Recommendation:

www.libimseti.cz - Dating website recommendation (collaborative filtering)
<http://www.occamslab.com/petricek/data/> Scholarly Paper Recommendation:

National University of Singapore - Scholarly Paper Recommendation
<http://www.comp.nus.edu.sg/~sugiyama/SchPaperRecData.html>

In []:



Introduction to Natural Language Processing



Reading Assignment

Read Wikipedia Article on
Natural Language Processing



NLP

Imagine you work for Google News and you want to group news articles by topic

Or you work for a legal firm and you need to sift through thousands of pages of legal documents to find relevant ones.

This is where NLP can help!



NLP

We will want to:

- Compile Documents
- Featurize Them
- Compare their features



NLP

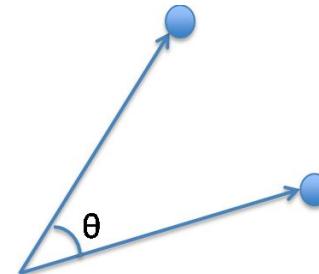
Simple Example:

- You have 2 documents:
 - “Blue House”
 - “Red House”
- Featurize based on word count:
 - “Blue House” -> (red,blue,house) -> (0,1,1)
 - “Red House” -> (red,blue,house) -> (1,0,1)



- A document represented as a vector of word counts is called a “Bag of Words”
 - “Blue House” -> (red,blue,house) -> (0,1,1)
 - “Red House” -> (red,blue,house) -> (1,0,1)
- You can use cosine similarity on the vectors made to determine similarity:

$$\text{sim}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$





NLP

- We can improve on Bag of Words by adjusting word counts based on their frequency in corpus (the group of all the documents)
- We can use TF-IDF (Term Frequency - Inverse Document Frequency)



- Term Frequency - Importance of the term within that document
 - $TF(d,t) = \text{Number of occurrences of term } t \text{ in document } d$
- Inverse Document Frequency - Importance of the term in the corpus
 - $IDF(t) = \log(D/t)$ where
 - $D = \text{total number of documents}$
 - $t = \text{number of documents with the term}$



NLP

- Mathematically, TF-IDF is then expressed:

$$w_{x,y} = tf_{x,y} \times \log \left(\frac{N}{df_x} \right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents



Example with R

Let's go to RStudio and begin to explore a project!

NLP (Natural Language Processing) with Python

This is the notebook that goes along with the NLP video lecture!

In this lecture we will discuss a higher level overview of the basics of Natural Language Processing, which basically consists of combining machine learning techniques with text, and using math and statistics to get that text in a format that the machine learning algorithms can understand!

Once you've completed this lecture you'll have a project using some Yelp Text Data!

```
In [ ]: Step 1: Removing the punctuation in the message  
Step 2: Removing the stoping words like(the,in,is,or,not) which would not helps for the
```

Get the Data

We'll be using a dataset from the [UCI datasets](#)! This dataset is already located in the folder for this section.

The file we are using contains a collection of more than 5 thousand SMS phone messages. You can check out the **readme** file for more info.

Let's go ahead and use `rstrip()` plus a list comprehension to get a list of all the lines of text messages:

```
In [3]: #Getting the data form the dataset and viewing the Length of the message  
messages = [line.rstrip() for line in open('smsspamcollection/SMSSpamCollection')]  
print(len(messages))
```

5574

A collection of texts is also sometimes called "corpus". Let's print the first ten messages and number them using **enumerate**:

```
In [4]: #Combining the nessages and with numbers  
#Enummerating means combining the 0,1,2,3... numbers with strings  
for message_no, message in enumerate(messages[:10]):  
    print(message_no, message)  
    print('\n')
```

0 ham Go until jurong point, crazy.. Available only in bugis n great world la e buffe t... Cine there got amore wat...

1 ham Ok lar... Joking wif u oni...

2 spam Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 8 7121 to receive entry question(std txt rate)T&C's apply 08452810075over18's

3 ham U dun say so early hor... U c already then say...

4 ham Nah I don't think he goes to usf, he lives around here though

5 spam FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, £1.50 to rcv

6 ham Even my brother is not like to speak with me. They treat me like aids patent.

7 ham As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune

8 spam WINNER!! As a valued network customer you have been selected to receivea £900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only.

9 spam Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030

Due to the spacing we can tell that this is a **TSV** ("tab separated values") file, where the first column is a label saying whether the given message is a normal message (commonly known as "ham") or "spam". The second column is the message itself. (Note our numbers aren't part of the file, they are just from the **enumerate** call).

Using these labeled ham and spam examples, we'll **train a machine learning model to learn to discriminate between ham/spam automatically**. Then, with a trained model, we'll be able to **classify arbitrary unlabeled messages** as ham or spam.

From the official SciKit Learn documentation, we can visualize our process:



Instead of parsing TSV manually using Python, we can just take advantage of pandas! Let's go ahead and import it!

```
In [6]: #Importing the required packages
import pandas as pd
```

We'll use **read_csv** and make note of the **sep** argument, we can also specify the desired column names by passing in a list of *names*.

```
In [7]: #Reading the dataset
messages = pd.read_csv('smsspamcollection/SMSSpamCollection', sep='\t',
                       names=["label", "message"])
messages.head()
```

	label	message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...

	label	message
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Exploratory Data Analysis

Let's check out some of the stats with some plots and the built-in methods in pandas!

```
In [8]: #Checking the count and unique messages
messages.describe()
```

	label	message
count	5572	5572
unique	2	5169
top	ham	Sorry, I'll call later
freq	4825	30

Let's use **groupby** to use describe by label, this way we can begin to think about the features that separate ham and spam!

```
In [9]: #Grouping the spam and ham seperately for viewong unique and count messages
messages.groupby('label').describe()
```

	label	message
count		4825
unique		4516
ham	top	Sorry, I'll call later
	freq	30
count		747
unique		653
spam	top	Please call our customer service representativ...
	freq	4

As we continue our analysis we want to start thinking about the features we are going to be using. This goes along with the general idea of [feature engineering](#). The better your domain knowledge on the data, the better your ability to engineer more features from it. Feature engineering is a very large part of spam detection in general. I encourage you to read up on the topic!

Let's make a new column to detect how long the text messages are:

```
In [10]: messages['length'] = messages['message'].apply(len)
messages.head()
```

	label	message	length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61

Data Visualization

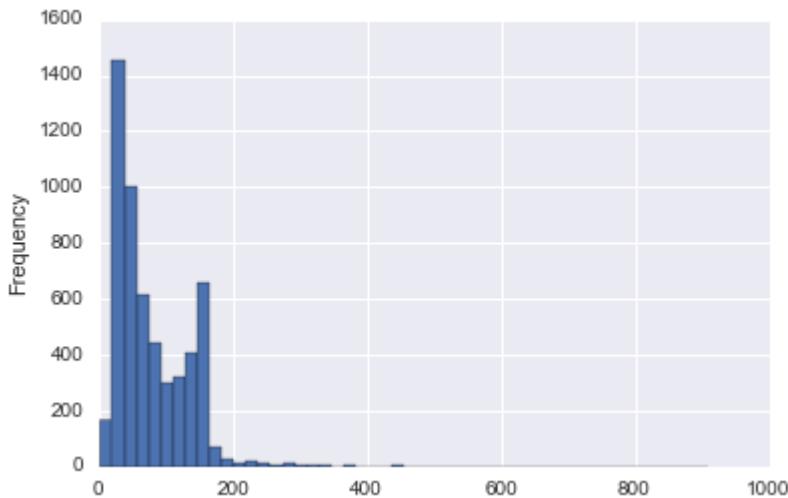
Let's visualize this! Let's do the imports:

```
In [11]: import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

```
In [12]: #Printing the length of the messages
messages['length'].plot(bins=50, kind='hist')
```

```
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x11a03c0b8>
```



Play around with the bin size! Looks like text length may be a good feature to think about! Let's try to explain why the x-axis goes all the way to 1000ish, this must mean that there is some really long message!

```
In [13]: messages.length.describe()
```

```
Out[13]: count    5572.000000
mean      80.489950
std       59.942907
min       2.000000
25%      36.000000
50%      62.000000
75%     122.000000
max     910.000000
Name: length, dtype: float64
```

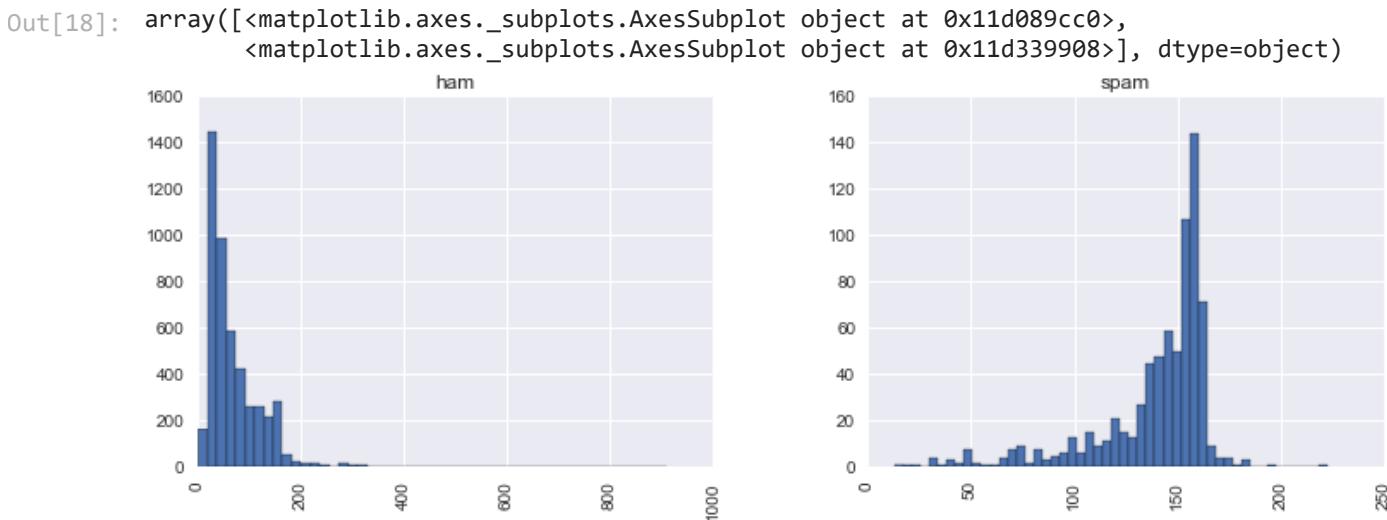
Woah! 910 characters, let's use masking to find this message:

```
In [14]: #Printing the particular message with Len
messages[messages['length'] == 910]['message'].iloc[0]
```

```
Out[14]: "For me the love should start with attraction.i should feel that I need her every time a round me.she should be the first thing which comes in my thoughts.I would start the day and end it with her.she should be there every time I dream.love will be then when my eve ry breath has her name.my life should happen around her.my life will be named to her.I w ould cry for her.will give all my happiness and take all her sorrows.I will be ready to fight with anyone for her.I will be in love when I will be doing the craziest things for her.love will be when I don't have to proove anyone that my girl is the most beautiful l ady on the whole planet.I will always be singing praises for her.love will be when I sta rt up making chicken curry and end up makiing sambar.life will be the most beautiful the n.will get every morning and thank god for the day because she is with me.I would like t o say a lot..will tell later.."
```

Looks like we have some sort of Romeo sending texts! But let's focus back on the idea of trying to see if message length is a distinguishing feature between ham and spam:

```
In [18]: #Printing the histogram between ham and spam messages
#here spam messages Length are more
messages.hist(column='length', by='label', bins=50, figsize=(12,4))
```



Very interesting! Through just basic EDA we've been able to discover a trend that spam messages tend to have more characters. (Sorry Romeo!)

Now let's begin to process the data so we can eventually use it with SciKit Learn!

Text Pre-processing

Our main issue with our data is that it is all in text format (strings). The classification algorithms that we've learned about so far will need some sort of numerical feature vector in order to perform the classification task. There are actually many methods to convert a corpus to a vector format. The simplest is the the [bag-of-words](#) approach, where each unique word in a text will be represented by one number.

In this section we'll convert the raw messages (sequence of characters) into vectors (sequences of numbers).

As a first step, let's write a function that will split a message into its individual words and return a list. We'll also remove very common words, ('the', 'a', etc..). To do this we will take advantage of the NLTK library. It's pretty much the standard library in Python for processing text and has a lot of useful features. We'll only use some of the basic ones here.

Let's create a function that will process the string in the message column, then we can just use **apply()** in pandas do process all the text in the DataFrame.

First removing punctuation. We can just take advantage of Python's built-in **string** library to get a quick list of all the possible punctuation:

```
In [1]: #Importing string module
import string

mess = 'Sample message! Notice: it has punctuation.'

# Check characters to see if they are in punctuation
nopunc = [char for char in mess if char not in string.punctuation]

# Join the characters again to form the string.
nopunc = ''.join(nopunc)
```

Now let's see how to remove stopwords. We can import a list of english stopwords from NLTK (check the documentation for more languages and info).

```
In [20]: #Getting stop words Like(is,or,not,in and) which is not required for natural language pr
from nltk.corpus import stopwords
stopwords.words('english')[0:10] # Show some stop words
```

```
Out[20]: ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your']
```

```
In [21]: nopunc.split()
```

```
Out[21]: ['Sample', 'message', 'Notice', 'it', 'has', 'punctuation']
```

```
In [22]: # Now just remove any stopwords
clean_mess = [word for word in nopunc.split() if word.lower() not in stopwords.words('e
```

```
In [23]: clean_mess
```

```
Out[23]: ['Sample', 'message', 'Notice', 'punctuation']
```

Now let's put both of these together in a function to apply it to our DataFrame later on:

```
In [24]: #Creating an function for the process
def text_process(mess):
    """
    Takes in a string of text, then performs the following:
    1. Remove all punctuation
    2. Remove all stopwords
    3. Returns a list of the cleaned text
    """
    # Check characters to see if they are in punctuation
    nopunc = [char for char in mess if char not in string.punctuation]
```

```
# Join the characters again to form the string.
nopunc = ''.join(nopunc)

# Now just remove any stopwords
return [word for word in nopunc.split() if word.lower() not in stopwords.words('eng')]
```

Here is the original DataFrame again:

In [25]: `messages.head()`

Out[25]:

	label	message	length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61

Now let's "tokenize" these messages. Tokenization is just the term used to describe the process of converting the normal text strings in to a list of tokens (words that we actually want).

Let's see an example output on on column:

Note: We may get some warnings or errors for symbols we didn't account for or that weren't in Unicode (like a British pound symbol)

In [26]: `# Check to make sure its working
messages['message'].head(5).apply(text_process)`

Out[26]:

```
0    [Go, jurong, point, crazy, Available, bugis, n...
1    [Ok, lar, Joking, wif, u, oni]
2    [Free, entry, 2, wkly, comp, win, FA, Cup, fin...
3    [U, dun, say, early, hor, U, c, already, say]
4    [Nah, dont, think, goes, usf, lives, around, t...
Name: message, dtype: object
```

In [27]: `# Show original dataframe
messages.head()`

Out[27]:

	label	message	length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61

Continuing Normalization

There are a lot of ways to continue normalizing this text. Such as [Stemming](#) or distinguishing by [part of speech](#).

NLTK has lots of built-in tools and great documentation on a lot of these methods. Sometimes they don't work well for text-messages due to the way a lot of people tend to use abbreviations or shorthand, For example:

'Nah dawg, IDK! Wut time u headin to da club?'

versus

'No dog, I don't know! What time are you heading to the club?'

Some text normalization methods will have trouble with this type of shorthand and so I'll leave you to explore those more advanced methods through the [NLTK book online](#).

For now we will just focus on using what we have to convert our list of words to an actual vector that SciKit-Learn can use.

Vectorization

Currently, we have the messages as lists of tokens (also known as [lemmas](#)) and now we need to convert each of those messages into a vector the SciKit Learn's algorithm models can work with.

Now we'll convert each message, represented as a list of tokens (lemmas) above, into a vector that machine learning models can understand.

We'll do that in three steps using the bag-of-words model:

1. Count how many times does a word occur in each message (Known as term frequency)
2. Weigh the counts, so that frequent tokens get lower weight (inverse document frequency)
3. Normalize the vectors to unit length, to abstract from the original text length (L2 norm)

Let's begin the first step:

Each vector will have as many dimensions as there are unique words in the SMS corpus. We will first use SciKit Learn's **CountVectorizer**. This model will convert a collection of text documents to a matrix of token counts.

We can imagine this as a 2-Dimensional matrix. Where the 1-dimension is the entire vocabulary (1 row per word) and the other dimension are the actual documents, in this case a column per text message.

For example:

	Message 1	Message 2	...	Message N
Word 1 Count	0	1	...	0
Word 2 Count	0	0	...	0

...	1	2	...	0
Word N Count	0	1	...	1

Since there are so many messages, we can expect a lot of zero counts for the presence of that word in that document. Because of this, SciKit Learn will output a [Sparse Matrix](#).

```
In [28]: #Importing the countvectorizer module
from sklearn.feature_extraction.text import CountVectorizer
```

There are a lot of arguments and parameters that can be passed to the CountVectorizer. In this case we will just specify the **analyzer** to be our own previously defined function:

```
In [31]: #Creating an model
bow_transformer = CountVectorizer(analyzer=text_process).fit(messages['message'])
#Creating an model for counting the words in each messages
# Print total number of vocab words
print(len(bow_transformer.vocabulary_))
```

11444

Let's take one text message and get its bag-of-words counts as a vector, putting to use our new `bow_transformer`:

```
In [32]: #To count number of words in the particular messages i,e taking 4th message
message4 = messages['message'][3]
print(message4)
```

U dun say so early hor... U c already then say...

Now let's see its vector representation:

```
In [34]: #transform is used to count the words in the message in comparing the model
bow4 = bow_transformer.transform([message4])
print(bow4)
print(bow4.shape)

#Finally the words are converted into the vector format i,e the number format
```

```
(0, 4073)      2
(0, 4638)      1
(0, 5270)      1
(0, 6214)      1
(0, 6232)      1
(0, 7197)      1
(0, 9570)      2
(1, 11444)
```

This means that there are seven unique words in message number 4 (after removing common stop words). Two of them appear twice, the rest only once. Let's go ahead and check and confirm which ones appear twice:

```
In [36]: #Getting the word in the message with index number
print(bow_transformer.get_feature_names()[4073])
print(bow_transformer.get_feature_names()[9570])
```

U
say

Now we can use `.transform` on our Bag-of-Words (bow) transformed object and transform the entire DataFrame of messages. Let's go ahead and check out how the bag-of-words counts for the

entire SMS corpus is a large, sparse matrix:

```
In [39]: messages_bow = bow_transformer.transform(messages['message'])
```

```
In [40]: print('Shape of Sparse Matrix: ', messages_bow.shape)
print('Amount of Non-Zero occurrences: ', messages_bow.nnz)
```

Shape of Sparse Matrix: (5572, 11444)

Amount of Non-Zero occurrences: 50795

```
In [46]: #Mathematical formula to get the sparsity value
sparsity = (100.0 * messages_bow.nnz / (messages_bow.shape[0] * messages_bow.shape[1]))
print('sparsity: {}'.format(round(sparsity)))
```

sparsity: 0

After the counting, the term weighting and normalization can be done with **TF-IDF**, using scikit-learn's `TfidfTransformer`.

So what is TF-IDF?

TF-IDF stands for *term frequency-inverse document frequency*, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.

One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions are variants of this simple model.

Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

$$TF(t) = (\text{Number of times term } t \text{ appears in a document}) / (\text{Total number of terms in the document}).$$

IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

$$IDF(t) = \log_e(\text{Total number of documents} / \text{Number of documents with term } t \text{ in it}).$$

See below for a simple example.

Example:

Consider a document containing 100 words wherein the word cat appears 3 times.

The term frequency (i.e., tf) for cat is then $(3 / 100) = 0.03$. Now, assume we have 10 million documents and the word cat appears in one thousand of these. Then, the inverse document frequency (i.e., idf) is calculated as $\log(10,000,000 / 1,000) = 4$. Thus, the Tf-idf weight is the product of these quantities: $0.03 * 4 = 0.12$.

Let's go ahead and see how we can do this in SciKit Learn:

```
In [48]: from sklearn.feature_extraction.text import TfidfTransformer

tfidf_transformer = TfidfTransformer().fit(messages_bow)
tfidf4 = tfidf_transformer.transform(bow4)
print(tfidf4)

(0, 9570)      0.538562626293
(0, 7197)      0.438936565338
(0, 6232)      0.318721689295
(0, 6214)      0.299537997237
(0, 5270)      0.297299574059
(0, 4638)      0.266198019061
(0, 4073)      0.408325899334
```

We'll go ahead and check what is the IDF (inverse document frequency) of the word "u" and of word "university" ?

```
In [50]: print(tfidf_transformer.idf_[bow_transformer.vocabulary_['u']])
print(tfidf_transformer.idf_[bow_transformer.vocabulary_['university']])

3.28005242674
8.5270764989
```

To transform the entire bag-of-words corpus into TF-IDF corpus at once:

```
In [51]: messages_tfidf = tfidf_transformer.transform(messages_bow)
print(messages_tfidf.shape)

(5572, 11444)
```

There are many ways the data can be preprocessed and vectorized. These steps involve feature engineering and building a "pipeline". I encourage you to check out SciKit Learn's documentation on dealing with text data as well as the expansive collection of available papers and books on the general topic of NLP.

Training a model

With messages represented as vectors, we can finally train our spam/ham classifier. Now we can actually use almost any sort of classification algorithms. For a [variety of reasons](#), the Naive Bayes classifier algorithm is a good choice.

We'll be using scikit-learn here, choosing the [Naive Bayes](#) classifier to start with:

```
In [52]: #Creating and
#Training an model with MultinomialNB
from sklearn.naive_bayes import MultinomialNB
spam_detect_model = MultinomialNB().fit(messages_tfidf, messages['label'])
```

Let's try classifying our single random message and checking how we do:

```
In [54]: #Predicting and expectation
print('predicted:', spam_detect_model.predict(tfidf4)[0])
print('expected:', messages.label[3])
```

```
predicted: ham
expected: ham
```

Fantastic! We've developed a model that can attempt to predict spam vs ham classification!

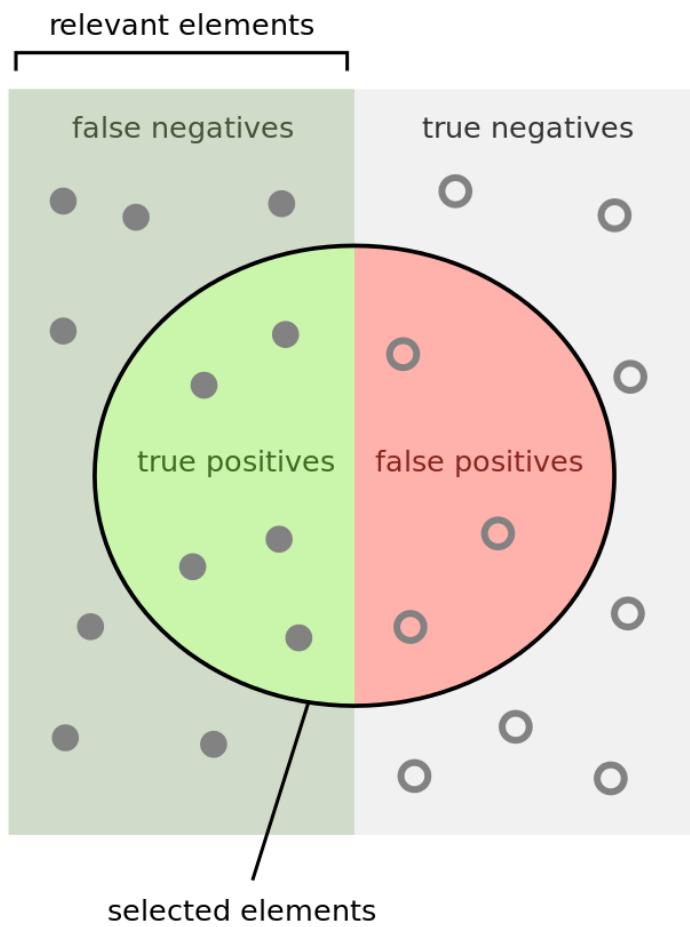
Part 6: Model Evaluation

Now we want to determine how well our model will do overall on the entire dataset. Let's begin by getting all the predictions:

```
In [55]: #Predicting the model
all_predictions = spam_detect_model.predict(messages_tfidf)
print(all_predictions)

['ham' 'ham' 'spam' ..., 'ham' 'ham' 'ham']
```

We can use SciKit Learn's built-in classification report, which returns [precision](#), [recall](#), [f1-score](#), and a column for support (meaning how many cases supported that classification). Check out the links for more detailed info on each of these metrics and the figure below:



How many selected items are relevant?

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

In [56]:

```
#Creating an classification report
from sklearn.metrics import classification_report
print(classification_report(messages['label'], all_predictions))
```

	precision	recall	f1-score	support
ham	0.98	1.00	0.99	4825
spam	1.00	0.85	0.92	747
avg / total	0.98	0.98	0.98	5572

Method 2 with pipeline

There are quite a few possible metrics for evaluating model performance. Which one is the most important depends on the task and the business effects of decisions based off of the model. For example, the cost of mis-predicting "spam" as "ham" is probably much lower than mis-predicting "ham" as "spam".

In the above "evaluation", we evaluated accuracy on the same data we used for training. **You should never actually evaluate on the same dataset you train on!**

Such evaluation tells us nothing about the true predictive power of our model. If we simply remembered each example during training, the accuracy on training data would trivially be 100%, even though we wouldn't be able to classify any new messages.

A proper way is to split the data into a training/test set, where the model only ever sees the **training data** during its model fitting and parameter tuning. The **test data** is never used in any way. This is then our final evaluation on test data is representative of true predictive performance.

Train Test Split

```
In [57]: #Importing train test split
from sklearn.model_selection import train_test_split
#Instead of X and y we use msg and labels
msg_train, msg_test, label_train, label_test = \
train_test_split(messages['message'], messages['label'], test_size=0.2)

print(len(msg_train), len(msg_test), len(msg_train) + len(msg_test))

4457 1115 5572
```

The test size is 20% of the entire dataset (1115 messages out of total 5572), and the training is the rest (4457 out of 5572). Note the default split would have been 30/70.

Creating a Data Pipeline

Let's run our model again and then predict off the test set. We will use SciKit Learn's **pipeline** capabilities to store a pipeline of workflow. This will allow us to set up all the transformations that we will do to the data for future use. Let's see an example of how it works:

```
In [58]: #Creating an pipeline
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('bow', CountVectorizer(analyzer=text_process)), # strings to token integer counts
    ('tfidf', TfidfTransformer()), # integer counts to weighted TF-IDF scores
    ('classifier', MultinomialNB())]) # train on TF-IDF vectors w/ Naive Bayes classifier
```

Now we can directly pass message text data and the pipeline will do our pre-processing for us! We can treat it as a model/estimator API:

```
In [59]: pipeline.fit(msg_train,label_train)
```

```
Out[59]: Pipeline(steps=[('bow', CountVectorizer(analyzer=<function text_process at 0x11e795bf8>,
```

```
binary=False,  
        decode_error='strict', dtype=<class 'numpy.int64'>,  
        encoding='utf-8', input='content', lowercase=True, max_df=1.0,  
        max_features=None, min_df=1, ngram_range=(1, 1), preprocessor=None,...f=False, u  
se_idf=True)), ('classifier', MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True))])
```

```
In [60]: predictions = pipeline.predict(msg_test)
```

```
In [61]: print(classification_report(predictions,label_test))
```

	precision	recall	f1-score	support
ham	1.00	0.96	0.98	1001
spam	0.75	1.00	0.85	114
avg / total	0.97	0.97	0.97	1115

Now we have a classification report for our model on a true testing set! There is a lot more to Natural Language Processing than what we've covered here, and its vast expanse of topic could fill up several college courses! I encourage you to check out the resources below for more information on NLP!

More Resources

Check out the links below for more info on Natural Language Processing:

[NLTK Book Online](#)

[Kaggle Walkthrough](#)

[SciKit Learn's Tutorial](#)