Linear Regression

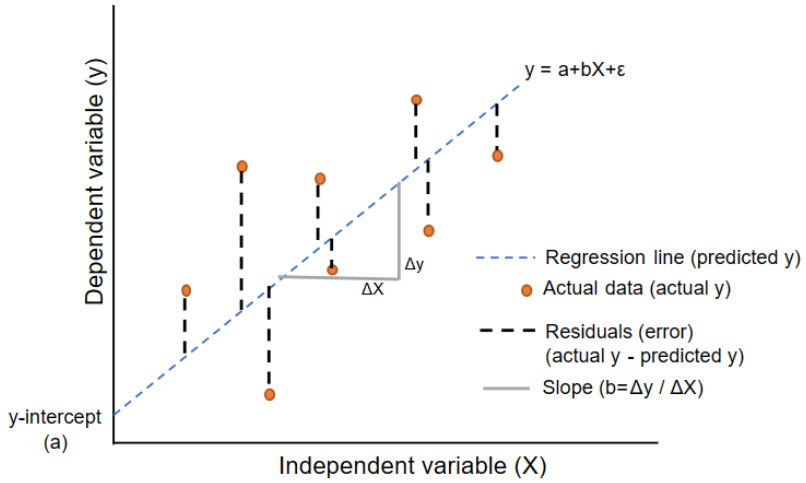
- It is used to estimate real values (cost of house, no. of calls, total sales etc.) based on continuous variable.
- Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation.

Simple Linear Regression

One independent feature and one dependent feature.

Eg: Aim: to create a model, which takes input as height and predict weight.
Dataset : Height, Weight.

Eg: Model: year of experience and salary
Predict: salary based on input salary



$$\hat{Y} = m\,\bar{x} + c$$

Where:- y = Dependent Variable
m = b = Slope
x = Independent Variable
c = a = Intercept

- Difference b/w real points and predicted points is called residuals or errors.
- Based on the training dataset, it finds the best fit line in such a way that the sum of difference between real points and predicted showed be minimum.

First we need to understand why are creating a straight line?
Best fit line is nothing but a equation of straight line.

$$\hat{Y} = m\,\bar{x} + c$$

c = intercept: when x = 0, the line meeting the y-axis that particular point is known as intercept.
m = slope: with the unit movement in the x-axis what is the moment in the y-axis by changing c and m, best fit line will be change.

Mathematical Solution:

Experience	1	2	3	4	5
Salary (LPA)	7	14	15	18	19

$\hat{Y} = m\,\bar{x} + c$ (If slope is + ive)
$\hat{Y} = -\,m\,\bar{x} + c$ (If slope is - ive)

$$b_1 = \frac{\sum_{i=1}^{n}\left((x_i - \bar{x})(y_i - \bar{y})\right)}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

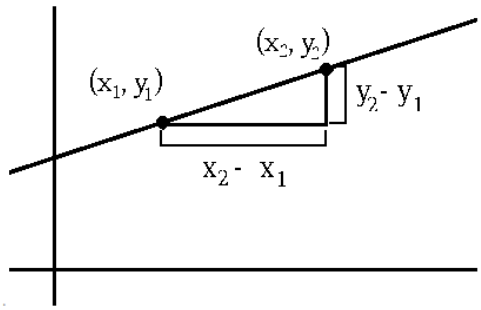
Experience	Salary (LPA)	(x- x̄)	(y- ȳ)	(x- x̄)2	(x- x̄) (y- ȳ)
1	7	-2	-7.6	4	+15.2
2	14	-1	-0.6	1	0.6
3	15	0	0	0	0
4	18	1	3.4	1	3.4
5	19	2	4.4	4	8.8
Σx = 15	Σy			Σ(x- x̄)2 = 10	Σ(x- x̄) (y- ȳ) = 28
x̄ = 3	ȳ = 14.6				

m = 28/10
= 2.8 (slope)

$\hat{Y} = m\,\bar{x} + c$

14.6 = 2.8*3+c
c = 14.6 – 8.4
c = 6.2 (Intercept)

<u>Formula of Slope</u>



$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

m = Δy/Δx

Using Coding:-

In [1]:
```
# Simple Linear Regression
import pandas as pd
```

In [2]:
```
df = pd.DataFrame({'Place':[1,2,3,4,5,],"Profit":[7,14,15,18,19]})
df
```

Out[2]:

	Place	Profit
0	1	7
1	2	14
2	3	15
3	4	18
4	5	19

In [3]:
```
x = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

In [4]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
ypre = model.fit(x,y)
```

```
C:\Users\hp\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for thi
s version of SciPy (detected version 1.23.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [5]:
```
ypre
```

Out[5]:
```
LinearRegression
LinearRegression()
```

In [6]:
```
model.coef_
```

Out[6]:
```
array([2.8])
```

In [7]:
```
model.intercept_
```

Out[7]:
```
6.1999999999999975
```

In [8]:
```
ypre=model.predict(x)
```

In [9]:
```
ypre
```

Out[9]:
```
array([ 9. , 11.8, 14.6, 17.4, 20.2])
```

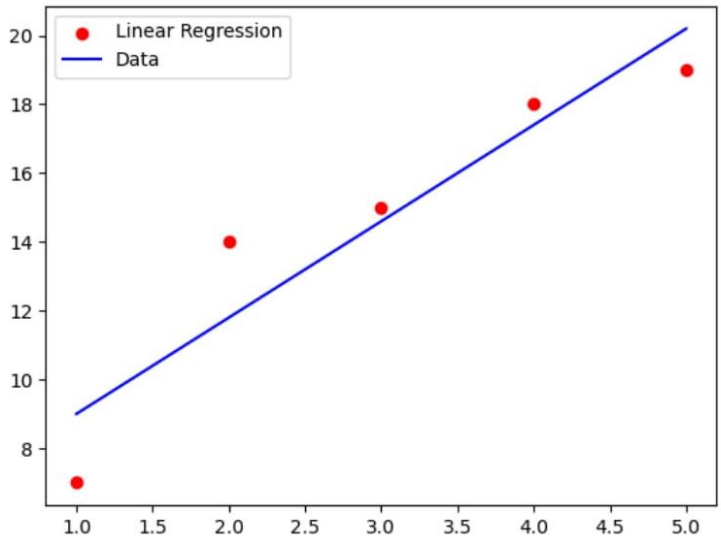
In [10]:
```
check = pd.DataFrame({'Place':df['Place'],'Actual Profit':df['Profit'],"Predicted Profit":ypre})
check
```

Out[10]:

	Place	Actual Profit	Predicted Profit
0	1	7	9.0
1	2	14	11.8
2	3	15	14.6
3	4	18	17.4
4	5	19	20.2

In [11]:
```
from matplotlib import pyplot as plt
plt.scatter(x,y,c='r')
plt.plot(x,ypre,c='b')

# Function add a legend
plt.legend(["Linear Regression", "Data"], loc ="upper left")
```

Out[11]: <matplotlib.legend.Legend at 0x205ea123ca0>



In [12]:
```
# Analyze the performance of the model by calculating mean squared error and R2
import numpy as np
error = y - ypre
se = np.sum(error**2)
print('Squared Error: ', se)
n = np.size(x)
mse = se/n
print('Mean Squared Error: ', mse)

rmse = np.sqrt(mse)
print('Root Mean Square Error: ', rmse)
ymean = np.mean(y)
SSt = np.sum((y - ymean)**2)
R2 = 1- (se/SSt)
print('R2 Score: ', R2)
```

```
Squared Error:  10.800000000000004
Mean Squared Error:  2.160000000000001
Root Mean Square Error:  1.4696938456699071
R2 Score:  0.8789237668161435
```

Multiple Linear Regression

Multiple Linear Regression is one of the important regression algorithms which model the linear relationship between a single dependent continuous variable and (more than 1) independent variable.

- It can be applied to many pactical fields like politics, economics, medical, research works and many different kinds of businesses.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ ... + \beta_i X_i$$

Y : Dependent variable
β_0 : Intercept
β_i : Slope for X_i
X = Independent variable

$$\hat{b}_1 = \frac{(\sum x_2^2)(\sum x_1 y) - (\sum x_1 x_2)(\sum x_2 y)}{(\sum x_1^2)(\sum x_2^2) - (\sum x_1 x_2)^2}$$

$$\hat{b}_2 = \frac{(\sum x_1^2)(\sum x_2 y) - (\sum x_1 x_2)(\sum x_1 y)}{(\sum x_1^2)(\sum x_2^2) - (\sum x_1 x_2)^2}$$

Solving by Hand

Suppose we have the following dataset with one response variable y and two predictor variables X_1 and X.

y	X_1	X_2
140	60	22
155	62	25
159	67	24
179	70	20
192	71	15
200	72	14
212	75	14
215	78	11

Use the following steps to fit a multiple linear regression model to this dataset.

Step 1: Calculate X_1^2, X_2^2, X_1y, X_2y and X_1X_2

	y	X_1	X_2		X_1^2	X_2^2	X_1y	X_2y	X_1X_2
	140	60	22		3600	484	8400	3080	1320
	155	62	25		3844	625	9610	3875	1550
	159	67	24		4489	576	10653	3816	1608
	179	70	20		4900	400	12530	3580	1400
	192	71	15		5041	225	13632	2880	1065
	200	72	14		5184	196	14400	2800	1008
	212	75	14		5625	196	15900	2968	1050
	215	78	11		6084	121	16770	2365	858
Mean	181.5	69.375	18.125	**Sum**	38767	2823	101895	25364	9859
Sum	1452	555	145						

Step 2: Calculate Regression Sums.

Next, make the following regression sum calculations:

- $\Sigma x_1^2 = \Sigma X_1^2 - (\Sigma X_1)^2 / n = 38{,}767 - (555)^2 / 8 =$ **263.875**
- $\Sigma x_2^2 = \Sigma X_2^2 - (\Sigma X_2)^2 / n = 2{,}823 - (145)^2 / 8 =$ **194.875**
- $\Sigma x_1 y = \Sigma X_1 y - (\Sigma X_1 \Sigma y) / n = 101{,}895 - (555*1{,}452) / 8 =$ **1,162.5**
- $\Sigma x_2 y = \Sigma X_2 y - (\Sigma X_2 \Sigma y) / n = 25{,}364 - (145*1{,}452) / 8 =$ **-953.5**
- $\Sigma x_1 x_2 = \Sigma X_1 X_2 - (\Sigma X_1 \Sigma X_2) / n = 9{,}859 - (555*145) / 8 =$ **-200.375**

	y	X_1	X_2
	140	60	22
	155	62	25
	159	67	24
	179	70	20
	192	71	15
	200	72	14
	212	75	14
	215	78	11
Mean	181.5	69.375	18.125
Sum	1452	555	145

	X_1^2	X_2^2	X_1y	X_2y	X_1X_2
	3600	484	8400	3080	1320
	3844	625	9610	3875	1550
	4489	576	10653	3816	1608
	4900	400	12530	3580	1400
	5041	225	13632	2880	1065
	5184	196	14400	2800	1008
	5625	196	15900	2968	1050
	6084	121	16770	2365	858
Sum	38767	2823	101895	25364	9859
Reg Sums	263.875	194.875	1162.5	-953.5	-200.375

Step 3: Calculate b_0, b_1, and b_2.

The formula to calculate b_1 is: $[(\Sigma x_2^2)(\Sigma x_1 y) - (\Sigma x_1 x_2)(\Sigma x_2 y)] / [(\Sigma x_1^2)(\Sigma x_2^2) - (\Sigma x_1 x_2)^2]$

Thus, $\mathbf{b_1} = [(194.875)(1162.5) - (-200.375)(-953.5)] / [(263.875)(194.875) - (-200.375)^2] =$ **3.148**

The formula to calculate b_2 is: $[(\Sigma x_1^2)(\Sigma x_2 y) - (\Sigma x_1 x_2)(\Sigma x_1 y)] / [(\Sigma x_1^2)(\Sigma x_2^2) - (\Sigma x_1 x_2)^2]$

Thus, $\mathbf{b_2} = [(263.875)(-953.5) - (-200.375)(1152.5)] / [(263.875)(194.875) - (-200.375)^2] =$ **-1.656**

The formula to calculate b_0 is: $y - b_1X_1 - b_2X_2$

Thus, $\mathbf{b_0} = 181.5 - 3.148(69.375) - (-1.656)(18.125) =$ **-6.867**

Step 5: Place b_0, b_1, and b_2 in the estimated linear regression equation.

The estimated linear regression equation is: $\hat{y} = b_0 + b_1*x_1 + b_2*x_2$

In our example, it is $\mathbf{\hat{y} = -6.867 + 3.148x_1 - 1.656x_2}$

How to Interpret a Multiple Linear Regression Equation

Here is how to interpret this estimated linear regression equation: $\hat{y} = -6.867 + 3.148x_1 - 1.656x_2$

b_0 = -6.867. When both predictor variables are equal to zero, the mean value for y is -6.867.

b_1 = 3.148. A one unit increase in x_1 is associated with a 3.148 unit increase in y, on average, assuming x_2 is held constant.

b_2 = -1.656. A one unit increase in x_2 is associated with a 1.656 unit decrease in y, on average, assuming x_1 is held constant.

Steps to implement Multivariate regression:-

1. Feature selection

The selection of features plays the most important role in multivariate regression.

Finding the feature that is needed for finding which variable is dependent on this feature.

2. Normalizing Features

For better analysis features are need to be scaled to get them into a specific range. We can also change the value of each feature.

3. Select Loss function and Hypothesis

The loss function calculates the loss when the hypothesis predicts the wrong value.

And hypothesis means predicted value from the feature variable.

4. Set Hypothesis Parameters

Set the hypothesis parameter that can reduce the loss function and can predict.

5. Minimize the Loss Function

Minimizing the loss by using some lose minimization algorithm and use it over the dataset which can help to adjust the hypothesis parameters. Once the loss is minimized then it can be used for prediction.

There are many algorithms that can be used for reducing the loss such as gradient descent.

6. Test the hypothesis function

Check the hypothesis function how correct it predicting values, test it on test data.

In [13]:
```
import pandas as pd
```

In [14]:
```
data = pd.read_csv('50_Startups.csv')
data.head()
```

Out[14]:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

In [15]:
```
data.shape
```

Out[15]:
```
(50, 5)
```

In [16]:
```
# All the variables in the numeric form so it will change the catoragical data in numeric form.
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [17]:
```
data['State']=le.fit_transform(data['State'])
```

In [18]:
```
data.tail()
```

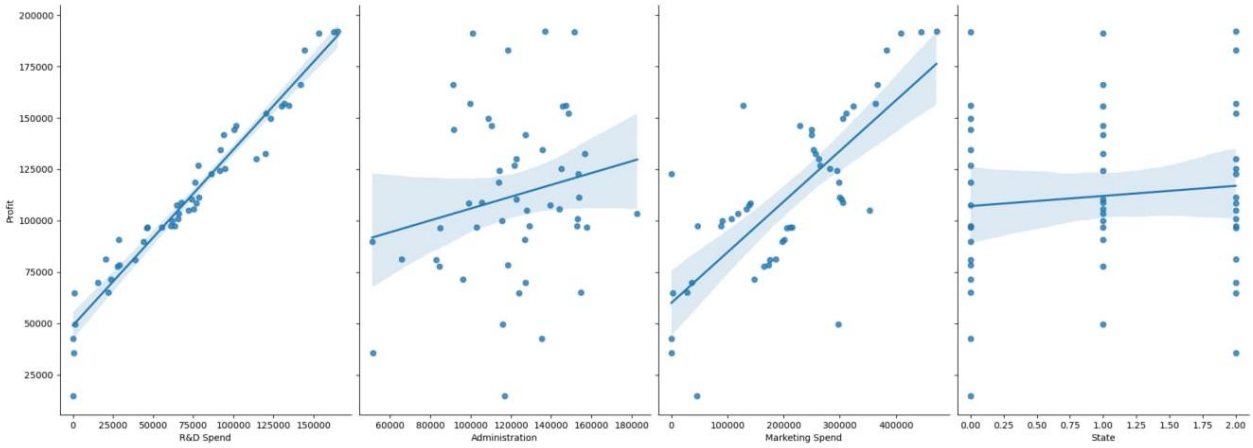
Out[18]:

	R&D Spend	Administration	Marketing Spend	State	Profit
45	1000.23	124153.04	1903.93	2	64926.08
46	1315.46	115816.21	297114.46	1	49490.75
47	0.00	135426.92	0.00	0	42559.73
48	542.05	51743.15	0.00	2	35673.41
49	0.00	116983.80	45173.06	0	14681.40

In [19]:
```
import seaborn as sns
%matplotlib inline
sns.pairplot(data,x_vars=['R&D Spend', 'Administration', 'Marketing Spend', 'State'], y_vars="Profit",
             size=7,aspect=0.7,kind='reg')
```

```
C:\Users\hp\anaconda3\lib\site-packages\seaborn\axisgrid.py:2095: UserWarning: The `size` parameter has been renamed to `height`; pl
ease update your code.
  warnings.warn(msg, UserWarning)
```

Out[19]:
```
<seaborn.axisgrid.PairGrid at 0x205ec184820>
```



In [20]:
```
# divided data into independent and dependent variable.
x = data.iloc[:,:-1]
y = data.iloc[:,-1]
```

In [21]:
```
# spliting the dataset into training set and test set.
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)
```

In [22]:
```
# import the regression model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

In [23]:
```
# fitting multi-regression in train test.
model.fit(x_train,y_train)
```

Out[23]:
```
▾ LinearRegression

LinearRegression()
```

In [24]:
```
# preadcting the test set result
ypre = model.predict(x_test)
```

In [25]:
```
ypre
```

Out[25]:
```
array([150032.53333639, 112810.92048286, 119789.51493002, 134519.22150981,
       129389.03112571, 112204.72469995, 116057.74096167,  51652.30687496,
       117577.907593  , 178811.87143387])
```

In [26]:
```
# comparing predicted profit with actual profit.
check = pd.DataFrame({'actual':y_test,'predicted':ypre})
check
```

Out[26]:

	actual	predicted
14	132602.65	150032.533336
24	108552.04	112810.920483
21	111313.02	119789.514930
10	146121.95	134519.221510
12	141585.52	129389.031126
26	105733.54	112204.724700
27	105008.31	116057.740962
49	14681.40	51652.306875
16	126992.93	117577.907593
2	191050.39	178811.871434

In [27]:

```
# find the MAE,MSE,RMSE,R2 Score,Adjusted R2 Score
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score

print('MAE: ',mean_absolute_error(y_test,ypre))

print('MSE:',mean_squared_error(y_test,ypre))

print('RMSE:',np.sqrt(mean_squared_error(y_test,ypre)))

print('R2 Score:',r2_score(y_test,ypre))
```

```
MAE:  13010.95396234671
MSE: 244640560.64363137
RMSE: 15640.98975907955
R2 Score: 0.8648710271208794
```

Polynomial Regression

A regression equation is a polynomial regression equation if the power of the independent variable is more than 1. The equation below represents a polynomial equation.

$$y= b_0+b_1x_1+ b_2x_1^2+ b_2x_1^3+..... b_nx_1^n$$

In this regression technique, the best fit line is not a straight line. It is rather a curve that fits into the data points.

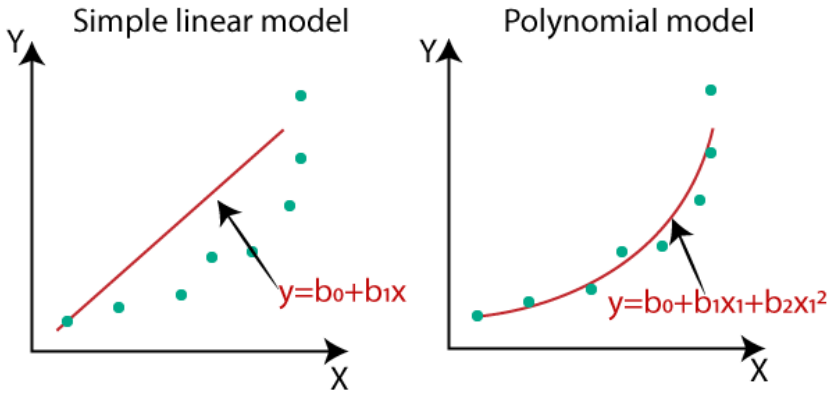
- It is also called the special case of Multiple Linear Regression in ML. Because we add some polynomial terms to the Multiple Linear regression equation to convert it into Polynomial Regression.
- It is a linear model with some modification in order to increase the accuracy.
- The dataset used in Polynomial regression for training is of non-linear nature.
- It makes use of a linear regression model to fit the complicated and non-linear functions and datasets.

Hence, *"In Polynomial regression, the original features are converted into Polynomial features of required degree (2,3,..,n) and then modeled using a linear model."*

Need for Polynomial Regression:

The need of Polynomial Regression in ML can be understood in the below points:

- If we apply a linear model on a **linear dataset**, then it provides us a good result as we have seen in Simple Linear Regression, but if we apply the same model without any modification on a **non-linear dataset**, then it will produce a drastic output. Due to which loss function will increase, the error rate will be high, and accuracy will be decreased.
- So for such cases, **where data points are arranged in a non-linear fashion, we need the Polynomial Regression model**. We can understand it in a better way using the below comparison diagram of the linear dataset and non-linear dataset.



- In the above image, we have taken a dataset which is arranged non-linearly. So if we try to cover it with a linear model, then we can clearly see that it hardly covers any data point. On the other hand, a curve is suitable to cover most of the data points, which is of the Polynomial model.
- Hence, *if the datasets are arranged in a non-linear fashion, then we should use the Polynomial Regression model instead of Simple Linear Regression.*

Note: *A Polynomial Regression algorithm is also called Polynomial Linear Regression because it does not depend on the variables, instead, it depends on the coefficients, which are arranged in a linear fashion.*

Equation of the Polynomial Regression Model:

Simple Linear Regression equation: $y = b_0+b_1x$ .....(a)

Multiple Linear Regression equation: $y= b_0+b_1x+ b_2x_2+ b_3x_3+....+ b_nx_n$ .....(b)

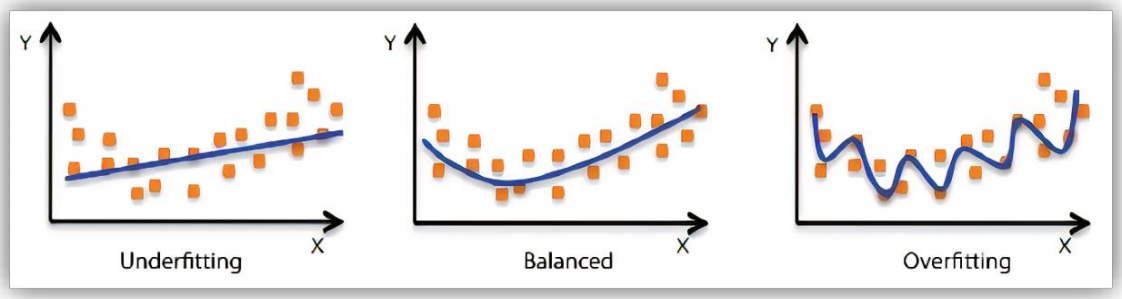
Polynomial Regression equation: $y= b_0+b_1x + b_2x^2+ b_3x^3+....+ b_nx^n$ .....(c)

When we compare the above three equations, we can clearly see that all three equations are Polynomial equations but differ by the degree of variables. The Simple and Multiple Linear equations are also Polynomial equations with a single degree, and the Polynomial regression equation is Linear equation with the nth degree. So if we add a degree to our linear equations, then it will be converted into Polynomial Linear equations.

Note: *To better understand Polynomial Regression, you must have knowledge of Simple Linear Regression.*

Important Points:-

- While there might be a temptation to fit a higher degree polynomial to get lower error, this can result in over-fitting. Always plot the relationships to see the fit and focus on making sure that the curve fits the nature of the nature problem.
- Here is an example of how ploting can help.



- Espically look out for curve towards the neds and see whether those shapes and trends make sense. Higher polynomials can end up producing wired results on extrapolation.

Implementation of Polynomial Regression using Python:

Here we will implement the Polynomial Regression using Python. We will understand it by comparing Polynomial Regression model with the Simple Linear Regression model. So first, let's understand the problem for which we are going to build the model.

Steps for Polynomial Regression:

The main steps involved in Polynomial Regression are given below:

- Data Pre-processing
- Build a Linear Regression model and fit it to the dataset
- Build a Polynomial Regression model and fit it to the dataset
- Visualize the result for Linear Regression and Polynomial Regression model.
- Predicting the output.

Note: *Here, we will build the Linear regression model as well as Polynomial Regression to see the results between the predictions. And Linear regression model is for reference.*

Data Pre-processing Step:

The data pre-processing step will remain the same as in previous regression models, except for some changes. In the Polynomial Regression model, we will not use feature scaling, and also we will not split our dataset into training and test set. It has two reasons:

- The dataset contains very less information which is not suitable to divide it into a test and training set, else our model will not be able to find the correlations between the salaries and levels.
- In this model, we want very accurate predictions for salary, so the model should have enough information.

In [23]:
```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

In [24]:
```
data_set = pd.DataFrame({'Position':["Business Analyst","Junior Consultant","Senior Consultant","Manager","Country Manager",
                         "Region Manager","Partner","Senior Partner","C-level","CEO"],
                         "Level":[1,2,3,4,5,6,7,8,9,10],
                         "Salary":[45000,50000,60000,80000,110000,150000,200000,300000,500000,1000000]})
data_set
```

Out[24]:

	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000
8	C-level	9	500000
9	CEO	10	1000000

In [25]:
```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, 1:2].values
y= data_set.iloc[:, 2].values
```

In [26]:
```
#Fitting the Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
lin_regs= LinearRegression()
lin_regs.fit(x,y)
```

Out[26]:
```
LinearRegression()
```

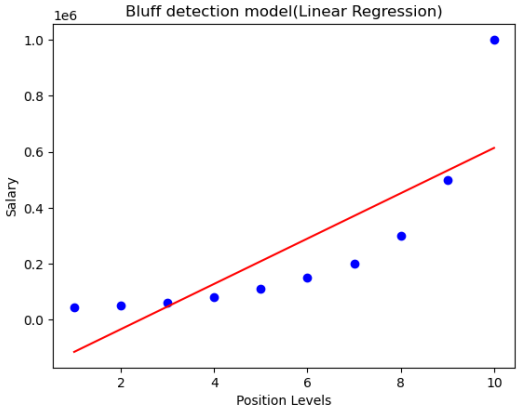
In [27]:
```
#Fitting the Polynomial regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_regs= PolynomialFeatures(degree= 2)
x_poly= poly_regs.fit_transform(x)
lin_reg_2 =LinearRegression()
lin_reg_2.fit(x_poly, y)
```

Out[27]:

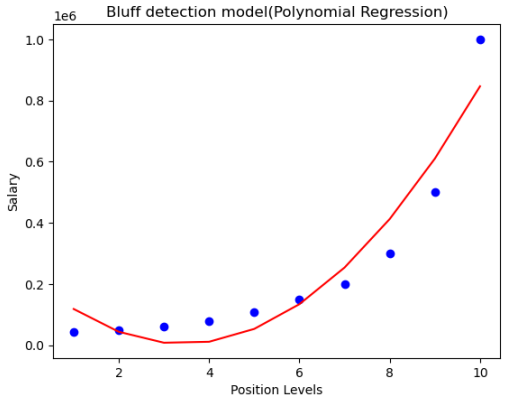
LinearRegression

LinearRegression()

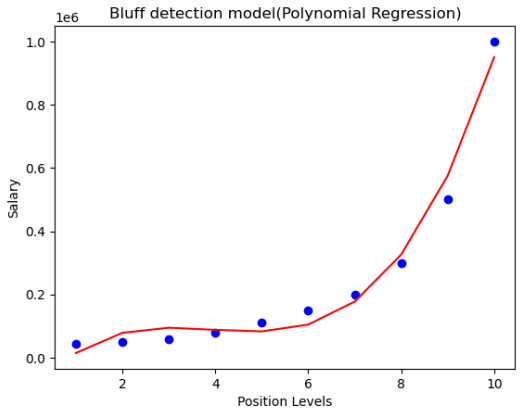
In [28]:
```
#Visulaizing the result for Linear Regression model
mtp.scatter(x,y,color="blue")
mtp.plot(x,lin_regs.predict(x), color="red")
mtp.title("Bluff detection model(Linear Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```



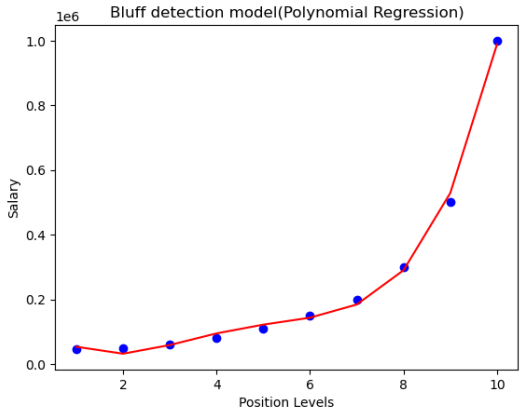
In [29]:
```
#Visulaizing the result for Polynomial Regression
mtp.scatter(x,y,color="blue")
mtp.plot(x, lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")
mtp.title("Bluff detection model(Polynomial Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```



In [31]:
```
#Fitting the Polynomial regression to the dataset for degree 3
poly_regs= PolynomialFeatures(degree= 3)
x_poly= poly_regs.fit_transform(x)
lin_reg_2 =LinearRegression()
lin_reg_2.fit(x_poly, y)

mtp.scatter(x,y,color="blue")
mtp.plot(x, lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")
mtp.title("Bluff detection model(Polynomial Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```



In [32]:
```
#Fitting the Polynomial regression to the dataset for degree 4
poly_regs= PolynomialFeatures(degree= 4)
x_poly= poly_regs.fit_transform(x)
lin_reg_2 =LinearRegression()
lin_reg_2.fit(x_poly, y)

mtp.scatter(x,y,color="blue")
mtp.plot(x, lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")
mtp.title("Bluff detection model(Polynomial Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```



In [33]:
```
#Predicting the final result with the Linear Regression model:
lin_pred = lin_regs.predict([[6.5]])
print(lin_pred)
```

```
[330378.78787879]
```

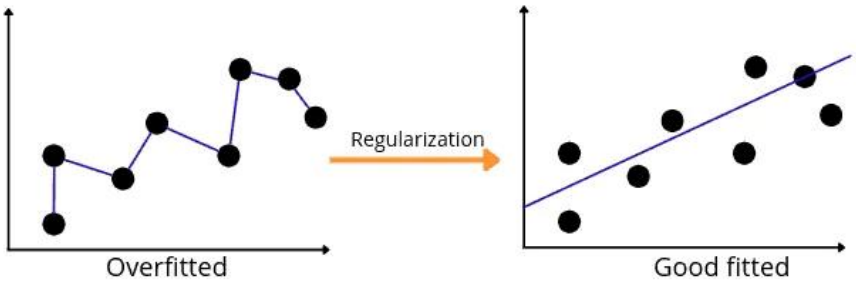
In [34]:
```
#Predicting the final result with the Polynomial Regression model:
poly_pred = lin_reg_2.predict(poly_regs.fit_transform([[6.5]]))
print(poly_pred)
```

```
[158862.45265153]
```

Regularization

Regularization is one of the most important concepts of machine learning. It is a technique to prevent the model from overfitting by adding extra information to it.

Regularization refers to techniques used to calibrate machine learning models to minimize the adjusted loss function and prevent overfitting or underfitting. Using regularization, we can fit our machine learning model appropriately on a given test set and reduce its errors.



Let us now learn how the L1 (the Lasso regression) and L2 (the Ridge regression) help to regularize the model.

Sometimes the machine learning model performs well with the training data but does not perform well with the test data. It means the model is not able to predict the output when deals with unseen data by introducing noise in the output, and hence the model is called overfitted. This problem can be deal with the help of a regularization technique.

This technique can be used in such a way that it will allow to maintain all variables or features in the model by reducing the magnitude of the variables. Hence, it maintains accuracy as well as a generalization of the model.

It mainly regularizes or reduces the coefficient of features toward zero. In simple words, "*In regularization technique, we reduce the magnitude of the features by keeping the same number of features.*"

Regularization works by adding a penalty or complexity term to the complex model. Let's consider the simple linear regression equation:

$$y= \beta0+\beta1x1+\beta2x2+\beta3x3+\cdots+\beta nxn +b$$

In the above equation, Y represents the value to be predicted

X1, X2, …Xn are the features for Y.

β0,β1,…..βn are the weights or magnitude attached to the features, respectively. Here represents the bias of the model, and b represents the intercept.

Linear regression models try to optimize the β0 and b to minimize the cost function. The equation for the cost function for the linear model is given below:

$$\sum_{i=1}^{M}\left(y_i - y'_i\right)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{n}\beta_j * Xij\right)^2$$

Now, we will add a loss function and optimize parameter to make the model that can predict the accurate value of Y. The loss function for the linear regression is called as **RSS or Residual sum of squares.**

Techniques of Regularization

There are mainly two types of regularization techniques, which are given below:

Ridge Regression

Lasso Regression

Ridge Regression

- Ridge regression is one of the types of linear regression in which a small amount of bias is introduced so that we can get better long-term predictions.
- Ridge regression is a regularization technique, which is used to reduce the complexity of the model. It is also called as **L2 regularization**.
- In this technique, the cost function is altered by adding the penalty term to it. The amount of bias added to the model is called **Ridge Regression penalty**. We can calculate it by multiplying with the lambda to the squared weight of each individual feature.
- As we know, the simple linear equation uses the following mathematical equation to find the best-fitted line:

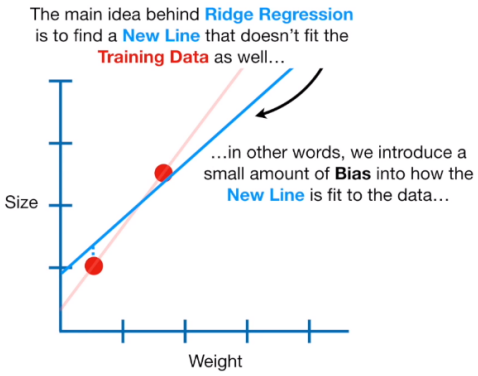
$$y = \text{y-intercept} + \text{slope}*x$$

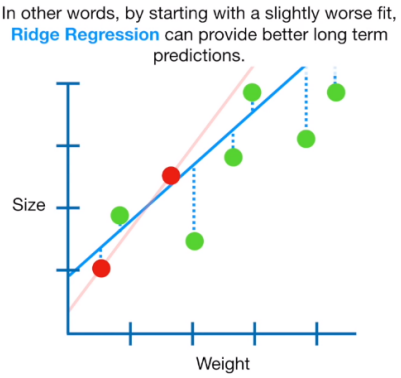
- While the Ridge regression uses a slightly modified equation with the penalty term to find the new best-fitted line to introduce a small bias and reduce variance. The penalty term, also known as the penalty function or cost function, contains lambda and slope square, as shown below.

$$y = (\text{y-intercept} + \text{slope}*x) + (\text{lamda} * \text{slope}^2)$$

$$\sum_{i=1}^{M}(y_i - y'_i)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{n}\beta_j * x_{ij}\right)^2 + \lambda\sum_{j=0}^{n}\beta_j{}^2$$

- In the above equation, the penalty term regularizes the coefficients of the model, and hence ridge regression reduces the amplitudes of the coefficients that decreases the complexity of the model.
- As we can see from the above equation, if the values of **λ tend to zero, the equation becomes the cost function of the linear regression model.** Hence, for the minimum value of λ, the model will resemble the linear regression model.
- A general linear or polynomial regression will fail if there is high collinearity between the independent variables, so to solve such problems, Ridge regression can be used.
- It helps to solve the problems if we have more parameters than samples.





Lasso Regression:

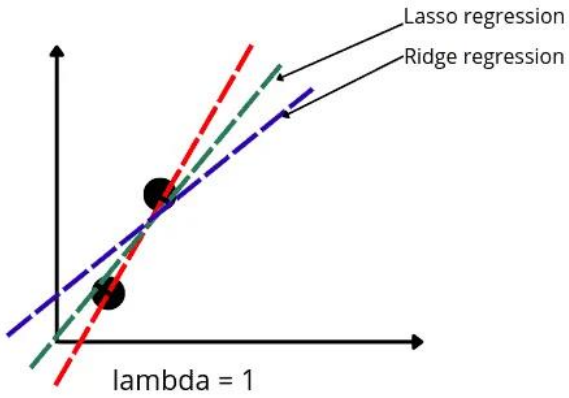
- Lasso regression is another regularization technique to reduce the complexity of the model. It stands for **Least Absolute and Selection Operator.**
- It is similar to the Ridge Regression except that the penalty term contains only the absolute weights instead of a square of weights.
- Since it takes absolute values, hence, it can shrink the slope to 0, whereas Ridge Regression can only shrink it near to 0.
- It is also called as **L1 regularization.** The equation for the cost function of Lasso regression will be:

y = (y-intercept+slope*x) + (lamda*|slope|)

$$\sum_{i=1}^{M}(y_i - y'_i)^2 = \sum_{i=1}^{M}\left(y_i - \sum_{j=0}^{n}\beta_j * x_{ij}\right)^2 + \lambda\sum_{j=0}^{n}|\beta_j|$$

- Some of the features in this technique are completely neglected for model evaluation.
- Hence, the Lasso regression can help us to reduce the overfitting in the model as well as the feature selection.

Note that instead of squaring the slope for the penalty term, the Lasso regression takes the absolute value of the slope (The absolute value will return all positive values). That means for the same value of lambda, the Lasso regression will produce less bias than the Ridge regression, as shown below:



Key Difference between Ridge Regression and Lasso Regression

- **Ridge regression** is mostly used to reduce the overfitting in the model, and it includes all the features present in the model. It reduces the complexity of the model by shrinking the coefficients.
- **Lasso regression** helps to reduce the overfitting in the model as well as feature selection.

Implementation in Jupyter

In [35]:
```
# Import standard modules
import io
import urllib3

# importing the Pandas module
import pandas as pd

# Download dataset from our site
http = urllib3.PoolManager()
r = http.request('GET', 'https://hands-on.cloud/wp-content/uploads/2022/04/Dushanbe_house.csv')

# importing the dataset
Dushanbe = pd.read_csv(io.StringIO(r.data.decode('utf-8')))

# get dataset demo
Dushanbe.head()
```

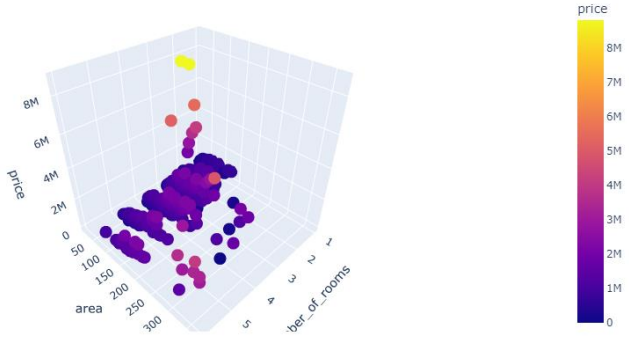
Out[35]:

	Unnamed: 0	number_of_rooms	floor	area	latitude	longitude	price
0	0	1	1	58.0	38.585834	68.793715	330000
1	1	1	14	68.0	38.522254	68.749918	340000
2	2	3	8	50.0	NaN	NaN	700000
3	3	3	14	84.0	38.520835	68.747908	700000
4	4	3	3	83.0	38.564374	68.739419	415000

In [36]:
```
# removing null values
Dushanbe.dropna(axis=0, inplace=True)

# display null values if exist
Dushanbe.isnull().sum()
```

Out[36]:
```
Unnamed: 0         0
number_of_rooms    0
floor              0
area               0
latitude           0
longitude          0
price              0
dtype: int64
```

In [37]:
```
# importing the plotly module
import plotly.express as px

# plotting 3-d plot
fig = px.scatter_3d(Dushanbe, x='number_of_rooms', y='area', z='price',
                    color='price')
fig.show()
```



Implementing Ridge regression

In [39]:
```
# taking the columns from the dataset
columns = Dushanbe.columns

# storing the input and output variables
Inputs = Dushanbe[columns[0:-1]]

#dependent variable
outputs = Dushanbe[columns[-1]]
```

In [40]:
```
# importing the module
from sklearn.model_selection import train_test_split

# splitting into test data and traind data for ridge regression
X_train, X_test, y_train, y_test = train_test_split(Inputs, outputs, test_size=0.25, random_state=42)
```

In [41]:
```
from sklearn.linear_model import Ridge

# alpha parameter 0.9 and initializing ridge regression
model = Ridge(alpha=0.9)

# ridge function
model.fit(X_train, y_train)
```

Out[41]:

Ridge

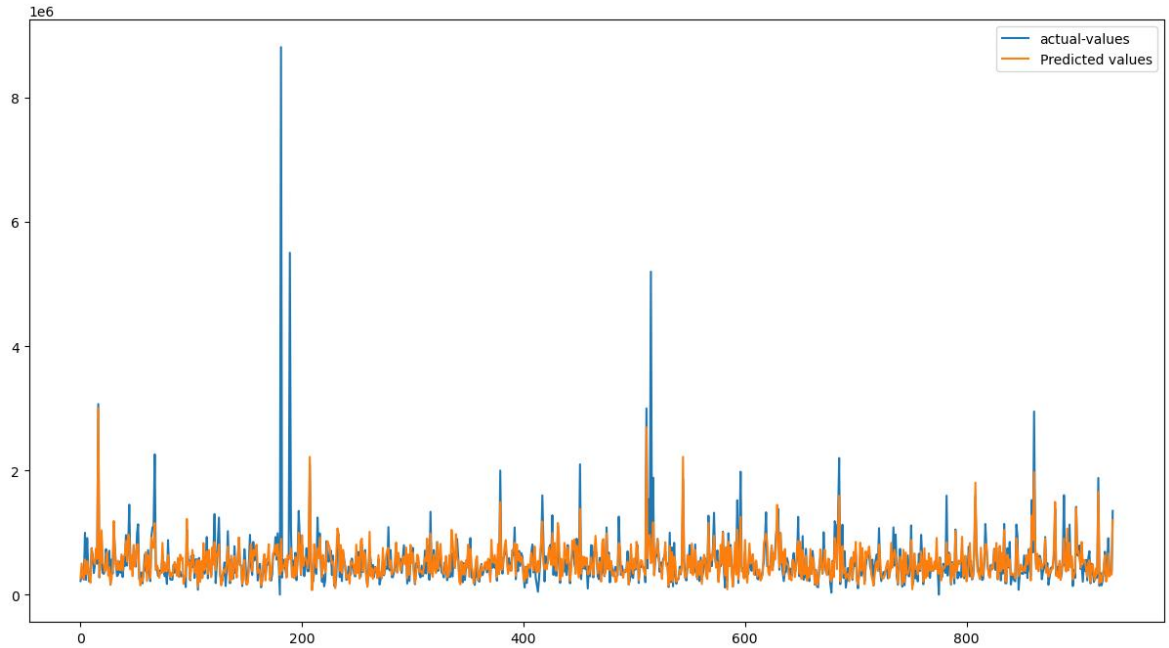
Ridge(alpha=0.9)

In [42]:
```
# predictive models of ridge regression models
y_pred = model.predict(X_test)
```

In [43]:
```
# Importing the required module
from sklearn.metrics import r2_score

# Evaluating model performance
print('R-square score is :', r2_score(y_test, y_pred))
```

```
R-square score is : 0.3734242547708655
```

In [44]:
```
# importing the module
import matplotlib.pyplot as plt

# fitting the size of the plot
plt.figure(figsize=(15, 8))

# plotting the graphs for actual-value and predicted values
plt.plot([i for i in range(len(y_test))],y_test, label="actual-values")
plt.plot([i for i in range(len(y_test))],y_pred, label="Predicted values")

# showing the plotting of predictive modelling technique
plt.legend()
plt.show()
```



Implementing Lasso regression

In [45]:
```
# lasso regression implementation
from sklearn.linear_model import Lasso

# lasso regression select initialization
lasso_model = Lasso(alpha=0.9)

# training the lasso regression model
lasso_model.fit(X_train, y_train)
```

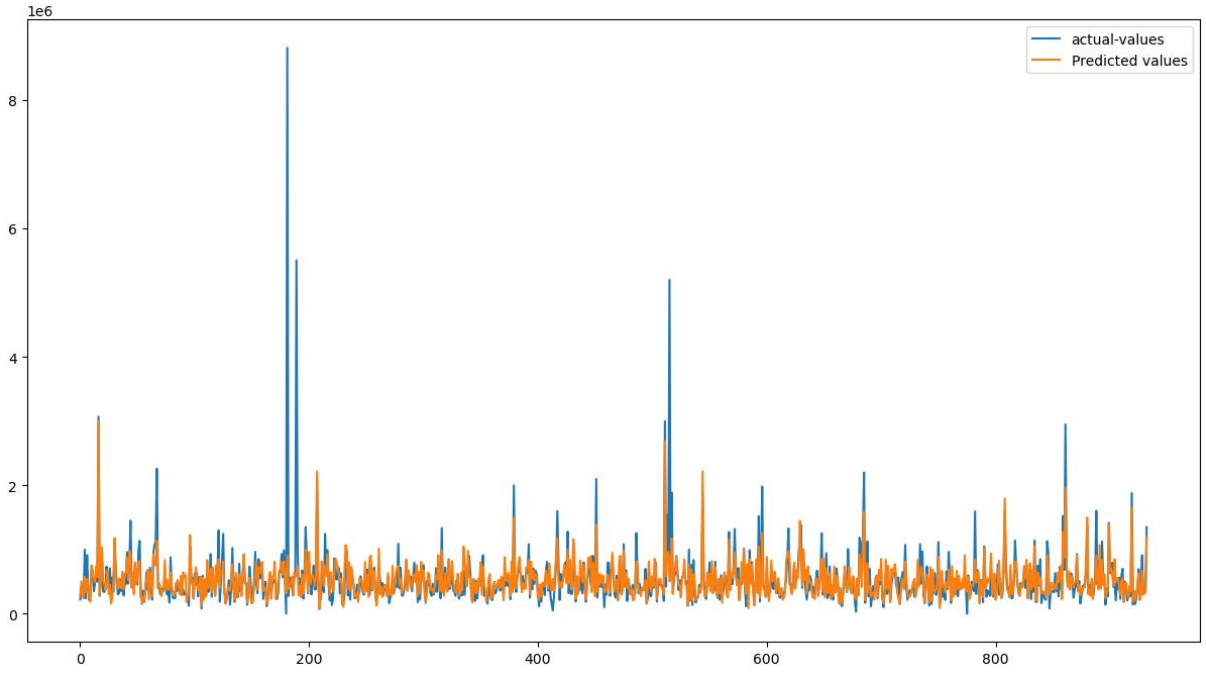
Out[45]:
```
Lasso
Lasso(alpha=0.9)
```

In [46]:
```
# predictive model of lasso regression with test data
lasso_predictions = lasso_model.predict(X_test)
```

In [47]:
```
# Evaluating model performance to see accurate model
print('R-square score is :', r2_score(y_test, lasso_predictions))
```

```
R-square score is : 0.3742383050894751
```

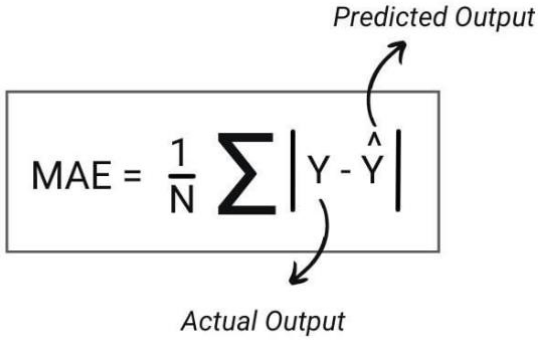
In [48]:
```
# fitting the size of the plot
plt.figure(figsize=(15, 8))

# plotting the graphs for observed value and real values
plt.plot([i for i in range(len(y_test))],y_test, label="actual-values")
plt.plot([i for i in range(len(y_test))],lasso_predictions, label="Predicted values")

# showing the plotting of lasso regression
plt.legend()
plt.show()
```



Regression Metrics

Mean Absolute Error

- MAE is a very simple metric which calculates the absolute difference between actual and predicted values.
- Let's take an example you have input data and output data and use Linear Regression, which draws a best-fit line.
- Now you have to find the MAE of your model which is basically a mistake made by the model known as an error.
- Now find the difference between the actual value and predicted value that is an absolute error but we have to find the mean absolute of the complete dataset.
- So, sum all the errors and divide them by a total number of observation and this is MAE. And we aim to get a minimum MAE because this is a loss.



Advantage of MAE

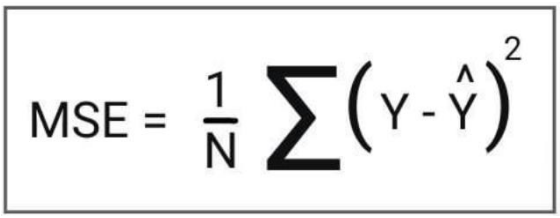
- The MAE you get is in the same unit as the output variable.
- It is most Robust to outliers.

Disadvantage of MAE

- The graph of MAE is not differentiable so we have to apply various optimizers like Gradient descent which can be differentiable.

Mean Square Error

- MSE is a most used and very simple metric with a little bit of change in mean absolute error.
- Mean squared error states that finding the squared difference between actual and predicted value.
- MSE represents the squared distance between actual and predicted values. We perform squared to avoid the cancellation of negative terms and it is the benefit of MSE.



Root Mean Square Error

- As RMSE is clear by the name itself, that it is a simple square root of mean squared error.

$$RMSE = \sqrt{\frac{1}{N}\sum\left(Y - \hat{Y}\right)}$$

Advantages of RMSE

- The output value you get is in the same unit as the required output variable which makes interpretation of loss easy.

Disadvantages of RMSE

- It is not that robust to outliers as compared to MAE for performing RMSE we have to Numpy square root function over MSE.
- Most of the time people use RMSE as an evaluation metric and mostly when you are working with deep learning techniques the most preferred metric is RMSE.

R Squared (R2)

- R2 score is a metric that tells the performance of your model, not the loss in an absolute sense that how many wells did your model perform.
- In contrast, MAE and MSE depend on the context as we have seen whereas the R2 score is independent of context.
- With help of R squared we have a baseline model to compare a model which none of the other metric provides.
- The same we have in classification problems which we cam a threshold which is fixwd at 0.5.
- So basically R2 squared calculates how much regression line is better than mean line.
- R2 squared is also known as Coefficant of Determination or sometimes also known as Goodness of fit.

$$\textbf{R2 Squared} = 1 - \frac{SSr}{SSm}$$

SSr = Squared sum error of regression line

SSm = Squared sum error of mean line

How will you interpret the R2 score?

- Suppose if the R2 score is zero then the above regression line by mean line is equal means 1 so 1-1 is zero.
- So, in this case, both lines are overlapping means model performance is worst, It is not capable to take advantage of the output column.
- Now the second case is when the R2 score is 1, which means when the division term is zero and it will happen when the regression line does not make any mistake, it is perfect. In the real world, it is not possible.
- So we can conclude that as our regression line moves towards perfection, the R2 score moves towards one. And the model performance improves.
- The normal case is when the R2 score is between zero and one like 0.8 which means your model is capable to explain 80% of the variance of data.

Adjusted R Squared

- The disadvantage of the R2 score is while adding new features in data the R2 score starts increasing or remains constant but it never decreases because It assumes that while adding more data variance of data increases.
- But the problem is when we add an irrelevant feature in the datasetthen at that time R2 sometimes starts increasing which is incorrect.
- Hence, to control this situation Adjusted R Squared came into existence.

$$R_a^2 = 1 - \left[\left(\frac{n-1}{n-k-1}\right) \times (1 - R^2)\right]$$

where:
n = number of observations
k = number of independent variables
R_a^2 = adjusted R^2

- Now as K increases by adding some features so the denominator will decreases, n-1 will remain constant.
- R2 score will remain constant or will increases slightly so the complete answer will increase and when we subtract this form one then the resultant score will decrease. So this is the case when we add an irrelevant feature in the dataset.
- If we add a relevant feture then the R2 score will increase and 1-R2 will decreases heavily and the denominator will also decrease so the complete term decreases, and on subtracting from one the score increases.
- Hence, this metric becomes one of the most important metrics to use during the evaluation of the model.

MAE, MSE, RMSE, R2 Score, Adjusted R2 score

In [1]:
```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
```

In [2]:
```
data = pd.read_csv("placements.txt",delimiter=",")
```

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

Out[3]:

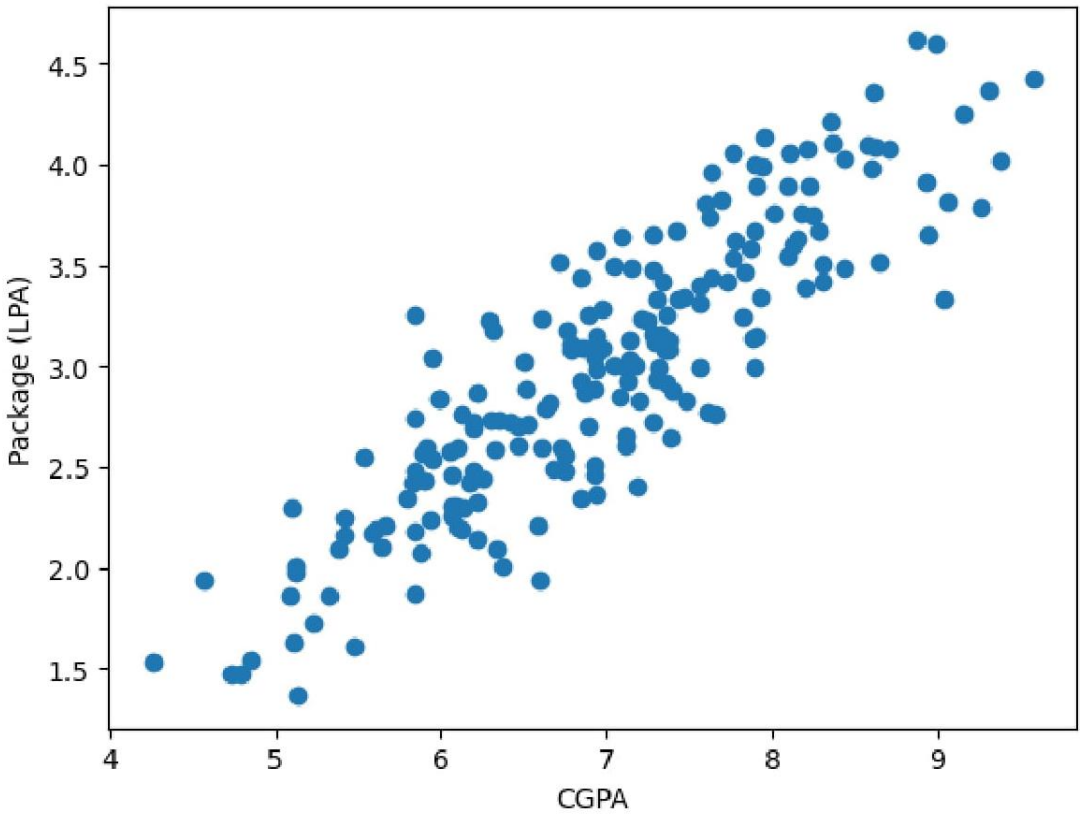
	cgpa	package
0	6.89	3.26
1	5.12	1.98
2	7.82	3.25
3	7.42	3.67
4	6.94	3.57

In [4]:
```
data.shape
```

Out[4]:
```
(200, 2)
```

In [5]:
```
plt.scatter(data['cgpa'],data['package'])
plt.xlabel('CGPA')
plt.ylabel('Package (LPA)')
```

Out[5]:
```
Text(0, 0.5, 'Package (LPA)')
```



In [6]:
```
x = data.iloc[:,0:1]
y = data.iloc[:,-1]
```

In [7]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=4)
```

```
C:\Users\hp\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A NumP
y version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected ver
sion 1.23.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

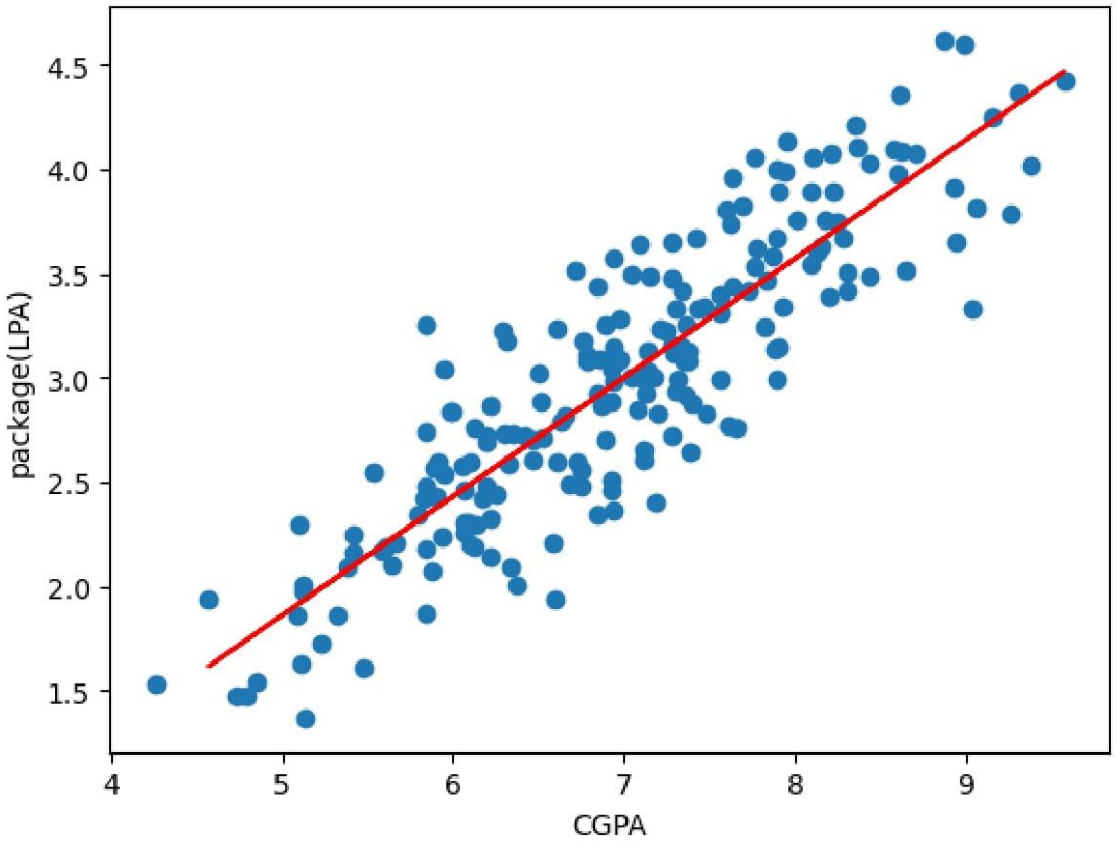
In [8]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

In [9]:
```
lr.fit(x,y)
```

Out[9]:
```
▾ LinearRegression

LinearRegression()
```

In [10]:
```
plt.scatter(data['cgpa'],data['package'])
plt.plot(x_train,lr.predict(x_train),c='r')
plt.xlabel('CGPA')
plt.ylabel('package(LPA)')
```

Out[10]:
```
Text(0, 0.5, 'package(LPA)')
```



In [11]:
```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
```

In [12]:
```
y_pre = lr.predict(x_test)
```

In [14]:
```
df = pd.DataFrame({'Test Value':y_test,'Predicted Value':y_pre})
df
```

	Test Value	Predicted Value
100	4.14	3.542573
83	3.49	3.821673
181	3.89	3.696362
71	3.34	3.269169

In [15]:
```
# Analyze the performance of the model by calculating mean squared error and R2

print('MAE:',mean_absolute_error(y_test,y_pre))

print('MSE:',mean_squared_error(y_test,y_pre))

print('RMSE:',np.sqrt(mean_squared_error(y_test,y_pre)))

print('R2 Score:',r2_score(y_test,y_pre))
```

```
MAE: 0.26913865091404976
MSE: 0.11320931085481958
RMSE: 0.3364659133624379
R2 Score: 0.8001010698130744
```

In [16]:
```
x_test.shape
```

Out[16]:
```
(40, 1)
```

In [17]:
```
# Adjusted R2 score
x_test.shape
1 - ((1-r2_score(y_test,y_pre))*(40-1)/(40-1-1))
```

Out[17]:
```
0.7948405716502606
```

In [18]:
```
# by formula and programing

import numpy as np
error = y_test - y_pre
se = np.sum(error**2)
print('Squared error is ', se)
n = np.size(x)
mse = se/n
print('Mean squared error is ', mse)

rmse = np.sqrt(mse)
print('Root mean square error is ', rmse)
ymean = np.mean(y_test)
SSt = np.sum((y_test - ymean)**2)
R2 = 1- (se/SSt)
print('R2 score is ', R2)
```

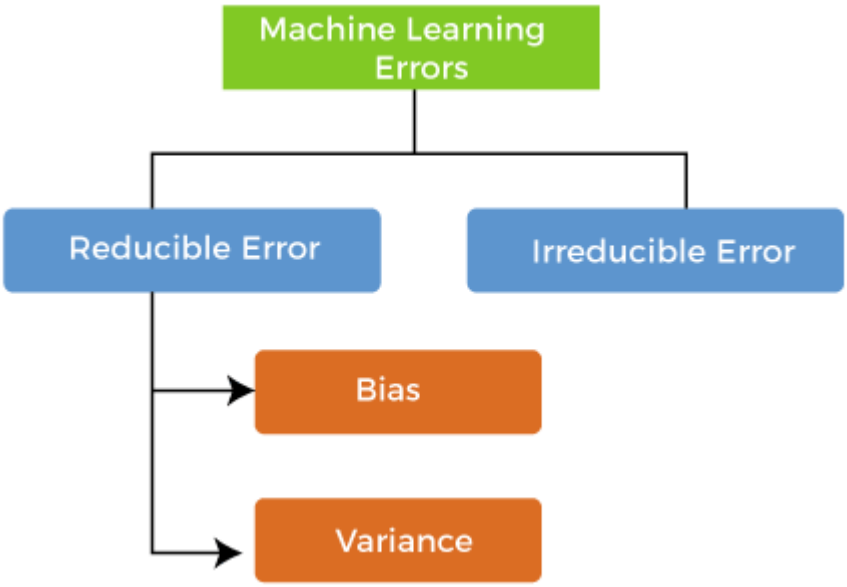
```
Squared error is  4.528372434192783
Mean squared error is  0.022641862170963915
Root mean square error is  0.15047213087799322
R2 score is  0.8001010698130744
```

Bias-Variance

Errors in Machine Learning?

In machine learning, an error is a measure of how accurately an algorithm can make predictions for the previously unknown dataset. On the basis of these errors, the machine learning model is selected that can perform best on the particular dataset. There are mainly two types of errors in machine learning, which are:

- **Reducible errors:** These errors can be reduced to improve the model accuracy. Such errors can further be classified into bias and Variance.



- **Irreducible errors:** These errors will always be present in the model regardless of which algorithm has been used. The cause of these errors is unknown variables whose value can't be reduced.

What is Bias?

While making predictions, a difference occurs between prediction values made by the model and actual values/expected values, and this difference is known as bias errors or Errors due to bias.

Bias represents the distance of our predicted values with the original values.

- **Low Bias:** A low bias model will make fewer assumptions about the form of the target function.
- **High Bias:** A model with a high bias makes more assumptions, and the model becomes unable to capture the important features of our dataset. **A high bias model also cannot perform well on new data.**
- Generally, a linear algorithm has a high bias, as it makes them learn fast. The simpler the algorithm, the higher the bias it has likely to be introduced. Whereas a nonlinear algorithm often has low bias.
- Some examples of machine learning algorithms with low bias **are Decision Trees, k-Nearest Neighbours and Support Vector Machines**. At the same time, an algorithm with high bias is **Linear Regression, Linear Discriminant Analysis and Logistic Regression.**

Ways to reduce High Bias:

High bias mainly occurs due to a much simple model. Below are some ways to reduce the high bias:

- Increase the input features as the model is underfitted.
- Decrease the regularization term.
- Use more complex models, such as including some polynomial features.

What is a Variance Error?

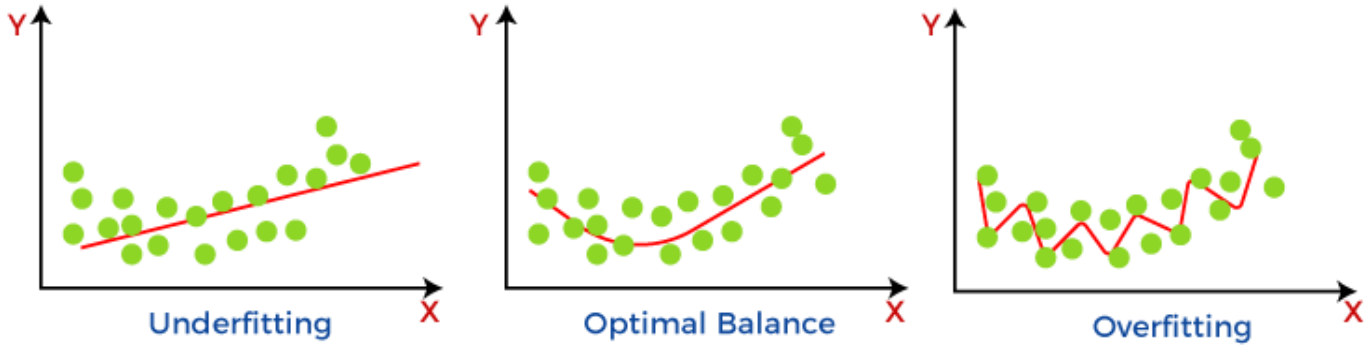
variance tells that how much a random variable is different from its expected value.

Variance represents the distance of the predicted values, which means how much the predicted values are scattered from each other.

A model with high variance has the below problems:

- A high variance model leads to overfitting.
- Increase model complexities.

Usually, nonlinear algorithms have a lot of flexibility to fit the model, have high variance.



Some examples of machine learning algorithms with low variance are, **Linear Regression, Logistic Regression, and Linear discriminant analysis**. At the same time, algorithms with high variance are **decision tree, Support Vector Machine, and K-nearest neighbours.**

Ways to Reduce High Variance:

- Reduce the input features or number of parameters as a model is overfitted.
- Do not use a much complex model.
- Increase the training data.
- Increase the Regularization term.

Different Combinations of Bias-Variance

There are four possible combinations of bias and variances, which are represented by the below diagram:

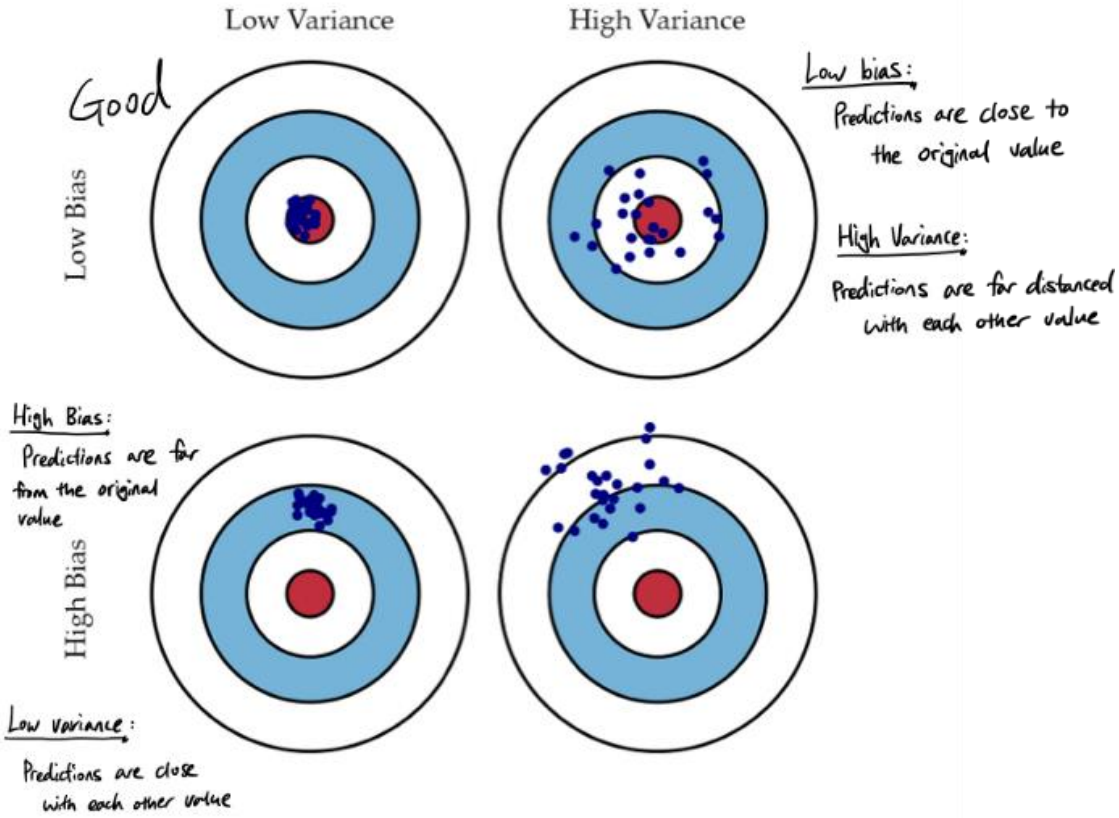
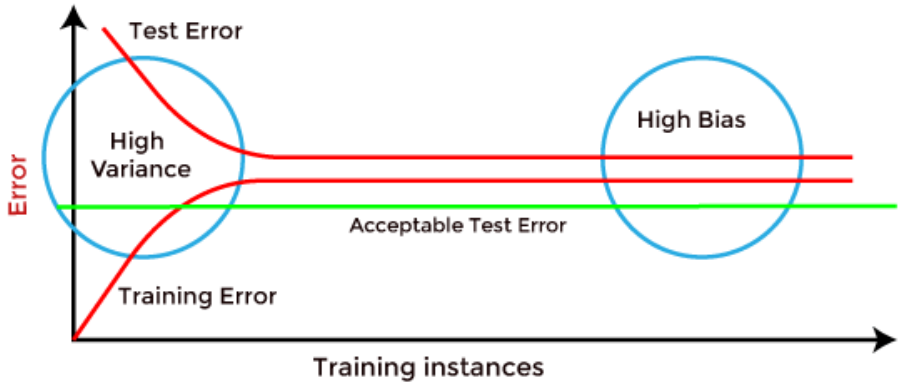


Fig. 1 Graphical illustration of bias and variance.

1. **Low-Bias, Low-Variance:** The combination of low bias and low variance shows an ideal machine learning model. However, it is not possible practically.
2. **Low-Bias, High-Variance:** With low bias and high variance, model predictions are inconsistent and accurate on average. This case occurs when the model learns with a large number of parameters and hence leads to an **overfitting**
3. **High-Bias, Low-Variance:** With High bias and low variance, predictions are consistent but inaccurate on average. This case occurs when a model does not learn well with the training dataset or uses few numbers of the parameter. It leads to **underfitting** problems in the model.
4. **High-Bias,High-Variance:** With high bias and high variance, predictions are inconsistent and also inaccurate on average.

How to identify High variance or High Bias?

High variance can be identified if the model has:



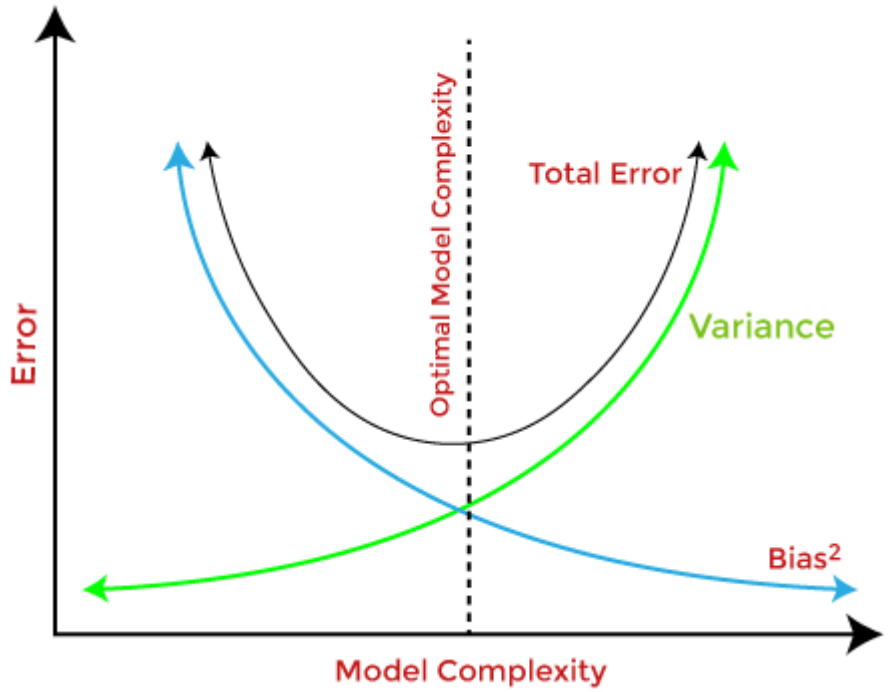
- Low training error and high test error.

High Bias can be identified if the model has:

- High training error and the test error is almost similar to training error.

Bias-Variance Trade-Off

While building the machine learning model, it is really important to take care of bias and variance in order to avoid overfitting and underfitting in the model. If the model is very simple with fewer parameters, it may have low variance and high bias. Whereas, if the model has a large number of parameters, it will have high variance and low bias. So, it is required to make a balance between bias and variance errors, and this balance between the bias error and variance error is known as **the Bias-Variance trade-off.**



For an accurate prediction of the model, algorithms need a low variance and low bias. But this is not possible because bias and variance are related to each other:

- If we decrease the variance, it will increase the bias.
- If we decrease the bias, it will increase the variance.

Bias-Variance trade-off is a central issue in supervised learning. Ideally, we need a model that accurately captures the regularities in training data and simultaneously generalizes well with the unseen dataset. Unfortunately, doing this is not possible simultaneously. Because a high variance algorithm may perform well with training data, but it may lead to overfitting to noisy data. Whereas, high bias algorithm generates a much simple model that may not even capture important regularities in the data. So, we need to find a sweet spot between bias and variance to make an optimal model.

Hence, the ***Bias-Variance trade-off is about finding the sweet spot to make a balance between bias and variance errors.***

Cost Function

Machine Learning models require a high level of accuracy to work in the actual world. But how do you calculate how wrong or right your model is? A machine learning parameter that is used for correctly judging the model, cost functions are important to understand to know how well the model has estimated the relationship between your input and output parameters.

What is Cost Function ?

After training your model, you need to see how well your model is performing. While accuracy functions tell you how well the model is performing, they do not provide you with an insight on how to better improve them. Hence, you need a correctional function that can help you compute when the model is the most accurate, as you need to hit that small spot between an undertrained model and an overtrained model.

A Cost Function is used to measure just how wrong the model is in finding a relation between the input and output. It tells you how badly your model is behaving/predicting

What Is Gradient Descent?

Gradient Descent is an algorithm that is used to optimize the cost function or the error of the model. It is used to find the minimum value of error possible in your model.

Gradient Descent can be thought of as the direction you have to take to reach the least possible error. The error in your model can be different at different points, and you have to find the quickest way to minimize it, to prevent resource wastage.

Gradient Descent can be visualized as a ball rolling down a hill. Here, the ball will roll to the lowest point on the hill. It can take this point as the point where the error is least as for any model, the error will be minimum at one point and will increase again after that.

In gradient descent, you find the error in your model for different values of input variables. This is repeated, and soon you see that the error values keep getting smaller and smaller. Soon you'll arrive at the values for variables when the error is the least, and the cost function is optimized.

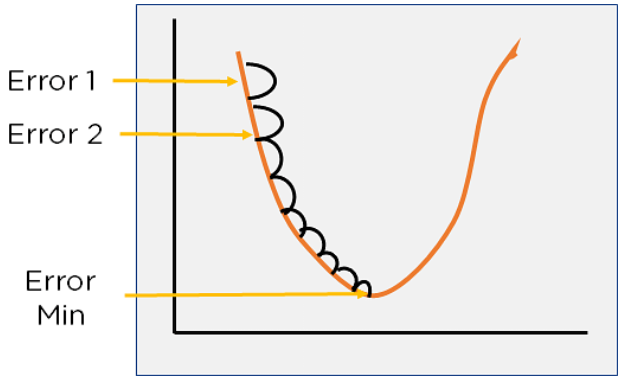


Figure 2: Gradient Descent

What Is the Cost Function For Linear Regression?

A Linear Regression model uses a straight line to fit the model. This is done using the equation for a straight line as shown :

$$\text{Output} = a*\text{Input} + b$$

Figure 3: Linear regression function

In the equation, you can see that two entities can have changeable values (variable) a, which is the point at which the line intercepts the x-axis, and b, which is how steep the line will be, or slope.

At first, if the variables are not properly optimized, you get a line that might not properly fit the model. As you optimize the values of the model, for some variables, you will get the perfect fit. The perfect fit will be a straight line running through most of the data points while ignoring the noise and outliers. A properly fit Linear Regression model looks as shown below :

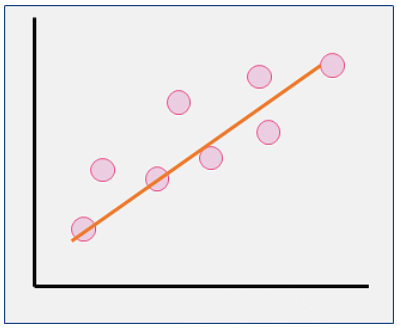


Figure 4: Linear regression graph

For the Linear regression model, the **cost function** will be the minimum of the **Root Mean Squared Error** of the model, obtained by subtracting the predicted values from actual values. The cost function will be the minimum of these error values.

$$\text{Cost Function (J)} = \frac{1}{n}\sum_{i=0}^{n}(h_\theta(x^i) - y^i)^2$$

Figure 5: Linear regression cost function

By the definition of gradient descent, you have to find the direction in which the error decreases constantly. This can be done by finding the difference between errors.

The small difference between errors can be obtained by differentiating the cost function and subtracting it from the previous gradient descent to move down the slope.

$$\text{Gradient Descent } \theta_j = \theta_j - \alpha \frac{\partial J}{\partial \theta}$$

After substituting the value of the cost function (J) in the above equation, you get :

Figure 6: Linear regression gradient descent function simplified

In the above equations, a is known as the learning rate. It decides how fast you move down the slope. If alpha is large, you take big steps, and if it is small; you take small steps. If alpha is too large, you can entirely miss the least error point and our results will not be accurate. If it is too small it will take too long to optimize the model and you will also waste computational power. Hence you need to choose an optimal value of alpha.

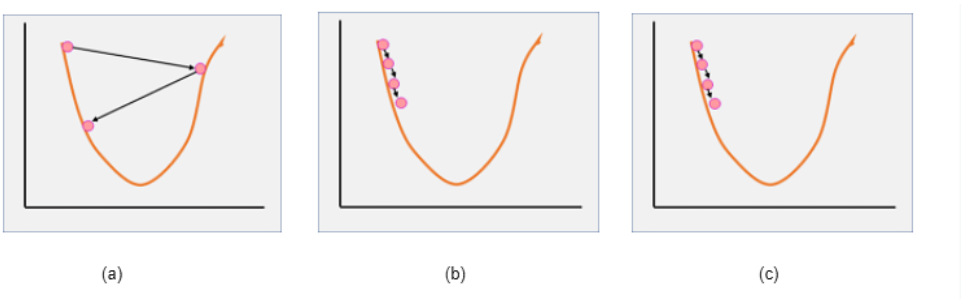


Figure 8: (a) Large learning rate, (b) Small learning rate, (c) Optimum learning rate

What is Cost Function for Neural Networks?

A neural network is a machine learning algorithm that takes in multiple inputs, runs them through an algorithm, and essentially sums the output of the different algorithms to get the final output.

The cost function of a neural network will be the sum of errors in each layer. This is done by finding the error at each layer first and then summing the individual error to get the total error. In the end, it can represent a neural network with cost function optimization as :

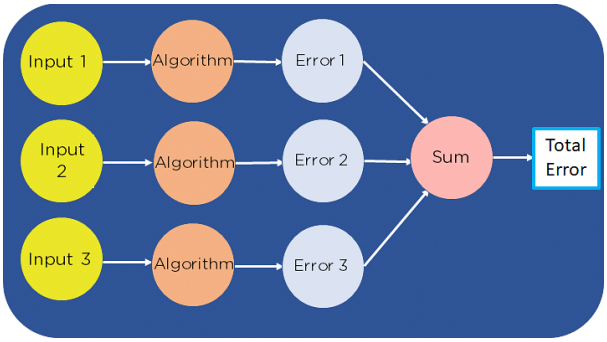


Figure 9: Neural network with the error function

For neural networks, each layer will have a cost function, and each cost function will have its own least minimum error value. Depending on where you start, you can arrive at a unique value for the minimum error. You need to find the minimum value out of all local minima. This value is called the global minima.

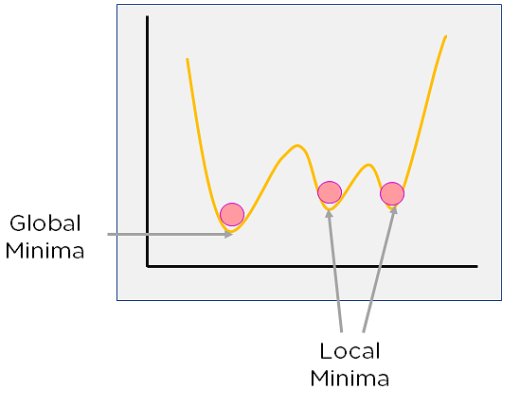


Figure 10: Cost function graph for Neural Networks

The cost function for neural networks is given as :

$$\text{Cost Function (J)} = \frac{1}{n}\sum_{i=0}^{n}(y^i - (mx^i + b))^2$$

Figure 11: Cost function for Neural Networks

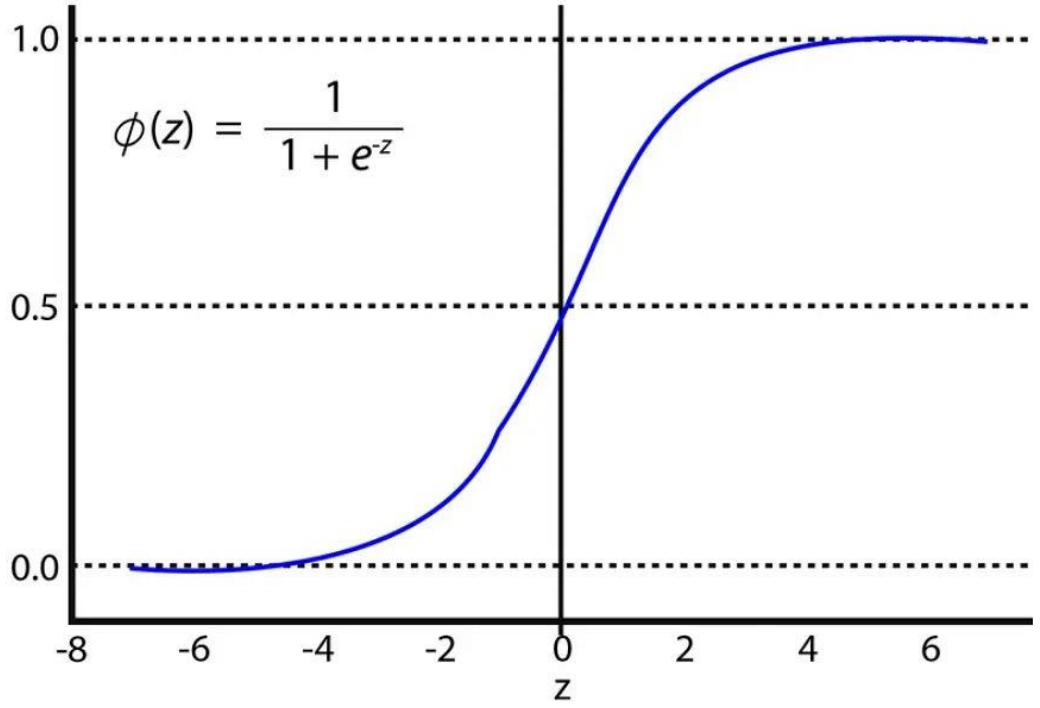
Gradient descent is just the differentiation of the cost function. It is given as :

$$\text{Gradient Descent } (\frac{\partial J}{\partial \theta}) = \begin{bmatrix} \frac{1}{N}\sum_{i=0}^{n}(-2\,x_i(y_i - (mx_i + b))) \\ \frac{1}{N}\sum_{i=0}^{n}(-2\,(y_i - (mx_i + b))) \end{bmatrix}$$

Figure 12: Gradient descent for Neural Networks

Logistic Regression

- Don't get confused by its name! It is a **classification**, not a regression algorithm.
- It is used to estimate discrete values (Binary values like 0/1, yes/no, true/false) based on given set of independent variable(s).
- In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. Hence, It also known as logit regression.
- Since, it predicts the probability, its output values lies between 0 to 1 (as expected).



$$\Pr(Y = 1|X) = F(\beta_0 + \beta_1 X)$$

F is the cumulative standard ***logistic*** distribution function:

where $F(\beta_0 + \beta_1 X) = \frac{1}{1+e^{-(\beta_0+\beta_1 X)}}$

Example: $\beta_0 = -3$, $\beta_1 = 2$, $X = .4$,
so $\beta_0 + \beta_1 X = -3 + 2*.4 = -2.2$ so
$\Pr(Y = 1|X=.4) = 1/(1+e^{-(-2.2)}) = .0998$

Why bother with logit if we have probit?

- Historically, logit is more convenient computationally
- In practice, logit and probit are very similar

Test for Trend - Logistic Regression Alternative

Logistic Model:

$$\Pr(Y=1|X_1,X_2,\ldots,X_p) = \frac{1}{1+e^{-(b_0+b_1X_1+b_2X_2+\ldots+b_pX_p)}}$$

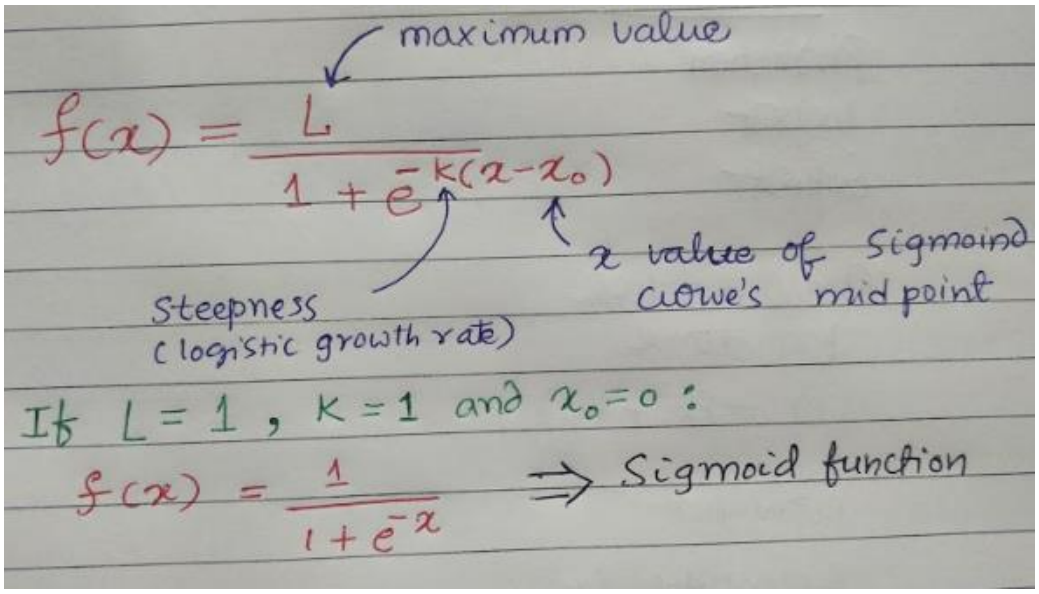
$$\text{logit } P = \ln\frac{P}{1-P} = b_0+b_1X_1+b_2X_2+\ldots+b_pX_p$$

where $P = \Pr(Y=1|X_1,X_2,\ldots,X_p)$

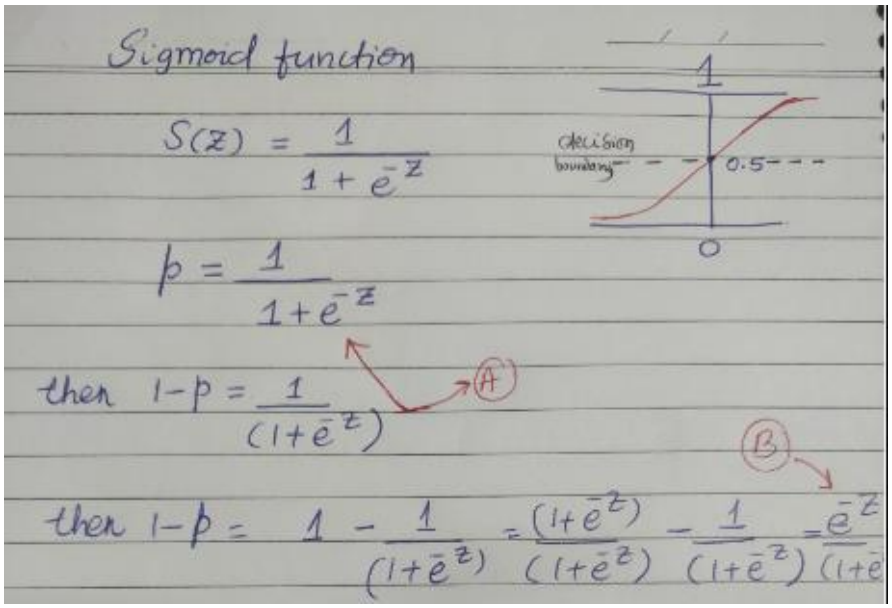
X = independent variable or predictor

Y = dichotomous dependent or outcome variable

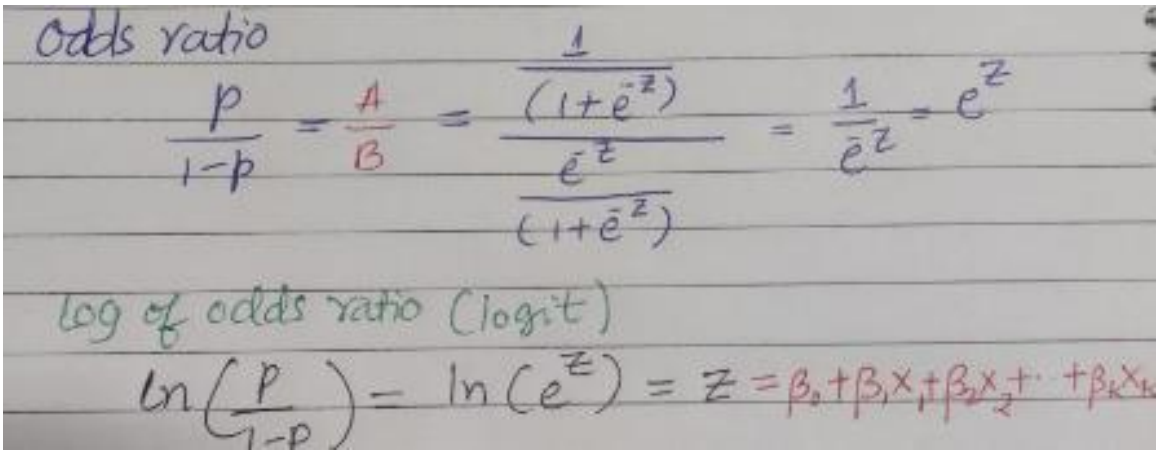
Sigmoid function is a special case of Logistic function and lies between 0 – 1. (Squashing)



Use sigmoid function to model the probability of dependent variable being 1 or 0 (binary classification).



Now we will see how to derive the log of odds ratio [p/(1-p)]



The logistic regression assigns each row a probability of bring True and then makes a prediction for each row where that prbability is >= 0.5 i.e**. 0.5 is the default threshold**.

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

In [2]:
```
df = pd.read_csv('d:Social_Network_Ads.csv')
df
```

Out[2]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

In [3]:
```
x = df.iloc[:,2:-1]
y = df.iloc[:,-1]
```

In [4]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=4)
```

In [5]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
```

In [6]:
```
lr.fit(x_train,y_train)
```

Out[6]:
```
LogisticRegression()
```

In [7]:
```
ypre = lr.predict(x_test)
```

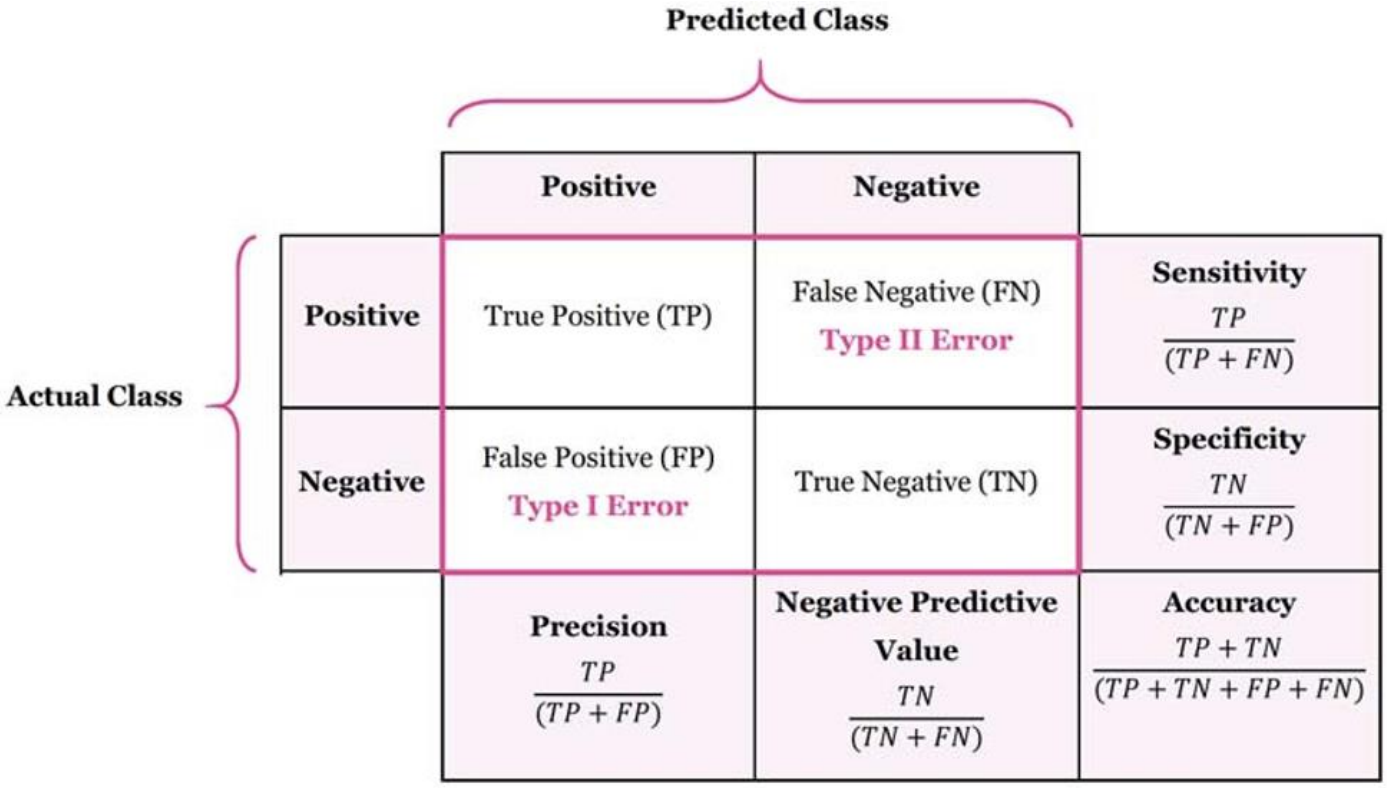
In [8]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,ypre)
```

Out[8]:
```
0.7125
```

Classification Metrics

Accuracy, Confusion Matrix, Precision, Recall, F1 score, AUC & ROC

Confusion matrices are the result of classification problems. There are four possible values that make up the result: True Positive, False Negative, False Positive, and True Negative.



- **True Positive** — these are cases in which the classification model created correctly predicted positive.
- **False Negative** — these are cases in which the classification model predicted false, but are actually positive. These are also considered Type II errors.
- **False Positive** — these are the cases in which the classification model predicted positive, but are actually negative. These are also considered Type I errors.
- **True Negative** — these are the cases in which the classification model correctly predicted negative.

- **Sensitivity (Recall)** — also known as the True Positive Rate or Recall. Outcomes that are correctly predicted as positive.
- **Specificity** — also known as the True Negative Rate. Outcomes that are correctly predicted as negative.
- **Accuracy** — Outcomes that are correctly labeled at true. it's not compulsory that a good result. Many times data imbalanced problem. Than we use Precision and Recall.
- **Negative Predictive Value** — Outcomes that are correctly labeled as false.
- **Precision** — Outcomes that are correctly predicted positive.

When use Precision and Recall?

- If the negative samples are important, we should focus on precision. Otherwise, we should focus on recall.
- However, as to F1-score, the value higher, the model is better.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\text{-}score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

AUC ROC Curve

Setting different thresholds for classifying positive class for data points will inadvertently change the Sensitivity and Specificity of the model. And one of these thresholds will probably give a better result than the others, depending on whether we are aiming to lower the number of False Negatives or False Positives.

ID	Actual	Prediction Probability	>0.6	>0.7	> 0.8	Metric
1	0	0.98	1	1	1	
2	1	0.67	1	0	0	
3	1	0.58	0	0	0	
4	0	0.78	1	1	0	
5	1	0.85	1	1	1	
6	0	0.86	1	1	1	
7	0	0.79	1	1	0	
8	0	0.89	1	1	1	
9	1	0.82	1	1	1	
10	0	0.86	1	1	1	
			0.75	0.5	0.5	TPR
			1	1	0.66	FPR
			0	0	0.33	TNR
			0.25	0.5	0.5	FNR

The metrics change with the changing threshold values. We can generate different confusion matrices and compare the various metrics that we discussed in the previous section. But that would not be a prudent thing to do. Instead, what we can do is generate a plot between some of these metrics so that we can easily visualize which threshold is giving us a better result.

The Receiver Operator Characteristic (ROC) curve is an evaluation metric for binary classification problems. It is a probability curve that plots the TPR against FPR at various threshold values and essentially separates the ‘signal’ from the ‘noise’. The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

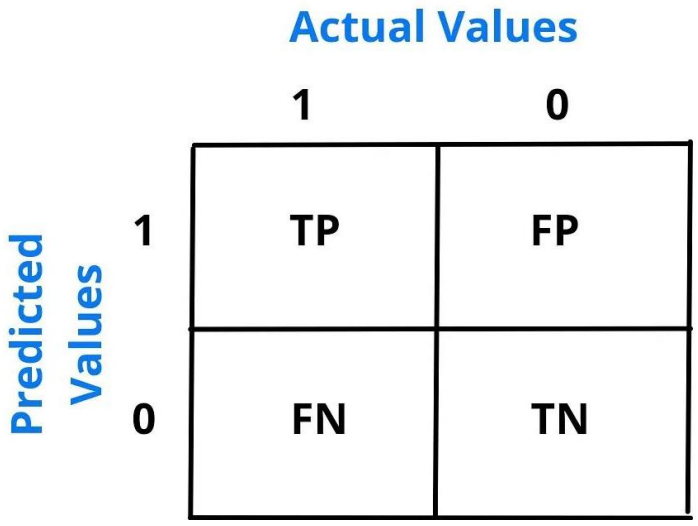
- The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.
- When AUC = 1, then the classifier is able to perfectly distinguish between all the Positive and the Negative class points correctly. If, however, the AUC had been 0, then the classifier would be predicting all Negatives as Positives, and all Positives as Negatives.
- When 0.5<AUC<1, there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values. This is so because the classifier is able to detect more numbers of True positives and True negatives than False negatives and False positives.
- When AUC=0.5, then the classifier is not able to distinguish between Positive and Negative class points. Meaning either the classifier is predicting random class or constant class for all the data points.

So, the higher the AUC value for a classifier, the better its ability to distinguish between positive and negative classes.

OR

Evaluation Metrics For Classification

Confusion Matrix:-



What is a confusion Metrix

- Confusion Matrix is the visual representation of the Actual v/s Predicted values.
- It measures the performance of our Machine Learning classification model and looks like a table-like structure.
- This is how a Confusion Matrix of a binary classification problem looks like:

Elements of Confusion Matrix

It represents the different combinations of Actual vs Predicted values. Let's define them one by one.

- **TP (True Positive):** The values which where actually positive and were predicted positive.
- **FP (False Positive):** The values which were actually negative but falsely predicted as positive. Also Known as Type II Error.
- **FN (False Negative):** The values which were actually positive but falsely predicted as negative. Also known as Type II Error.
- **TN (True Negative):** The values which were actually negative and were predicted negative.

Precision And Recall

Precision : Out of all positive predictions, how many are actually positive.

$$Precision = \frac{Predictions\ Actually\ Positive}{Total\ Predicted\ Positive}$$

$$Precision = \frac{True\ Positives}{True\ Positive + False\ Positive}$$

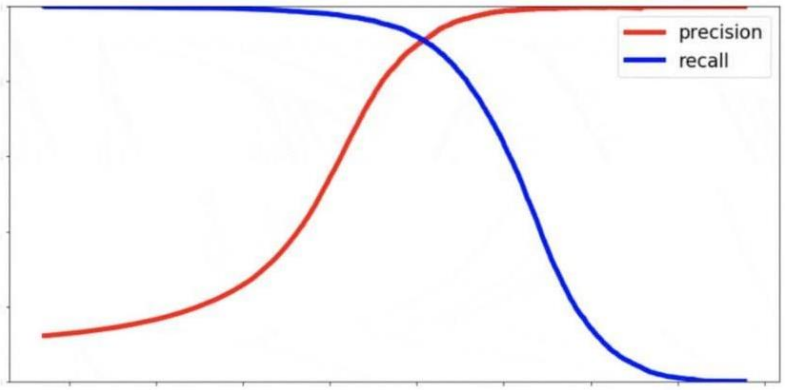
Precision is used when avoiding False Positive is more important than encountering False Negative.

Recall : Out of all actual positive, how many have been predicted as positive.

$$Recall = \frac{Predictions\ Actually\ Positive}{Total\ Actual\ Positive}$$

$$Recall = \frac{True\ Positives}{True\ Positive + False\ Negative}$$

Precision is used when avoideing False Negative is more important than encountering False Positive



- High Precision, Low Recall
- High Recall, Low Precision
- Precision and Recall are used in diverse case and prioritizing one over the other depends upon the use case.

High Precision, Low Recall

If we aim for very high Precision i.e. trying to strictly predict the True Positive and avoiding False Positive, there is a good chance that the False Negative will go higher. Therefore, the recall will drop to a lower value.

High Recall, Low Precision

Similarly, If we aim for very high Recall i.e. trying to strictly predict all the actual positive as True positive and avoiding False negative, there is a good chance that the False positive will go higher. Therefore, the precision will drop to a lower value.

F1 Score

F1-score is a **HARMONIC MEAN** of Precision and Recall. And so it gives a combined idea about these two metrics. It is maximus when Precision is equal to Recall.

When we try toincrease the precision of our model, the recall goes down, and vice-versa. The F1-score captures both the trends in a single value.

F1-score is a harmonic mean of Precision and Recall, But why are taking a harmonic mean and not an arithmetic mean?

$$\text{F1 Score} = \frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$$

- This is because harmonic mean punishes extreme values more.
- Let us understand this with an example. We have a binary classification model with the following results:

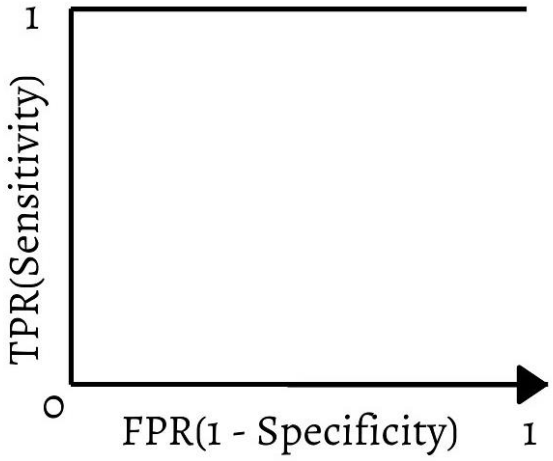
  Precision: 0, Recall: 1

  Here, if we take the arithmetic mean, we get 0.5. It is clear that the above result comes from a dumb classifier which just ignores the input and just predicts one of the classes as outputs.
- Now, if we were to take HM, we will gat 0 which is accurate as this model is useless for all purposes.
- This seems simple. There are situations however for which a data scientist would like to give a percentage more importance/weight to either precision or recall.

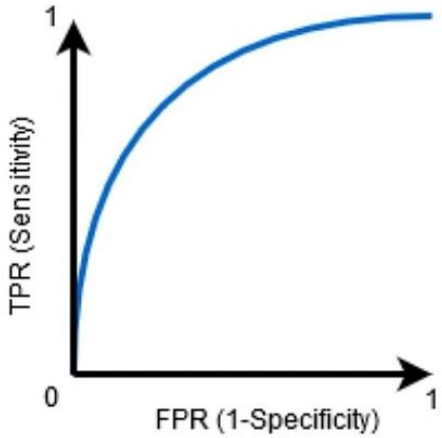
AUC ROC Curve

The Receiver Operator Characteristics (ROC) curve is an evaluation metric for **BINARY CLASSIFICATION** problems. It is a probability curve that plots the TPR against FPR at various threshold values and essentially separates the 'signal' from the 'noise'. The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

Note: The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.



When AUC = 1, then the classifer is able to perfectly distinguish between all the Positive and the Negative class points correctly. If, hower, the AUC had been 0, then the classifier would be predicting all Negatives as Positives, and all Positives as Negatives.



When 0.5<AUC<1, there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values. This is so because the classifier is able to detect more numbers of True positives and True negatives than False negatives and False positives.

Log Loss :- AUR ROC considers the predicted probabilities for determining our model's performance. However, there is an Issue with AUC – ROC, it only takes into account the order of probabilities and hence it does not take into account the model's capability to predict higher probability for samples more likely to be positive.

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

In [3]:
```
df = pd.read_csv('hotel_booking.csv')
```

In [5]:
```
df.sample()
```

Out[5]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_nun
11420	Resort Hotel	1	86	2017	May	

1 rows × 36 columns

In [6]:
```
df.shape
```

Out[6]:
```
(119390, 36)
```

In [7]:
```
df.columns
```

Out[7]:
```
Index(['hotel', 'is_canceled', 'lead_time', 'arrival_date_year',
       'arrival_date_month', 'arrival_date_week_number',
       'arrival_date_day_of_month', 'stays_in_weekend_nights',
       'stays_in_week_nights', 'adults', 'children', 'babies', 'meal',
       'country', 'market_segment', 'distribution_channel',
       'is_repeated_guest', 'previous_cancellations',
       'previous_bookings_not_canceled', 'reserved_room_type',
       'assigned_room_type', 'booking_changes', 'deposit_type', 'agent',
       'company', 'days_in_waiting_list', 'customer_type', 'adr',
       'required_car_parking_spaces', 'total_of_special_requests',
       'reservation_status', 'reservation_status_date', 'name', 'email',
       'phone-number', 'credit_card'],
      dtype='object')
```

In [8]:
```
# Deleting unuseful columns
df = df.drop(['days_in_waiting_list', 'arrival_date_year', 'assigned_room_type', 'b
              'reservation_status', 'country', 'days_in_waiting_list','name', 'ema
```

In [9]:
```
df.shape
```

Out[9]:
```
(119390, 26)
```

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
hotel                               0
is_canceled                         0
lead_time                           0
arrival_date_month                  0
arrival_date_week_number            0
arrival_date_day_of_month           0
stays_in_weekend_nights             0
stays_in_week_nights                0
adults                              0
children                            4
babies                              0
meal                                0
market_segment                      0
distribution_channel                0
is_repeated_guest                   0
previous_cancellations              0
previous_bookings_not_canceled      0
reserved_room_type                  0
deposit_type                        0
agent                           16340
company                        112593
customer_type                       0
adr                                 0
required_car_parking_spaces         0
total_of_special_requests           0
reservation_status_date             0
dtype: int64
```

In [11]:
```
df = df.bfill().ffill()
```

In [12]:
```
df.dtypes
```

Out[12]:
```
hotel                           object
is_canceled                      int64
lead_time                        int64
arrival_date_month              object
arrival_date_week_number         int64
arrival_date_day_of_month        int64
stays_in_weekend_nights          int64
stays_in_week_nights             int64
adults                           int64
children                       float64
babies                           int64
meal                            object
market_segment                  object
distribution_channel            object
is_repeated_guest                int64
previous_cancellations           int64
previous_bookings_not_canceled   int64
reserved_room_type              object
deposit_type                    object
agent                          float64
company                        float64
customer_type                   object
adr                            float64
required_car_parking_spaces      int64
total_of_special_requests        int64
reservation_status_date         object
dtype: object
```

In [13]:
```
l = []
```

```
for i in df.columns:
    if df[i].dtypes =='O':
        l.append(i)
```

In [14]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for i in l:
    df[i] = le.fit_transform(df[i])
```

In [15]:
```
# df.dtypes
```

In [16]:
```
x = df.drop('is_canceled',axis=1)
y = df['is_canceled']
```

In [17]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.20)
```

In [18]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
```

In [19]:
```
dt.fit(x_train,y_train)
```

Out[19]:
```
DecisionTreeClassifier()
```

In [18]:
```
dt_pre = dt.predict(x_test)
```

In [19]:
```
data = pd.DataFrame({'Actual Value':y_test,'Predicted Value':dt_pre})
```

In [20]:
```
data
```

Out[20]:

	Actual Value	Predicted Value
247	1	1
81798	1	1
114200	0	0
110394	0	0
33450	0	0
...	...	...
60337	1	1
13293	1	1
50184	1	1
43940	1	1
94686	0	0

23878 rows × 2 columns

In [21]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, dt_pre)
```

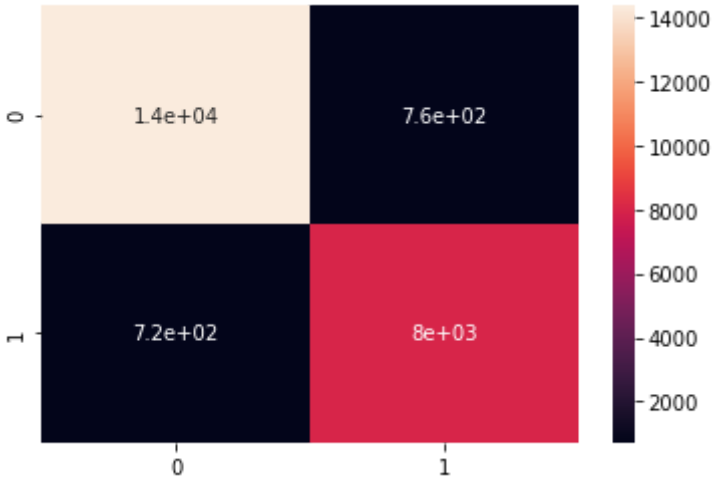
Out[21]:
```
0.9377669821593099
```

In [22]:
```
from sklearn.metrics import classification_report, confusion_matrix
cf_matrix = confusion_matrix(y_test,dt_pre)
print(cf_matrix)
print(classification_report(y_test,dt_pre))
```

```
[[14351   764]
 [  722  8041]]
              precision    recall  f1-score   support

           0       0.95      0.95      0.95     15115
           1       0.91      0.92      0.92      8763

    accuracy                           0.94     23878
   macro avg       0.93      0.93      0.93     23878
weighted avg       0.94      0.94      0.94     23878
```

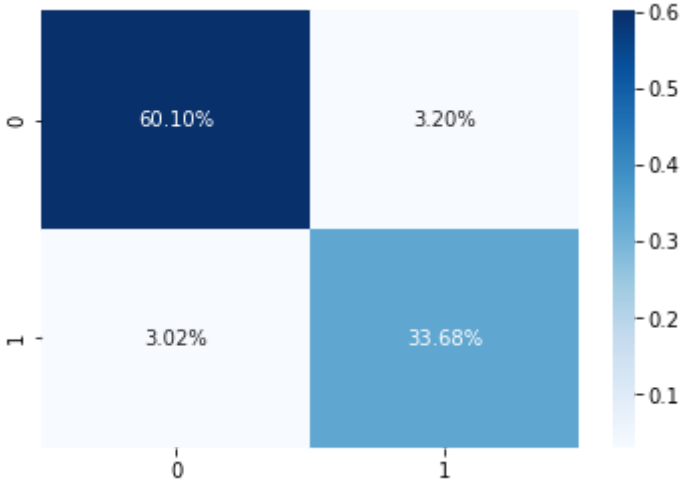
In [23]:
```
import seaborn as sns
sns.heatmap(cf_matrix, annot=True)
```

Out[23]: <AxesSubplot:>



In [24]:
```
sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True,
            fmt='.2%', cmap='Blues')
```

Out[24]: <AxesSubplot:>



In [25]:
```
# non-binary classifier (3x3 in this case)

# make_confusion_matrix(cf_matrix_3x3, figsize=(8,6), cbar=False)
```

In [26]:
```
confusion = confusion_matrix(y_test, dt_pre)
confusion.ravel()
```

Out[26]:
```
array([14351,   764,   722,  8041], dtype=int64)
```

In [27]:
```
from sklearn.metrics import precision_score
precision_positive = precision_score(y_test, dt_pre, pos_label=1)
precision_negative = precision_score(y_test, dt_pre, pos_label=0)

precision_positive, precision_negative
```

Out[27]:
```
(0.9132311186825667, 0.9520997810654813)
```

In [28]:
```
from sklearn.metrics import recall_score
recall_sensitivity = recall_score(y_test, dt_pre, pos_label=1)
recall_specificity = recall_score(y_test, dt_pre, pos_label=0)

recall_sensitivity, recall_specificity
```

Out[28]:
```
(0.9176081250713226, 0.9494541845848495)
```

In [29]:
```
from sklearn.metrics import f1_score
f1_positive = f1_score(y_test, dt_pre, pos_label=1)
f1_negative = f1_score(y_test, dt_pre, pos_label=0)

f1_positive, f1_negative
```

Out[29]:
```
(0.9154143897996357, 0.9507751424407049)
```

In [30]:
```
from sklearn.metrics import log_loss
log_loss(y_test,dt_pre)
```

Out[30]:
```
2.149477871483121
```

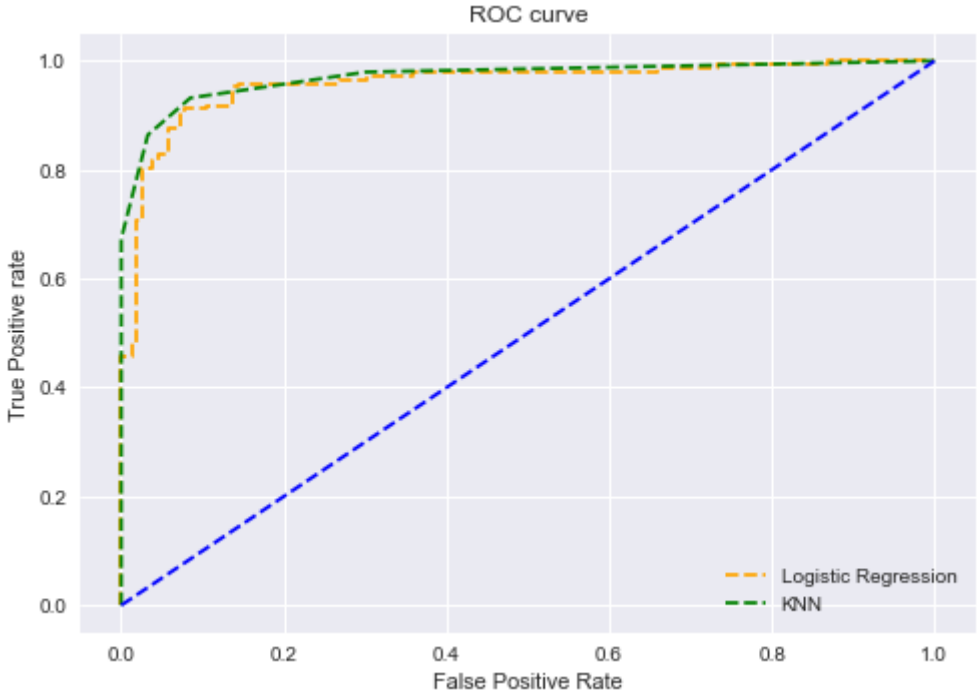
In [31]:
```
# AUC ROC Curve

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# generate two class dataset
x, y = make_classification(n_samples=1000,n_classes=2,n_features=20,random_state=24

# split into train-test sets
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3,random_state=

# train models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

# logistic regression
model1 = LogisticRegression()
# knn
model2 = KNeighborsClassifier(n_neighbors=4)

# fit model
model1.fit(x_train, y_train)
model2.fit(x_train, y_train)

# predict probabilities
```

```
pred_prob1 = model1.predict_proba(x_test)
pred_prob2 = model2.predict_proba(x_test)

from sklearn.metrics import roc_curve

# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(y_test, pred_prob1[:, 1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(y_test, pred_prob2[:, 1], pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)

from sklearn.metrics import roc_auc_score

# auc scores
auc_score1 = roc_auc_score(y_test, pred_prob1[:, 1])
auc_score2 = roc_auc_score(y_test, pred_prob2[:, 1])

print(auc_score1, auc_score2)

print(auc_score1, auc_score2)

print(auc_score1, auc_score2)
```

```
0.958249966653328 0.970432617491441
0.958249966653328 0.970432617491441
0.958249966653328 0.970432617491441
```

In [32]:

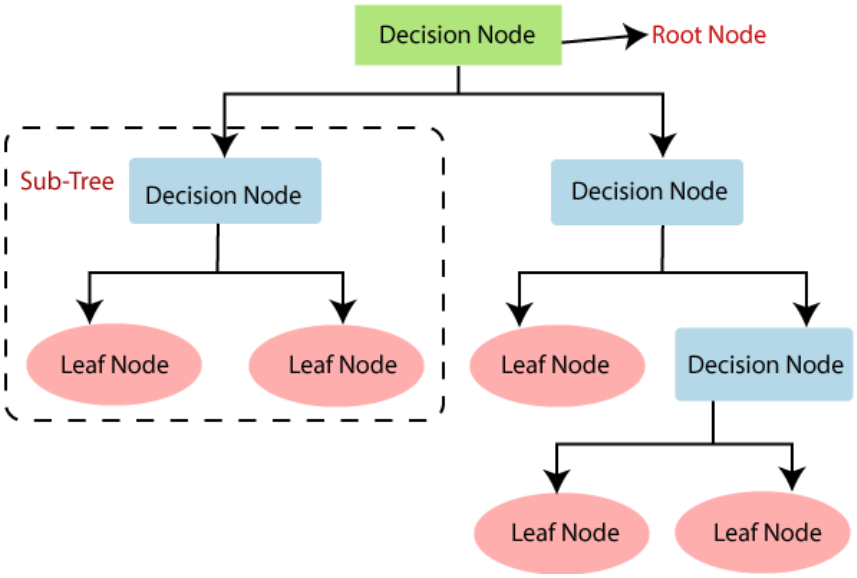
```
# matplotlib
import matplotlib.pyplot as plt
plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic Regression')
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='KNN')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```


ROC curve
True Positive rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Logistic Regression
KNN

Decission Tree

The major difference between a classification tree and a regression tree is the nature of the variable to be predicted. In a regression tree, the variable is continuous rather than categorical.

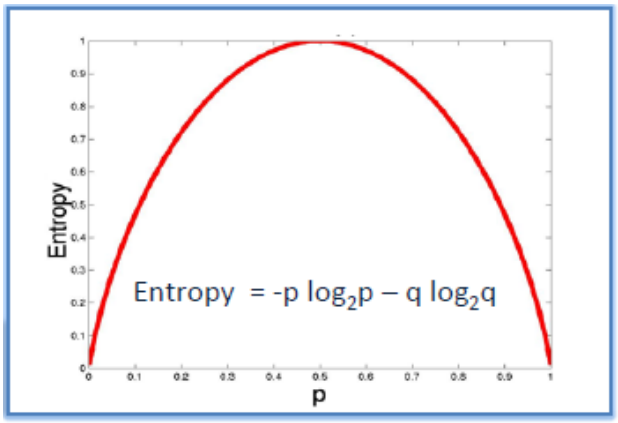
The decision tree is a type of supervised learning algorithm (having a predefined target variable) that is mostly used in classification problems.
It works for both categorical and continuous input and output variable. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator) in input variables.



- **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
- **Splitting:** It is a process of dividing a node into two or more sub-nodes.
- **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
- **Leaf/Terminal Node:** Nodes do not split is called Leaf or Terminal node.
- **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
- **Branch / Sub-Tree:** A sub section of entire tree is called branch or sub-tree.
- **Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes whereas sub-nodes are the child of parent node.

	outlook	temp	humidity	wind	decision
0	2	1	0	1	0
1	2	1	0	0	0
2	0	1	0	1	1
3	1	2	0	1	1
4	1	0	1	1	1
5	1	0	1	0	0
6	0	0	1	0	1
7	2	2	0	1	0
8	2	0	1	1	1
9	1	2	1	1	1
10	2	2	1	0	1
11	0	2	0	0	1
12	0	1	1	1	1
13	1	2	0	0	0

Entropy formula



Entropy = -0.5 $\log_2$0.5 – 0.5 $\log_2$0.5 = 1

a) Entropy using the frequency table of one attribute:

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)
= Entropy (0.36, 0.64)
= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)
= 0.94

b) Entropy using the frequency table of two attributes:

$$E(T, X) = \sum_{c \in X} P(c) E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

E(PlayGolf, Outlook) = **P**(Sunny)***E**(3,2) + **P**(Overcast)***E**(4,0) + **P**(Rainy)***E**(2,3)
= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971
= 0.693

Step 1: Calculate entropy of the target.

Entropy(PlayGolf) = Entropy (5,9)

= Entropy (0.36, 0.64)

= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)

= 0.94

Step 2: The dataset is then split on the different attributes. The entropy for each branch is calculated. The resulting entropy is subtracted from the entropy before the split. The result is the Information Gain.

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
Gain = 0.029			

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
Gain = 0.152			

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			

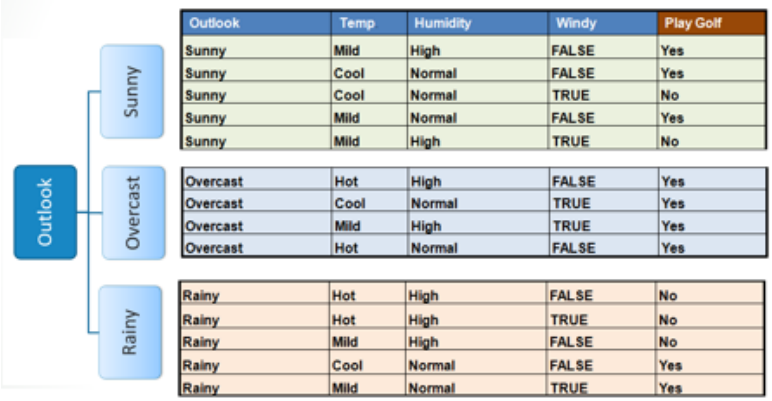
$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

G(PlayGolf, Outlook) = **E**(PlayGolf) – **E**(PlayGolf, Outlook)

= 0.940 – 0.693 = 0.247

Step 3: Choose attribute with the largest information gain as the decision node, divide the dataset by its branches and repeat the same process on every branch.

★		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			



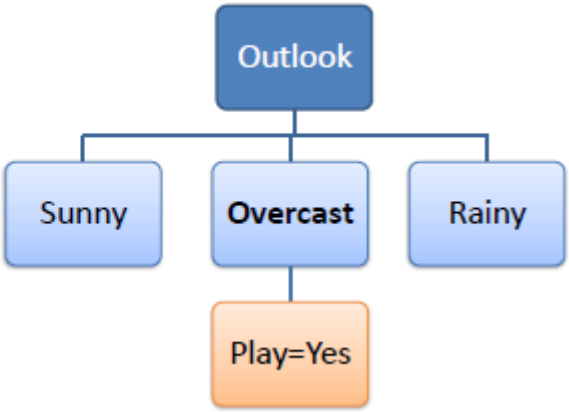
Outlook	Temp	Humidity	Windy	Play Golf
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Sunny	Mild	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

Overcast	Hot	High	FALSE	Yes
Overcast	Cool	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes

Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes

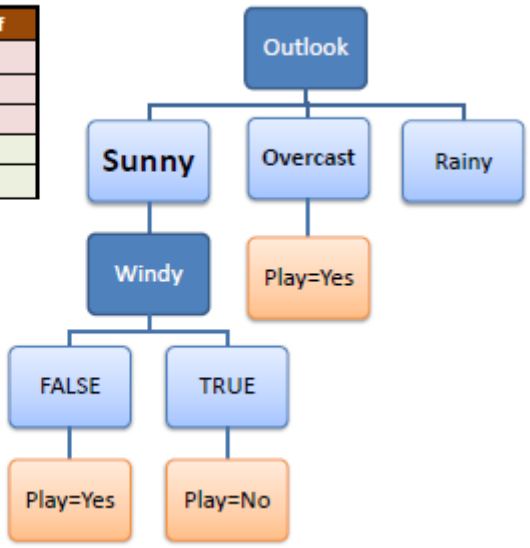
Step 4a: A branch with entropy of 0 is a leaf node.

Temp.	Humidity	Windy	Play Golf
Hot	High	FALSE	Yes
Cool	Normal	TRUE	Yes
Mild	High	TRUE	Yes
Hot	Normal	FALSE	Yes



Step 4b: A branch with entropy more than 0 needs further splitting.

Temp.	Humidity	Windy	Play Golf
Mild	High	FALSE	Yes
Cool	Normal	FALSE	Yes
Mild	Normal	FALSE	Yes
Cool	Normal	TRUE	No
Mild	High	TRUE	No



Step 5: The ID3 algorithm is run recursively on the non-leaf branches, until all data is classified.

Decision Trees - Issues

- Working with continuous attributes (binning)
- Avoiding overfitting
- Super Attributes (attributes with many unique values)

- Working with missing values

Impurity Criterion

Gini Index

$$I_G = 1 - \sum_{j=1}^{c} p_j^2$$

p_j: proportion of the samples that belongs to class c for a particular node

Entropy

$$I_H = -\sum_{j=1}^{c} p_j log_2(p_j)$$

p_j: proportion of the samples that belongs to class c for a particular node.

*This is the the definition of entropy for all non-empty classes ($p \neq 0$). The entropy is 0 if all samples at a node belong to the same class.

Gini Index

So as the first step we will find the root node of our decision tree. For that Calculate the Gini index of the class variable

- Gini(S) = 1 - $[(9/14)^2 + (5/14)^2] = 0.4591$
- First, consider case of Outlook

		play		
		yes	no	total
Outlook	sunny	3	2	5
	overcast	4	0	4
	rainy	2	3	5
				14

Gini(S, outlook) = (5/14)gini(3,2) + (4/14)*gini(4,0)+ (5/14)*gini(2,3) => $(5/14)(1 - (3/5)^2 - (2/5)^2) + (4/14)*0 + (5/14)(1 - (2/5)^2 - (3/5)^2)$=> 0.171+0+0.171 => 0.342

Find for all columns

- Choose one that has lower Gini gain. Gini gain is lower for outlook. So we can choose it as our root node.

Than repeat the same steps:-

Decision Tree Regressor

In [2]:
```
import pandas as pd
```

In [3]:
```
df = pd.read_csv('50_Startups.csv')
df.head()
```

Out[3]:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

In [4]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [5]:
```
df['State']=le.fit_transform(df['State'])
```

In [6]:
```
df.head(2)
```

Out[6]:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.2	136897.80	471784.10	2	192261.83
1	162597.7	151377.59	443898.53	0	191792.06

In [7]:
```
x=df.iloc[::-1]
y=df.iloc[:,-1]
```

In [8]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20)
```

In [9]:
```
from sklearn.tree import DecisionTreeRegressor
model=DecisionTreeRegressor()
```

In [10]:
```
model.fit(x_train,y_train)
```

Out[10]:
```
DecisionTreeRegressor()
```

In [11]:
```
model.predict(x_test)
```

Out[11]:
```
array([ 96712.8 ,  81005.76, 110352.25,  71498.49,  99937.59, 134307.35,
        71498.49,  71498.49, 103282.38,  81005.76])
```

Decision Tree Classifer

In [13]:
```
import pandas as pd
data = pd.read_csv('gini_index.csv')
data.head()
```

Out[13]:

	outlook	temp	humidity	wind	decision
0	sunny	hot	high	weak	no
1	sunny	hot	high	strong	no
2	overcast	hot	high	weak	yes
3	rain	mild	high	weak	yes
4	rain	cool	normal	weak	yes

In [14]:
```
data.shape
```

Out[14]:
```
(14, 5)
```

In [15]:
```
data.columns
```

Out[15]:
```
Index(['outlook', 'temp', 'humidity', 'wind', 'decision'], dtype='object')
```

In [16]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [15]:
```
categorical_columns = ['outlook', 'temp', 'humidity', 'wind', 'decision']
# I would recommend using columns names here if you're using pandas. If you're using numpy then stick with range(n) instead

for column in categorical_columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
# if numpy instead of pandas use X[:, column] instead
```

In [16]:
```
data
```

Out[16]:

	outlook	temp	humidity	wind	decision
0	2	1	0	1	0
1	2	1	0	0	0
2	0	1	0	1	1
3	1	2	0	1	1
4	1	0	1	1	1
5	1	0	1	0	0
6	0	0	1	0	1
7	2	2	0	1	0
8	2	0	1	1	1
9	1	2	1	1	1
10	2	2	1	0	1
11	0	2	0	0	1
12	0	1	1	1	1
13	1	2	0	0	0

In [17]:
```
x=data.iloc[:,:-1]
y=data.iloc[:,-1]
```

In [18]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20)
```

In [19]:
```
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
```

In [20]:
```
dtc.fit(x_train,y_train)
```

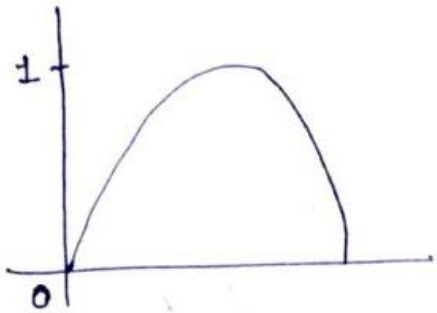
Out[20]:
```
DecisionTreeClassifier()
```

In [21]:
```
dtc.predict(x_test)
```

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

Decision Tree

Data set - gini_index.csv



Entropy :

Entropy lies b/w '0' to '1'.

$$Gain(S, A) = Entropy(S) - \Sigma\ Entropy(A)$$

$$Entropy(S) = -P_1(\log_2 P_1) - P_2(\log_2 P_2) - \dots - P_n(\log_2 P_n)$$

$$E(S) = -P(\checkmark) \times \log_2 P(\checkmark) - P(x) \log_2 P(x)$$

$$\Rightarrow S = 14\ [9+\ 5-]$$

$$\Rightarrow -\left(\frac{9}{14}\right)\log_2\left(\frac{9}{14}\right) - \left(\frac{5}{14}\right)\log_2\left(\frac{5}{14}\right)$$

$E(S) \Rightarrow 0.94029$ ← Entropy of all samples

- Now find the entropy of each sample.

$$Entropy(v) = -P(+)\log_2(P+) - P(-)\log_2 P(-)$$

Sunny = 5 = [2+ 3-]
Overcast = 4 = [4+ 0-]
Rainy = 5 = [3+ 2-]

$$E_{sunny} = -\left(\frac{2}{5}\right)\log\left(\frac{2}{5}\right) - \left(\frac{3}{5}\right)\log\left(\frac{3}{5}\right)$$
$$= 0.97095$$

$$E_{overcast} = 0$$

$$E_{rainny} = 0.97015$$

Now, we find the Information Gain

$$Gain(S, outlook) = Entropy(S) - \left[\frac{\sum P_i N_i}{P+N} \times E(v) \text{ --- }\right]$$

$$G(s,o) = 0.94029 - \left[\frac{5}{14} \times 0.97095 + \frac{4}{14} \times 0 + \frac{5}{14} \times 0.97095\right]$$

$$\boxed{Gain(S, outlook) = 0.24675}$$

Now, we calculate the Gain of Temperature.

Hot = 4 = [2+ 2-] $\Rightarrow E = 1$

Mid = 6 = [4+ 2-] $\Rightarrow E = 0.91829$

cool = 4 = [3+ 1-] $\Rightarrow E = 0.811278$

$$Gain(S, Temp) = 0.94029 - \left[\frac{4}{14} \times 1 + \frac{6}{14} \times 0.91829 + \frac{4}{16} \times 0\right.$$

$$\boxed{Gain(S, Temp) = 0.02922}$$

Now, we calculate the Gain of Humidity.

High = 7 = [3+ 4-] => E =

Normal = 7 = [6+ 1-] => E =

Gain(S, Humidity) = 0.94029 − [

Gain(S, Humidity) = 0.02522

Now, we calculate the Gain of Wind:

Weak = 8 [6+ 2-] => E =

Strong = 6 [3+ 3-] => E =

Gain(S, Wind) = 0.94029 − [

Gain(S, Wind) = 0.093061

Which value is greater, than will mak a root node

Now, we find the Entropy for Sunny -

$Sample_{Sunny} = 5 = [2+ \ 3-]$

$Entropy_{(Sunny)} = 0.97095$

Find the gain of all left columns like, Temp, wind etc

$Gain(Sunny, Temp) = ?$

Hot = 2 = [0+ 2-] $\Rightarrow E = 0$

Mid = 2 = [1- 1+] $\Rightarrow E = 1$

Cool = 1 = [1+ 0-] $\Rightarrow E = 0$

$$Gain(Sunny, Temp) = 0.97095 - \left[\frac{2}{5} \times 0 + \frac{2}{5} \times 1 + \frac{1}{5} \times 0\right]$$

$$\boxed{Gain(S, Temp) = 0.57095}$$

Now, we calculate gain, sunny for humidity.

High = 3 = [0+ 3-] $\Rightarrow E = 0$

Normal = 2 = [2+ 0-] $\Rightarrow E = 0$

$$Gain(S, hum) = 0.97095 - \left(\frac{3}{5} \times 0 + \frac{2}{5} \times 0\right)$$

$$= 0.97095$$

- This is the higher entropy for Outlook.

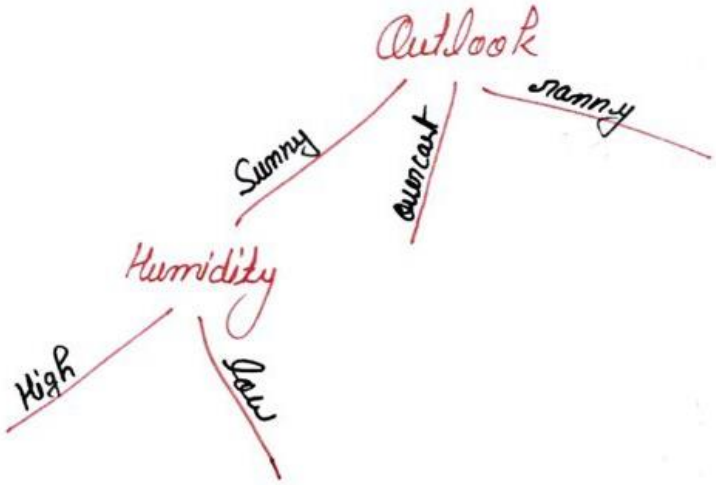
Now, we calculate gain, sunny for wind :

Weak = 3 [1+ 2-] => E = 0.918295

Strong = 2 [1+ 1-] => E = 1

Gain (S, Wind) = 0.019973

Then I will draw the next node of the tree by check the higher Entropy value.



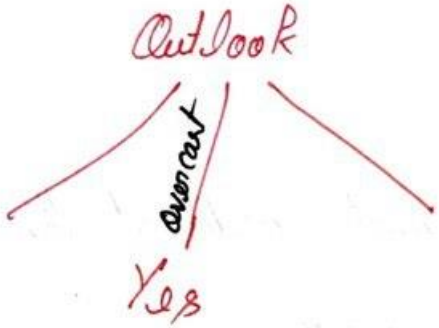
Now, we check the entropy of High and normal :

Sample (High) = 3 = [0+ 3-] => E = 0 [Stop]

Sample (Normal) = 2 = [2+ 0-] = E = 0 [Stop]

Entropy for Overcast :

Entropy (Overcast) = 0



Entropy for Rain :

S = 5 = [3+ 2-]

Entropy (Rainy) = 0.97085

Gain (Rain, Temp) = ?

Mild = 3 = [2+ 1-]

Cool = 2 = [1+ 1-]

$E_{(mild)} = 0.918295$

$E_{(cool)} = 1$

$$\text{Gain (Rain, Temp)} = 0.97085 - \left[\frac{3}{5} \times 0.918295 + \frac{2}{5} \times 1\right]$$

$$= 0.019873$$

Gain (Rain, Humidity) = ?

High = 2 = [1+ 1-]

Normal = 3 = [2+ 1-]

$E_{High} = 1$

$E_{normal} = 0.918295$

$Gain(Rain, Humidity) = 0.97085 - \left(\frac{2}{5} \times 1 + \frac{3}{5} \times 0.918295\right)$

$= 0.019873$

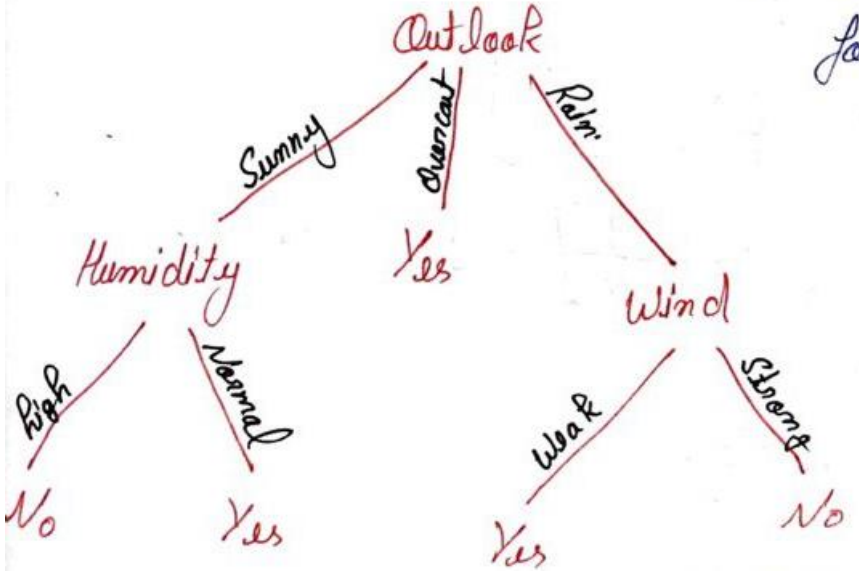
Gain (Rain, Wind) = ?

Weak = 3 = [3+ 0-] => E = 0

Strong = 2 = [0+ 2-] => E = 0

Gain (Rain, Wind) = 0.97085

This is the highest Entropy for Rain



Gini Index :

$$Gini(S) = 1 - \sum p^2$$

For whole data set - of output data.

Sunny $S = 14\ [9+\ 5-]$

$$Gini(S) = 1 - \left[\left(\frac{9}{14}\right)^2 + \left(\frac{5}{14}\right)^2\right]$$

$$= 1 - [0.41326 + 0.12755]$$

$$= 1 - 0.54081$$

$$Gini(S) = 0.45919$$

Gini index for each columns value

OutLook :

Sunny = 5 = [2 + 3-]
Overcast = 4 = [4 + 0-]
Rain = 5 = [3 + 2-]

$$Gini\ index\ (Sunny) = 1 - \left[\left(\frac{2}{4}\right)^2 + \left(\frac{3}{4}\right)^2\right]$$

$$= 1 - \left[\frac{4}{16} + \frac{9}{16}\right] = 1 - [0.25 + 0.562]$$

$$= 1 - 0.8125$$

$$GI(Sunny) = 0.1875$$

$GI(\text{Overcast}) = 4 = [4+\ 0-]$

$= 1 - \left[\left(\frac{4}{4}\right)^2 + \left(\frac{0}{4}\right)^2\right] \Rightarrow 1 - [0]$

$GI(\text{Overcast}) = 1$

$GI(\text{Rain}) = ?$

$S = S = [3+\ 2-]$

$= 1 - \left[\left(\frac{3}{5}\right)^2 + \left(\frac{2}{5}\right)^2\right]$

$= 1 - [0.36 + 0.16] = 1 - 0.52$

$GI(\text{Rain}) = 0.48$

Weighted Average Outlook :

$GI_{\text{Column}} = \sum Pv \times (Gi(v))$

$GI(\text{outlook}) = 0.1875 \times \left(\frac{5}{14}\right) + 1 \times \left(\frac{4}{14}\right) + 0.48 \times \left(\frac{5}{14}\right)$

$= 0.1875 \times 0.3571 + 1 \times 0.28571 + 0.48 \times 0.35714$

$= 0.06696 + 0.28571 + 0.171428$

$GI(\text{outlook}) = 0.52409$

Temperature :

Hot = 4 = [2+ 2-]
Mild = 6 = [4+ 2-]
Cool = 4 = [3+ 1-]

$GI(Hot) = 1 - [0.25 + 0.25] = 0.5$

$GI(Mild) = 1 - [0.444 + 0.111] = 0.4448$

$GI(Cool) = 1 - [0.562 + 0.062] = 0.3755$

$Gini(Temp) = \frac{4}{14} \times 0.5 + \frac{6}{14} \times 0.444 + \frac{4}{14} \times 0.375$

$= 0.142 + 0.190 + 0.107$

GI (Temp) = 0.439

Humidity :

High = 3 = [0+ 3-] = GI = 1
normal = 2 = [2+ 0-] = GI = 1

$GI(Humidity) = \frac{3}{14} \times 1 + \frac{2}{14} \times 1$

$= 0.2142 + 0.1428$

GI (Humidity) = 0.357

Wind :

Weak = 8 = [6+2-]

Strong = 6 = [3+3-]

GI(Weak) = 1 - [0.5625 + 0.0625] => 0.375

GI(Strong) = 1 - [0.25 + 0.25] => 0.5

$$GI(Wind) = \frac{8}{14} \times 0.375 + \frac{6}{14} \times 0.5$$

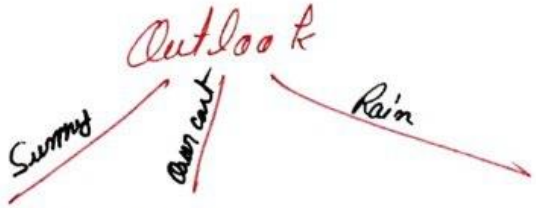
$$= 0.2142 + 0.2142$$

$$\boxed{GI(Wind) = 0.4285}$$

All Gini Index Value :

GI	Value
Outlook	= 0.34 (Correct value)
Temp	= 0.43
Humidity	= 0.35
Wind	= 0.42

And now assuming the smallest value of GI And than make G.I is root Node -



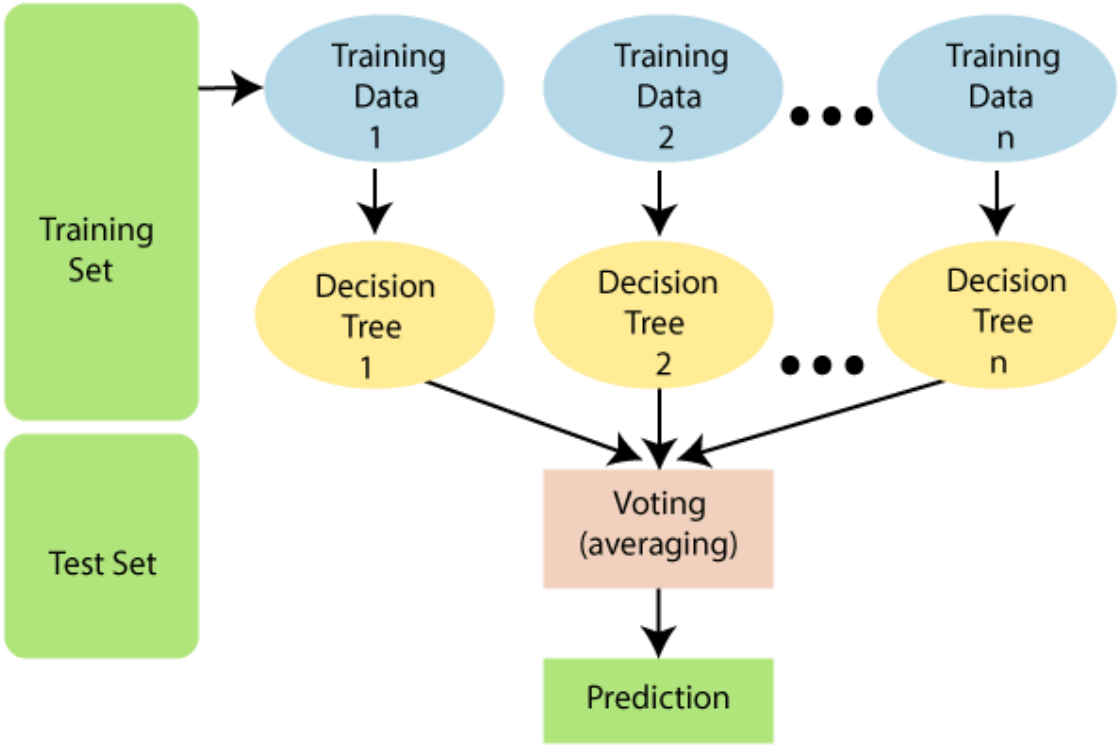
After this all the process going as well as Entropy.

Random Forest

Random Forest is a trademark term for an ensemble of decision trees. In Random Forest, we've collection of decision trees (so known as "Forest"). To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

- The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.



- Random Forest is a trademark term for an ensemble of decision trees. In Random Forest, we've collection of decision trees (so known as "Forest").
- To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Each tree is planted & grown as follows:

- If the number of cases in the traning set is N, then sample of N cases is taken at random but with replacement. This sample will be the traning set for growing the tree.
- If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
- Each tree is grown to the largest extent possible. There is no pruning.

Assumptions for Random Forest

- Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:
- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

Below are some points that explain why we should use the Random Forest algorithm:

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

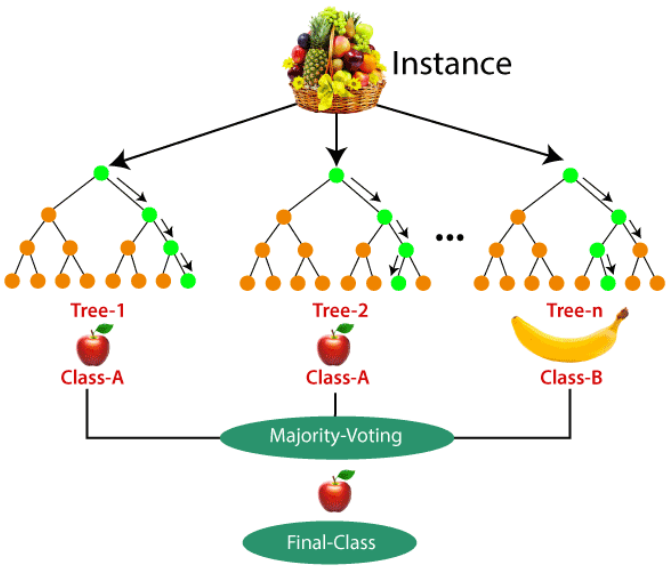
Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

- Step-1: Select random K data points from the training set.
- Step-2: Build the decision trees associated with the selected data points (Subsets).
- Step-3: Choose the number N for decision trees that you want to build.
- Step-4: Repeat Step 1 & 2.
- Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

Example:

- Suppose there is a dataset that contains multiple fruit images. So, this dataset is given to the Random forest classifier. The dataset is divided into subsets and given to each decision tree. During the training phase, each decision tree produces a prediction result, and when a new data point occurs, then based on the majority of results, the Random Forest classifier predicts the final decision. Consider the below image:



There are mainly four sectors where Random forest mostly used:

- Banking: Banking sector mostly uses this algorithm for the identification of loan risk.
- Medicine: With the help of this algorithm, disease trends and risks of the disease can be identified.
- Land Use: We can identify the areas of similar land use by this algorithm.
- Marketing: Marketing trends can be identified using this algorithm.

Advantages of Random Forest

- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue.

Disadvantages of Random Forest

- Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

Implementation Steps are given below:

- Data Pre-processing step
- Fitting the Random forest algorithm to the Training set
- Predicting the test result
- Test accuracy of the result (Creation of Confusion matrix)
- Visualizing the test set result.

In [2]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

In [3]:
```
# importing datasets
data = pd.read_csv('d:gini_index.csv')
data.head()
```

Out[3]:

	outlook	temp	humidity	wind	decision
0	sunny	hot	high	weak	no
1	sunny	hot	high	strong	no
2	overcast	hot	high	weak	yes
3	rain	mild	high	weak	yes
4	rain	cool	normal	weak	yes

In [4]:
```
# change the data in numeric form
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for i in data.columns:
    data[i]=le.fit_transform(data[i])
```

In [5]:
```
data.head()
```

Out[5]:

	outlook	temp	humidity	wind	decision
0	2	1	0	1	0
1	2	1	0	0	0
2	0	1	0	1	1
3	1	2	0	1	1
4	1	0	1	1	1

In [6]:
```
# Extracting Independent and dependent Variable
x = data.drop('decision',axis=1)
y = data['decision']
```

In [7]:
```
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20)
```

In [8]:
```
# festure scaling
from sklearn.preprocessing import StandardScaler
st_x = StandardScaler()
x_train = st_x.fit_transform(x_train)
x_test = st_x.fit_transform(x_test)
```

In [9]:
```
# Fitting Random Forest classifier to the training set
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=10,criterion='entropy')
model.fit(x_train,y_train)
```

Out[9]:
```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

- n_estimators= The required number of trees in the Random Forest. The default value is 10. We can choose any number but need to take care of the overfitting issue.
- criterion= It is a function to analyze the accuracy of the split. Here we have taken "entropy" for the information gain.

In [10]:
```
# Predicting the test set result
y_pred = model.predict(x_test)
```

In [11]:
```
# Creating the Confusion matrix
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,y_pred))

print(classification_report(y_test,y_pred))
```

```
[[0 1]
 [1 1]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      0.50      0.50         2

    accuracy                           0.33         3
   macro avg       0.25      0.25      0.25         3
weighted avg       0.33      0.33      0.33         3
```

Difference b/w Decission tree and Random Forest

Decision trees

1. Decision trees normally suffer from the problem of overfitting if it's allowed to grow without any control
2. A single decision tree is faster in computation.
3. When a data set with features is taken as input by a decision tree it will formulate some set of rules to do prediction.

Random Forest

1. Random forests are created from subsets of data and the final output is based on average or majority ranking and hence the problem of overfitting is taken care of.
2. It is comparatively slower.
3. Random forest randomly selects observations, builds a decision tree and the average result is taken. It doesn't use any set of formulas.

Following hyperparameters increases the predictive power:

1. **n_estimators –** number of trees the algorithm builds before averaging the predictions.
2. **max_features –** maximum number of features random forest considers splitting a node.
3. **mini_sample_leaf –** determines the minimum number of leaves required to split an internal node.

Following hyperparameters increases the speed:

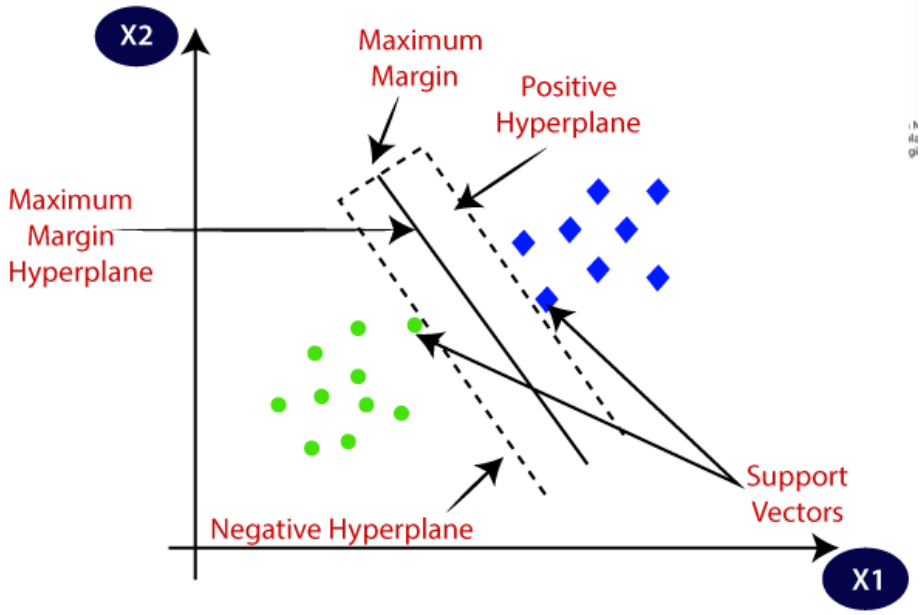
1. **n_jobs** – it tells the engine how many processors it is allowed to use. If the value is 1, it can use only one processor but if the value is -1 there is no limit.
2. **random_state –** controls randomness of the sample. The model will always produce the same results if it has a definite value of random state and if it has been given the same hyperparameters and the same training data.
3. **oob_score –** OOB means out of the bag. It is a random forest cross-validation method. In this one-third of the sample is not used to train the data instead used to evaluate its performance. These samples are called out of bag samples.

Support Vector Machine (SVM)

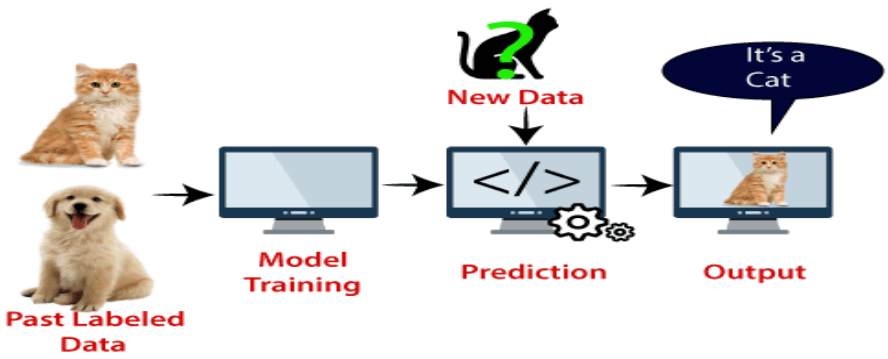
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

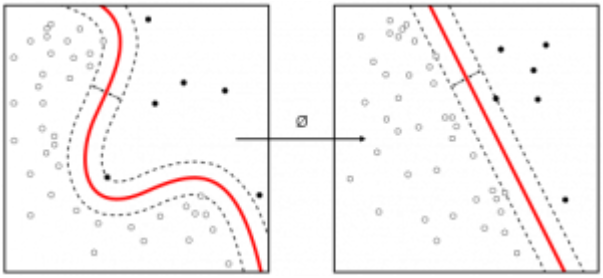
SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:

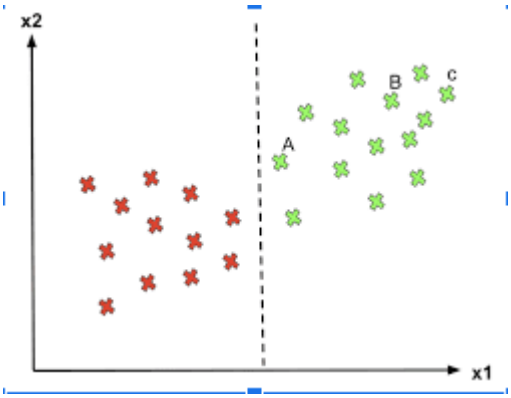


- There is also a subset of SVM called SVR which stands for Support Vector Regression which uses the same principles to solve regression problems.
- SVM is most commonly used and effective because of the use of the Kernel method which basically helps in solving the non-linearity of the equation in a very easy manner.



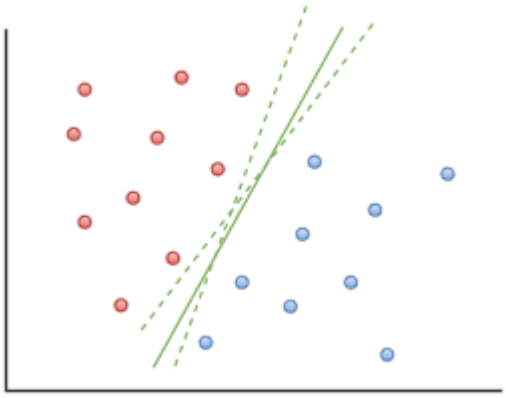
- The equation of the main separator line is called a hyperplane equation.
- A classification problem can have only two (binary) classes for separating or can have more than two too which are known as a multi-class classification problems.
- But not all classification predictive models support multi-class classification, algorithms such as the Logistic Regression and Support Vector Machines (SVM) were designed for binary classification and do not natively support classification tasks with more than two classes.
- But if someone stills want to use the binary classification algorithms for multi-classification problems, one approach which is widely used is to split the multi-class classification datasets into multiple binary classification datasets and then fit a binary classification model on each.
- As already mentioned above, SVM works much better for binary class
- It would be easy to understand the math since our target variable (variable / unseen data targeted to predict, whether the point is a male or a female)
- Note: This will be a One Vs One approach.

Case 1: (Perfect Separation for Binary Classified data)



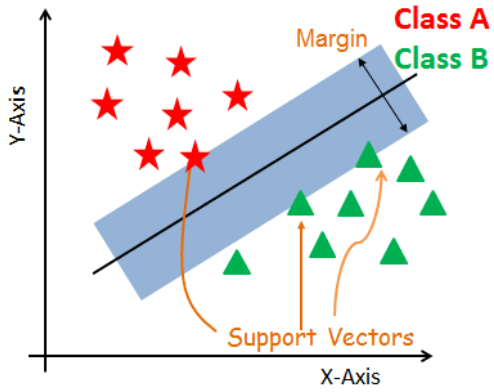
- Continuing with our example, if the hyperplane will be able to differentiate between males and females perfectly without doing any miss-classification, then that case of separation is known as Perfect Separation.
- Here, in the figure, if males are green and females are red and we can see that the hyperplane which is a line here has perfectly differentiated the two classes.

- (Training data — Data through which algorithm/model learns the pattern on how to differentiate by looking at the features
- Testing data — After the model is trained on training data, the model is asked to predict the values for unseen data where only the features are given, and now the model will tell whether its a male or a female)



- Now, there could be many hyperplanes giving 100% accuracy, as seen in the photograph.
- “” So to choose the optimal/best hyperplane, place the hyperplane right at the center where the distance is maximum from the closest points and give the least test errors further. “”
- To notice: We have to aim at the least TEST errors and NOT TRAINING errors.
- So, we have to maximize the distance to give some space to the hyperplane equation which is also the goal / main idea behind SVM.

The goal of the algorithm involved behind SVM:



- Finding a hyperplane with the maximum margin (margin is basically a protected space around hyperplane equation) and algorithm tries to have maximum margin with the closest points (known as support vectors).
- In other words, “The goal is to maximize the minimum distance.”

SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Types of SVM

SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

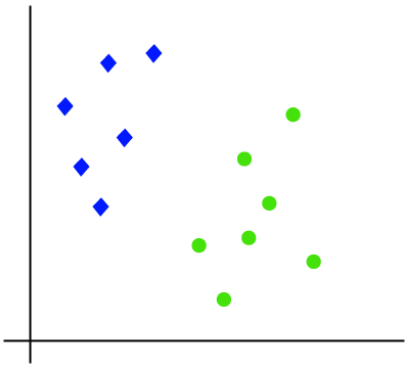
Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

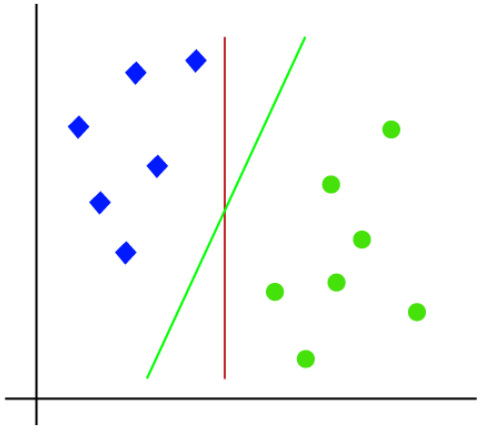
How does SVM works?

Linear SVM:

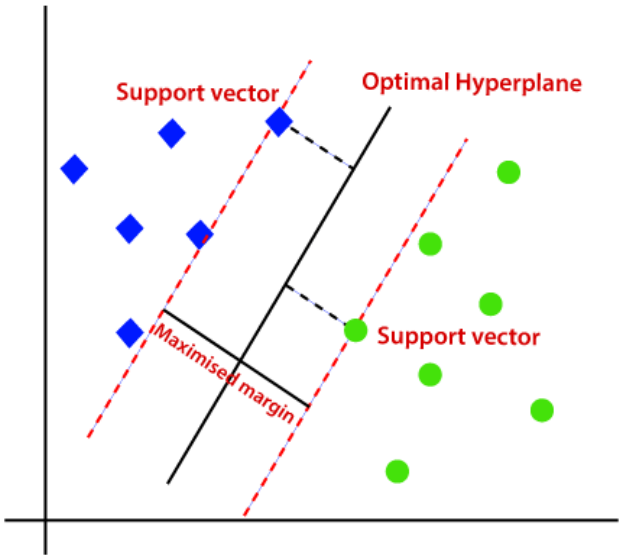
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

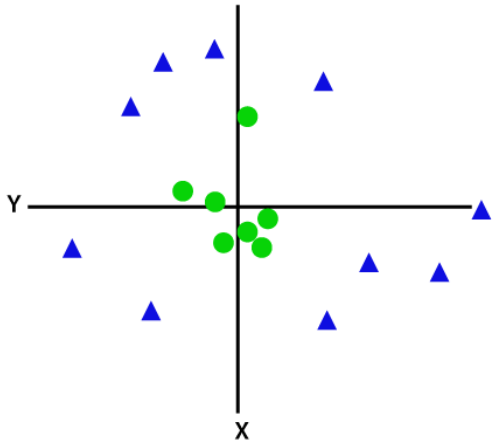


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.



Non-Linear SVM:

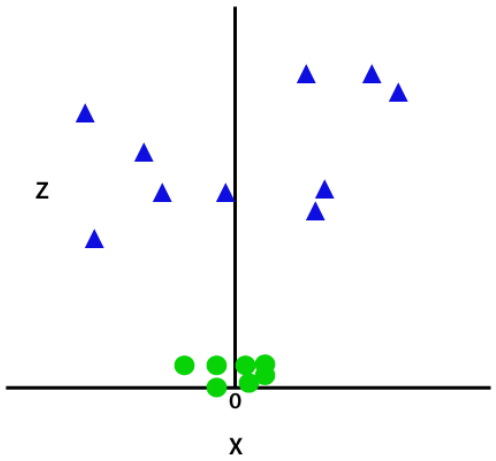
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



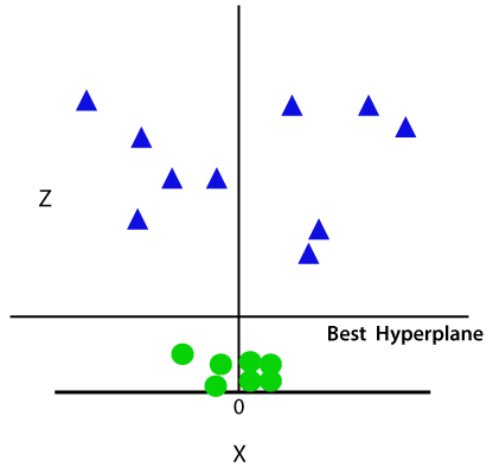
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$z=x^2 +y^2$

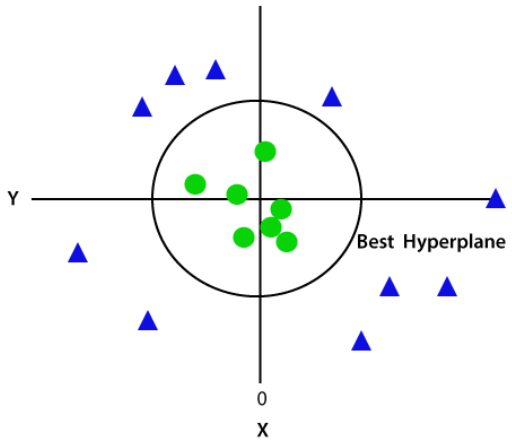
By adding the third dimension, the sample space will become as below image:



So now, SVM will divide the datasets into classes in the following way. Consider the below image:



Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:



Hence we get a circumference of radius 1 in case of non-linear data.

Major Kernel Functions in Support Vector Machine (SVM)

Kernel Function is a method used to take data as input and transform it into the required form of processing data. “Kernel” is used due to a set of mathematical functions used in Support Vector Machine providing the window to manipulate the data. So, Kernel Function generally transforms the training set of data so that a non-linear decision surface is able to transform to a linear equation in a higher number of dimension spaces. Basically, It returns the inner product between two points in a standard feature dimension.

Gaussian Kernel: It is used to perform transformation when there is no prior knowledge about data.

Gaussian Kernel Radial Basis Function (RBF): Same as above kernel function, adding radial basis method to improve the transformation.

Sigmoid Kernel: this function is equivalent to a two-layer, perceptron model of the neural network, which is used as an activation function for artificial neurons.

Polynomial Kernel: It represents the similarity of vectors in the training set of data in a feature space over polynomials of the original variables used in the kernel.

Linear Kernel: used when data is linearly separable.

SAMPLES

Name of the Kernel	Mathematical Formula
Linear	$k(x, y) = x^T . y$
Polynomial	$k(x, y) = (x^T, y)^P \; or \; k(x, y) = (x^T . y + 1)^p$ where p is the polynomial degree
RBF(Gaussian)	$\phi(x) = \exp(-\frac{x^2}{2\sigma^2}), \sigma > 0$

$$sigmoid\ function = \frac{1}{1 + e^{-(\Theta)}}$$
$$where\ \Theta = (1)x + 0$$

Python Implementation

In [2]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

In [3]:
```
data = pd.read_csv('d:bill_authentication.csv')
data.head()
```

Out[3]:

	Variance	Skewness	Curtosis	Entropy	Class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

In [4]:
```
data.shape
```

Out[4]:
```
(1372, 5)
```

In [6]:
```
data.isnull().sum()
```

Out[6]:
```
Variance    0
Skewness    0
Curtosis    0
Entropy     0
Class       0
dtype: int64
```

In [7]:
```
x = data.drop('Class',axis=1)
y = data['Class']
```

In [10]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

In [12]:
```
from sklearn.svm import SVC
model = SVC(kernel='linear')
```

In [13]:
```
model.fit(x_train,y_train)
```

Out[13]:
```
SVC(kernel='linear')
```

In [15]:
```
ypre = model.predict(x_test)
```

In [18]:
```
# comparing the result
df = pd.DataFrame({'acutal value':y_test,'predicted value':ypre})
df
```

Out[18]:

	acutal value	predicted value
660	0	0
326	0	0
1051	1	1
32	0	0
803	1	1
...	...	...
1171	1	1
702	0	0
786	1	1
1189	1	1
603	0	0

275 rows × 2 columns

In [21]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,ypre))
print(classification_report(y_test,ypre))
```

```
[[138   1]
 [  1 135]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       139
           1       0.99      0.99      0.99       136

    accuracy                           0.99       275
   macro avg       0.99      0.99      0.99       275
weighted avg       0.99      0.99      0.99       275
```

Naive Bayes Classifier

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve**: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes**: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B)= \frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No

12	Overcast	Yes
13	Overcast	Yes

Frequency table for the Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	4/14=0.29	10/14=0.71	

Applying Bayes'theorem:

P(Yes|Sunny)= P(Sunny|Yes)*P(Yes)/P(Sunny)

P(Sunny|Yes)= 3/10= 0.3

P(Sunny)= 0.35

P(Yes)=0.71

So P(Yes|Sunny) = 0.3*0.71/0.35= **0.60**

P(No|Sunny)= P(Sunny|No)*P(No)/P(Sunny)

P(Sunny|NO)= 2/4=0.5

P(No)= 0.29

P(Sunny)= 0.35

So P(No|Sunny)= 0.5*0.29/0.35 = **0.41**

So as we can see from the above calculation that **P(Yes|Sunny)>P(No|Sunny)**

Hence on a Sunny day, Player can play the game.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications of Naïve Bayes Classifier:

- It is used for **Credit Scoring**.
- It is used in **medical data classification**.
- It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.

Types of Naïve Bayes Model:

There are three types of Naive Bayes Model, which are given below:

- **Gaussian**: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.
- **Multinomial**: The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc. The classifier uses the frequency of words for the predictors.
- **Bernoulli**: The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

Python Implementation

In [1]:
```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
x = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

```
C:\Users\hp\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required fo
r this version of SciPy (detected version 1.23.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [2]:
```
# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train, y_train)
```

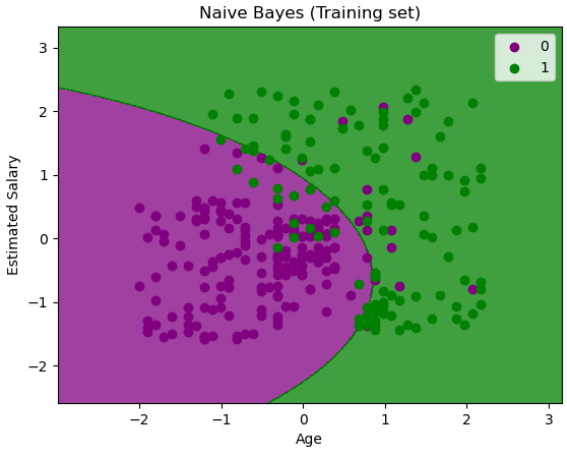
Out[2]:
```
GaussianNB()
```

In [3]:
```
# Predicting the Test set results
y_pred = classifier.predict(x_test)
```

In [4]:
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

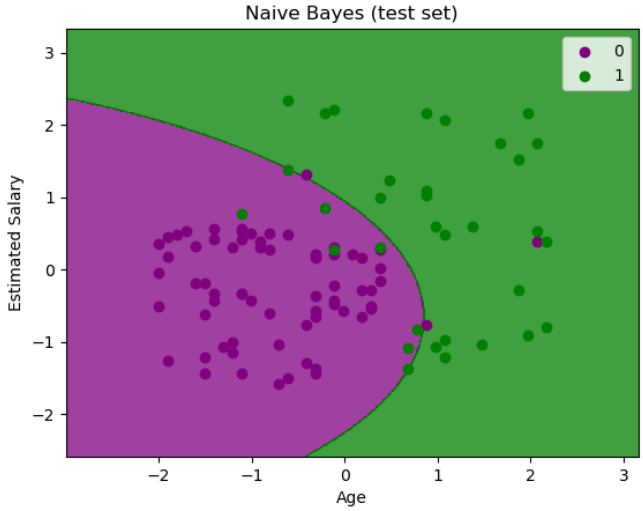
In [5]:
```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),
                     nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('purple', 'green')))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
                c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you
intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you
intend to specify the same RGB or RGBA value for all points.
```



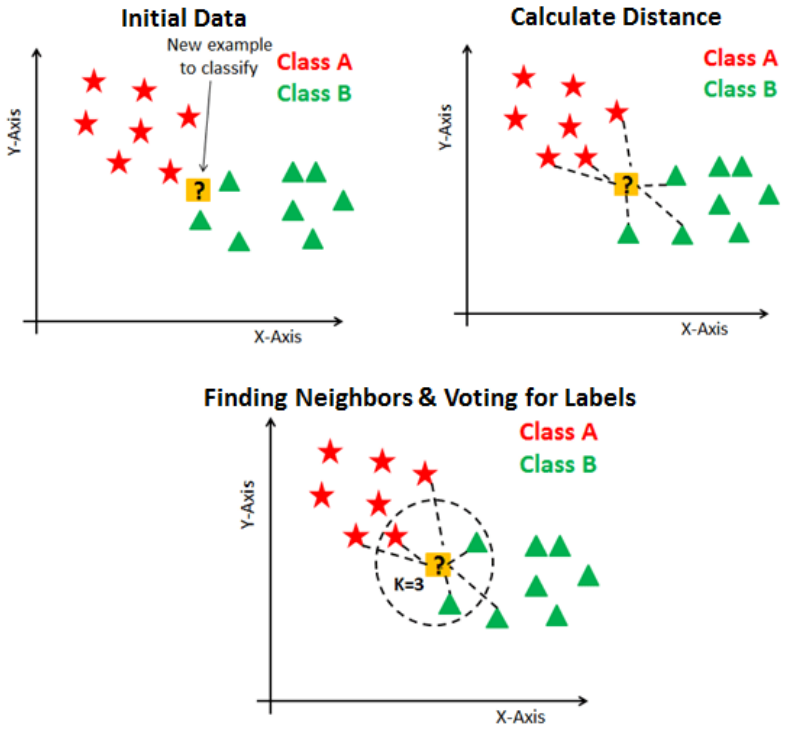
In [6]:
```
# Visualising the Test set results
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),
                     nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('purple', 'green')))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
                c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



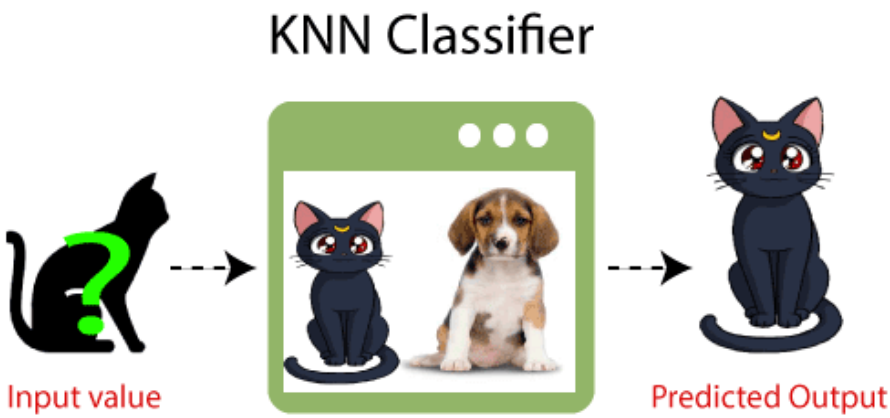
K-Nearest Neighbor(KNN)

KNN used in the variety of applications such as finance, healthcare, political science, handwriting detection, image recognition and video recognition. In Credit ratings, financial institutes will predict the credit rating of customers. In loan disbursement, banking institutes will predict whether the loan is safe or risky. In political science, classifying potential voters in two classes will vote or won't vote. KNN algorithm used for both classification and regression problems. KNN algorithm based on feature similarity approach.



- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.



Pros

- The training phase of K-nearest neighbor classification is much faster compared to other classification algorithms. There is no need to train a model for generalization, That is why KNN is known as the simple and instance-based learning algorithm. KNN can be useful in case of nonlinear data. It can be used with the regression problem. Output value for the object is computed by the average of k closest neighbors value.

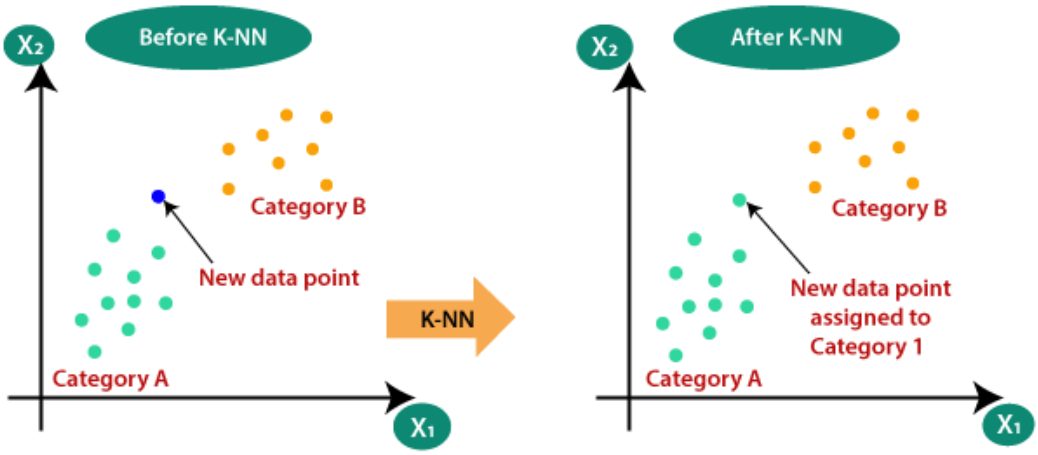
Cons

- The testing phase of K-nearest neighbor classification is slower and costlier in terms of time and memory. It requires large memory for storing the entire training dataset for prediction. KNN requires scaling of data because KNN uses the Euclidean distance between two data points to find nearest neighbors. Euclidean distance is sensitive to magnitudes. The features with high magnitudes will weight more than features with low magnitudes. KNN also not suitable for large dimensional data.

- For better results, normalizing data on the same scale is highly recommended. Generally, the normalization range considered between 0 and 1. KNN is not suitable for the large dimensional data. In such cases, dimension needs to reduce to improve the performance. Also, handling missing values will help us in improving results.

Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

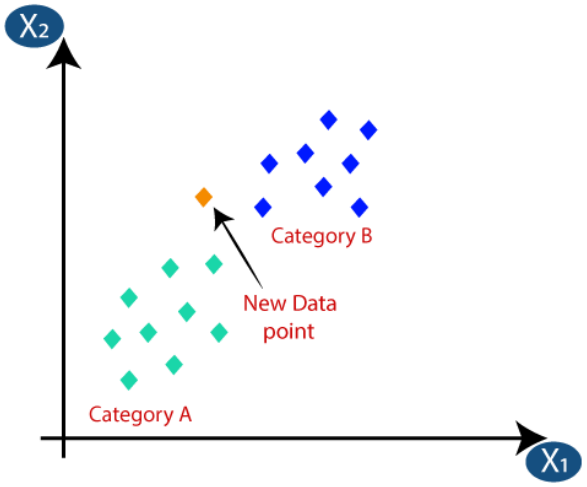


How does K-NN work?

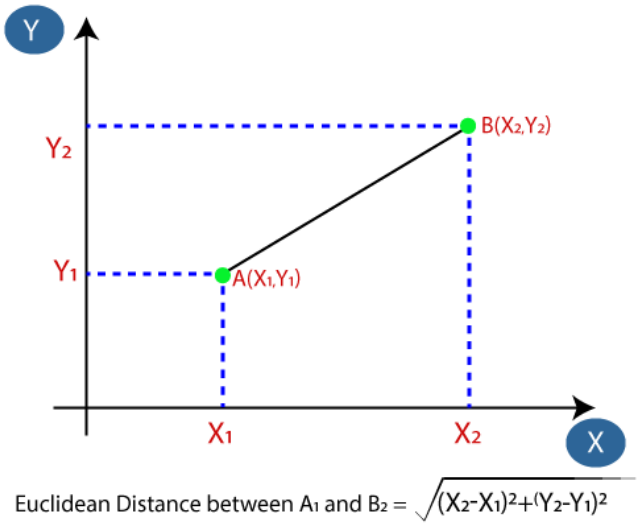
The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

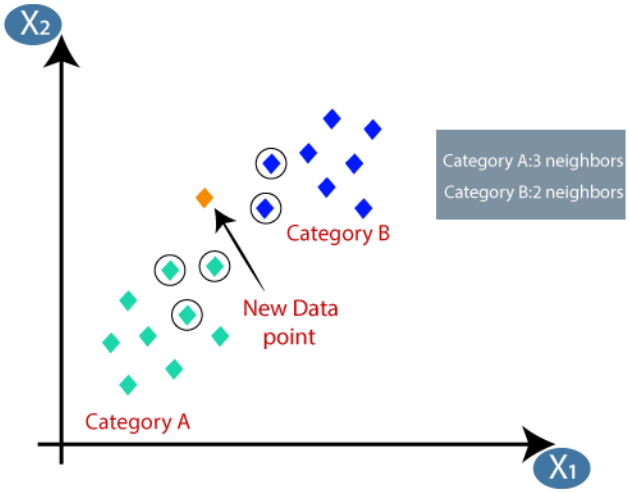
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

Python implementation of the KNN algorithm

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

In [3]:
```
df =  pd.read_csv('d:gini_index.csv')
df
```

Out[3]:

	outlook	temp	humidity	wind	decision
0	sunny	hot	high	weak	no
1	sunny	hot	high	strong	no
2	overcast	hot	high	weak	yes
3	rain	mild	high	weak	yes
4	rain	cool	normal	weak	yes
5	rain	cool	normal	strong	no
6	overcast	cool	normal	strong	yes
7	sunny	mild	high	weak	no
8	sunny	cool	normal	weak	yes
9	rain	mild	normal	weak	yes
10	sunny	mild	normal	strong	yes
11	overcast	mild	high	strong	yes
12	overcast	hot	normal	weak	yes
13	rain	mild	high	strong	no

In [4]:
```
# change the categorical column into numerical
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for i in df.columns:
    df[i]=le.fit_transform(df[i])
```

In [8]:
```
x = df.drop('decision',axis=1)
y = df['decision']
```

In [9]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.20)
```

In [10]:
```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
```

In [11]:
```
model.fit(x_train,y_train)
```

Out[11]:
```
KNeighborsClassifier()
```

In [13]:
```
ypre = model.predict(x_test)
```

In [14]:
```
data = pd.DataFrame({'actual':y_test,'predicted':ypre})
```

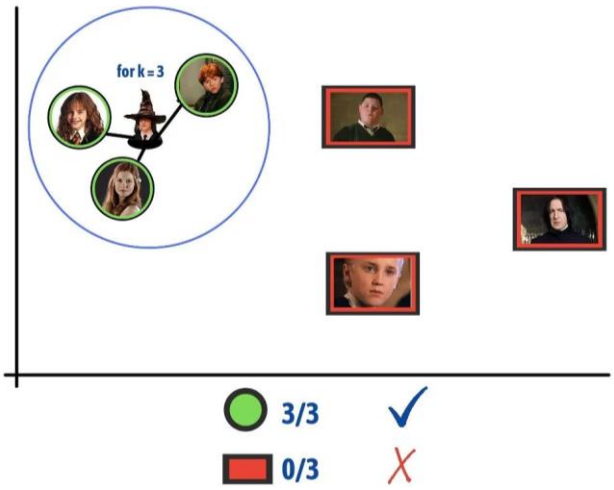
In [15]:
```
data
```

Out[15]:

	actual	predicted
7	0	0
12	1	1
10	1	0

In [16]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,ypre)
```

Out[16]:
```
0.6666666666666666
```



- It can be used for both classification and regression problem .
- However, it is more widely used in classification problems in the industry.
- K nearest neighbors is a simple algorithm that stores all the data points and predict the new data point by looking at the K nearest neighbors measured by a distance function.
- K-nearest neighbors are computed using a distance function.
- These distance functions can be **Eucildean, Manhattan, Minkowski, and Hamming distance.**
- The first three functions are used for continuous variables and the fourth one (Hamming) for categorical variables.
- If K =1, then the case is simple assigned to the class of its nearest neighbour. At times, choosing K turns out to be a challenge while performing kNN modelling.
- KNN can easily be mapped to our real lives. If you want to learn about a person, of whom you have no information, you might like to find out about his close friends and the circles he moves in and gain access to his/her information.

Things to consider before selecting KNN:

- KNN is computationally expensive
- Variable should be normalized else higher range variables can bias it
- Works on pre-processing stage more before going for kNN like an outlier, noise removal

Ensemble learning

Ensemble learning is the process of **training multiple machine learning models and combining their outputs together**. The different models are used as a base to create one optimal predictive model.

(Multiple (different) Machine Learning models working together as a group)

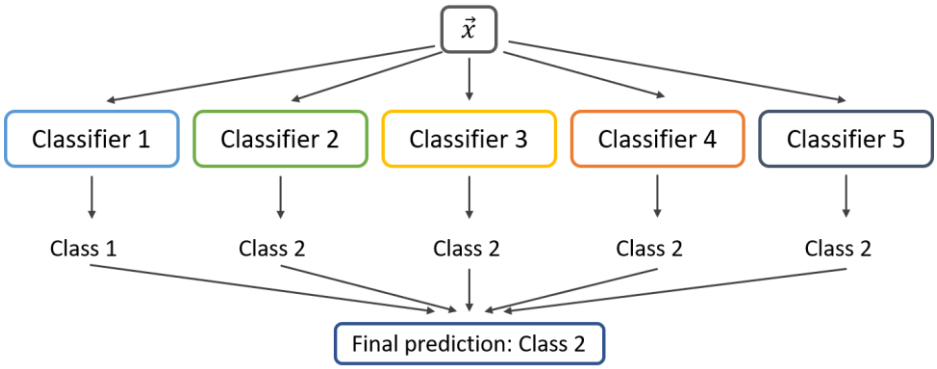
Simple Ensemble Techniques

In this section, we will look at a few simple but powerful techniques, namely:

1. Max Voting
2. Averaging
3. Weighted Averaging

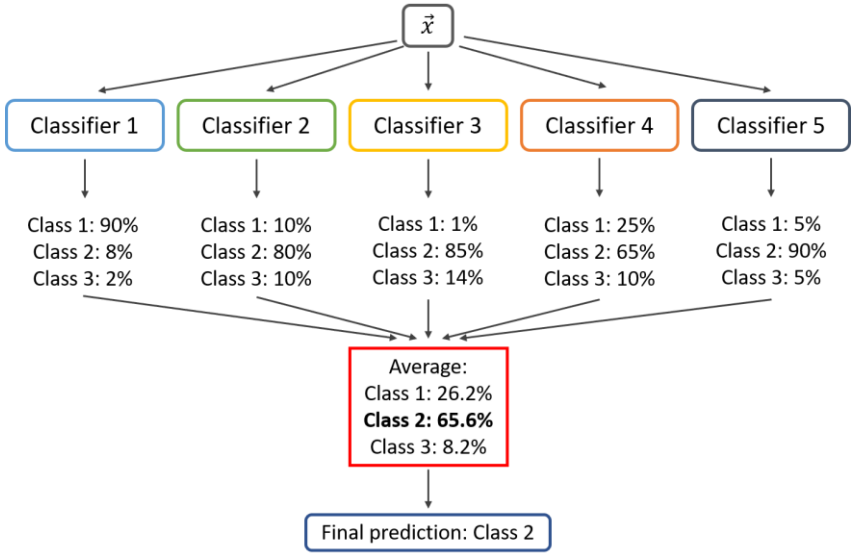
Max Voting

The max voting method is generally used for classification problems. In this technique, multiple models are used to make predictions for each data point. The predictions by each model are considered as a 'vote'. The predictions which we get from the majority of the models are used as the final prediction.



Averaging

Similar to the max voting technique, multiple predictions are made for each data point in averaging. In this method, we take an average of predictions from all the models and use it to make the final prediction.



Weighted Average

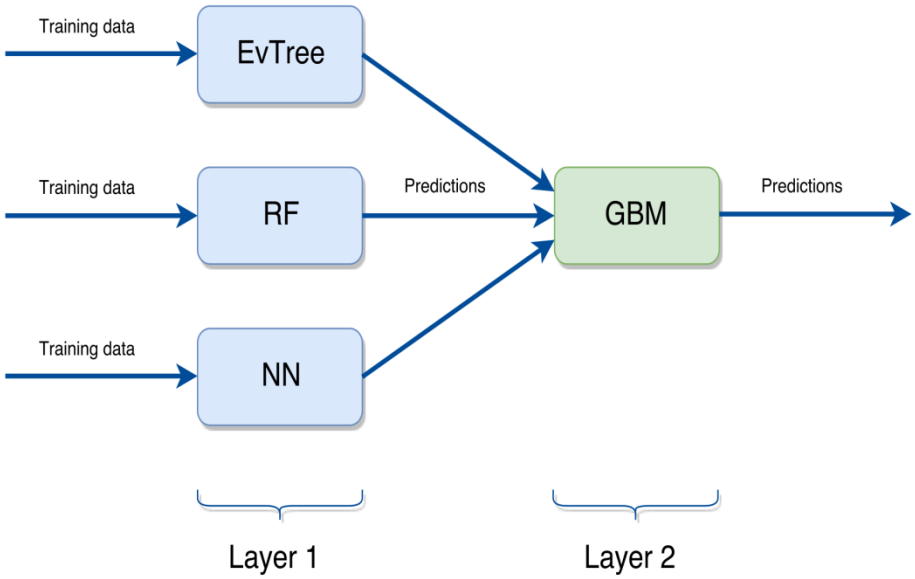
This is an extension of the averaging method. All models are assigned different weights defining the importance of each model for prediction. For instance, if two of your colleagues are critics, while others have no prior experience in this field, then the answers by these two friends are given more importance as compared to the other people.

Advanced Ensemble Techniques

1. Stacking
2. Bagging
3. Boosting

Stacking

Stacking is an ensemble learning technique that uses predictions from multiple models (for example decision tree, knn or svm) to build a new model. This model is used for making predictions on the test set. Below is a step-wise explanation for a simple stacked ensemble.



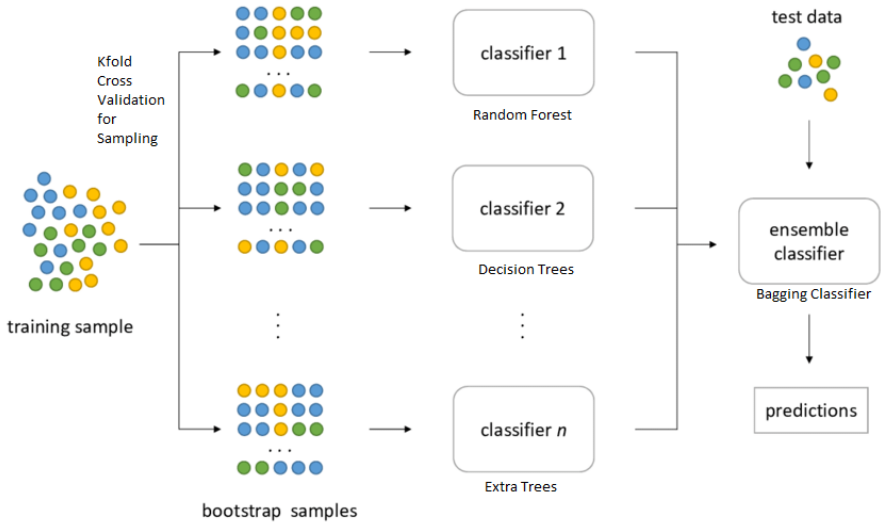
Bagging

Bagging or bootstrap averaging is a technique where multiple models are created on the subset of data, and the final predictions are determined by combining the predictions of all the models.

There is a high chance that these models will give the same result since they are getting the same input. So how can we solve this problem? One of the techniques is bootstrapping.
Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, with replacement. The size of the subsets is the same as the size of the original set.

Some of the algorithm that used bagging technique are:

- Bagging meta-estimator
- Random Forest



Bagging Classifier Process Flow

Boosting

Boosting algorithms are one of the most widely used algorithm in data science competitions.
The term 'Boosting' refers to a family of algorithms which converts weak learner to strong learners.

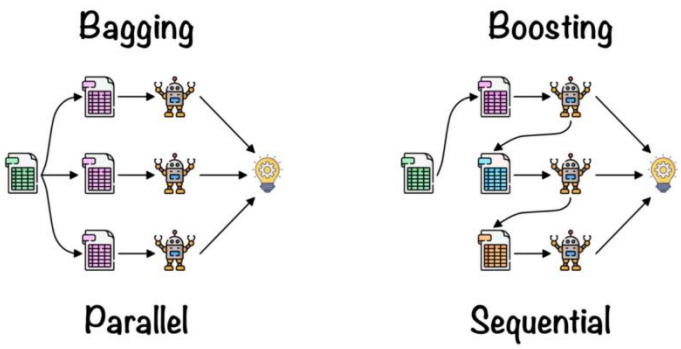
To convert weak learner to strong learner, we'll combine the prediction of each weak learner using methods like:

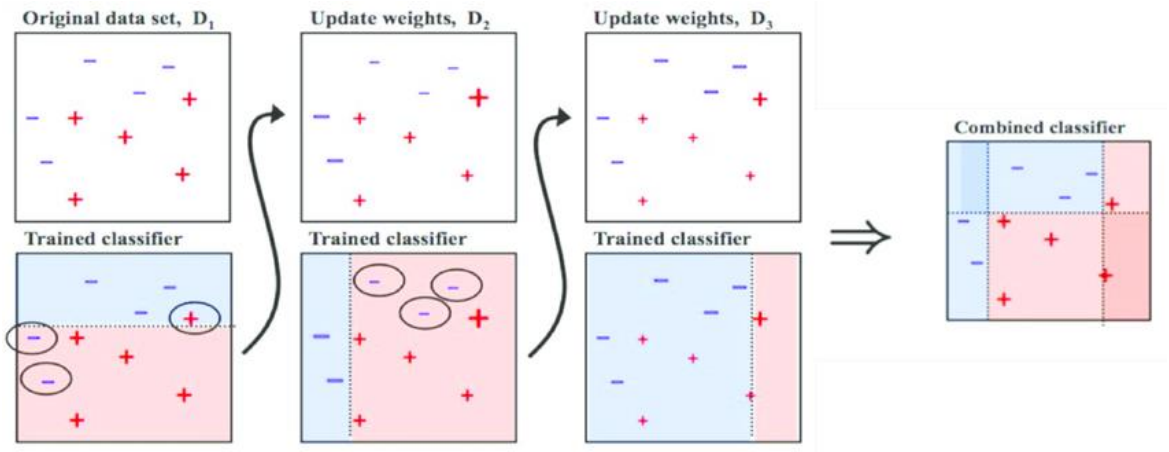
- Using average/weighted average
- Considering prediction has higher vote

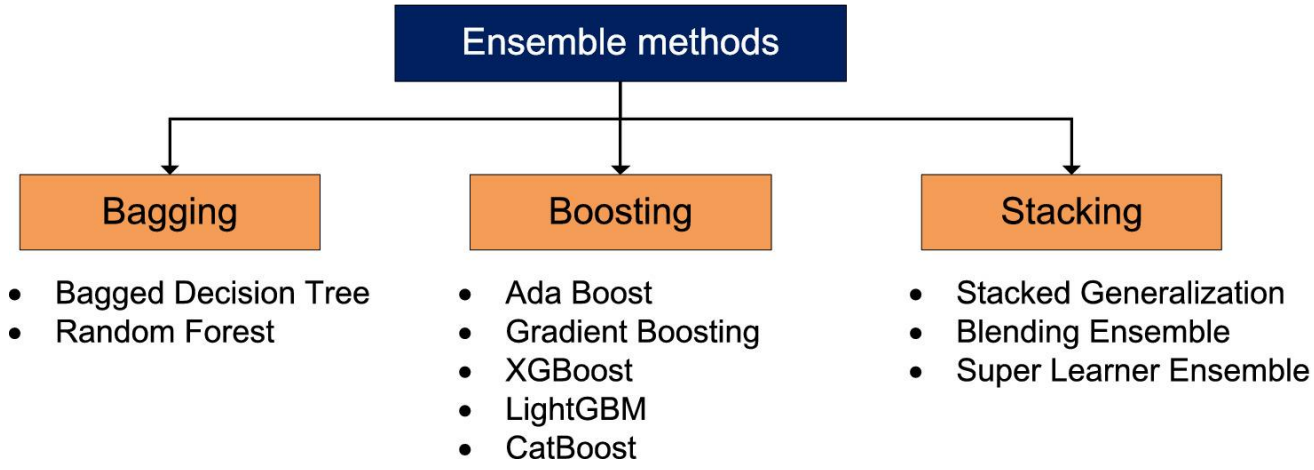
For example: Above, we have defined 5 weak learners. Out of these 5, 3 are voted as 'SPAM' and 2 are voted as 'Not a SPAM'. In this case, by default, we'll consider an email as SPAM because we have higher(3) vote for 'SPAM'.

Boosting algorithms:

AdaBoost
GBM
XGBM
Light GBM
CatBoost







Bagging algorithms!

1) Bagging meta-estimator

Bagging meta-estimator is an ensembling algorithm that can be used for both classification (BaggingClassifier) and regression (BaggingRegressor) problems. It follows the typical bagging technique to make predictions. Following are the steps for the bagging meta-estimator algorithm:

1. Random subsets are created from the original dataset (Bootstrapping).
2. The subset of the dataset includes all features.
3. A user-specified base estimator is fitted on each of these smaller sets.
4. Predictions from each model are combined to get the final result.

Code:

```
from sklearn.ensemble import BaggingClassifier
from sklearn import tree
model = BaggingClassifier(tree.DecisionTreeClassifier(random_state=1))
model.fit(x_train, y_train)
model.score(x_test,y_test)
0.75135135135135134
```

Sample code for regression problem:

```
from sklearn.ensemble import BaggingRegressor
model = BaggingRegressor(tree.DecisionTreeRegressor(random_state=1))
model.fit(x_train, y_train)
model.score(x_test,y_test)
```

---

2) Random Forest

Random Forest is another ensemble machine learning algorithm that follows the bagging technique. It is an extension of the bagging estimator algorithm.
The base estimators in random forest are decision trees. Unlike bagging meta estimator, random forest randomly selects a set of features which are used to decide the best split at each node of the decision tree.

Looking at it step-by-step, this is what a random forest model does:

1. Random subsets are created from the original dataset (bootstrapping).
2. At each node in the decision tree, only a random set of features are considered to decide the best split.
3. A decision tree model is fitted on each of the subsets.
4. The final prediction is calculated by averaging the predictions from all decision trees.

Note: The decision trees in random forest can be built on a subset of data and features. Particularly, the sklearn model of random forest uses all features for decision tree and a subset of features are randomly selected for splitting at each node.

Boosting algorithms!

1) AdaBoost

Adaptive boosting or AdaBoost is one of the simplest boosting algorithms. Usually, decision trees are used for modelling. Multiple sequential models are created, each correcting the errors from the last model. AdaBoost assigns weights to the observations which are incorrectly predicted and the subsequent model works to predict these values correctly.

Below are the steps for performing the AdaBoost algorithm:

1. Initially, all observations in the dataset are given equal weights.
2. A model is built on a subset of data.
3. Using this model, predictions are made on the whole dataset.
4. Errors are calculated by comparing the predictions and actual values.
5. While creating the next model, higher weights are given to the data points which were predicted incorrectly.
6. Weights can be determined using the error value. For instance, higher the error more is the weight assigned to the observation.
7. This process is repeated until the error function does not change, or the maximum limit of the number of estimators is reached.

Code:

```
from sklearn.ensemble import AdaBoostClassifier
model = AdaBoostClassifier(random_state=1)
model.fit(x_train, y_train)
model.score(x_test,y_test)
0.81081081081081086
```

Sample code for regression problem:

```
from sklearn.ensemble import AdaBoostRegressor
model = AdaBoostRegressor()
model.fit(x_train, y_train)
model.score(x_test,y_test)
```

2) Gradient Boosting (GBM)

Gradient Boosting or GBM is another ensemble machine learning algorithm that works for both regression and classification problems. GBM uses the boosting technique, combining a number of weak learners to form a strong learner. Regression trees used as a base learner, each subsequent tree in series is built on the errors calculated by the previous tree.

Code:

```
from sklearn.ensemble import GradientBoostingClassifier
model= GradientBoostingClassifier(learning_rate=0.01,random_state=1)
model.fit(x_train, y_train)
model.score(x_test,y_test)
0.81621621621621621
```

Sample code for regression problem:

```
from sklearn.ensemble import GradientBoostingRegressor
model= GradientBoostingRegressor()
model.fit(x_train, y_train)
model.score(x_test,y_test)
```

---

3) XGBoost

XGBoost (extreme Gradient Boosting) is an advanced implementation of the gradient boosting algorithm. XGBoost has proved to be a highly effective ML algorithm, extensively used in machine learning competitions and hackathons. XGBoost has high predictive power and is almost 10 times faster than the other gradient boosting techniques. It also includes a variety of regularization which reduces overfitting and improves overall performance. Hence it is also known as 'regularized boosting' technique.
Let us see how XGBoost is comparatively better than other techniques:

Regularization:
Standard GBM implementation has no regularisation like XGBoost.
Thus XGBoost also helps to reduce overfitting.

Parallel Processing:
XGBoost implements parallel processing and is faster than GBM .
XGBoost also supports implementation on Hadoop.

High Flexibility:
XGBoost allows users to define custom optimization objectives and evaluation criteria adding a whole new dimension to the model.

Handling Missing Values:
XGBoost has an in-built routine to handle missing values.

Tree Pruning:
XGBoost makes splits up to the max_depth specified and then starts pruning the tree backwards and removes splits beyond which there is no positive gain.

Built-in Cross-Validation:
XGBoost allows a user to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.
Code:-

Since XGBoost takes care of the missing values itself, you do not have to impute the missing values. You can skip the step for missing value imputation from the code mentioned above. Follow the remaining steps as always and then apply xgboost as below.

```
import xgboost as xgb
model=xgb.XGBClassifier(random_state=1,learning_rate=0.01)
model.fit(x_train, y_train)
model.score(x_test,y_test)
0.82702702702702702
```

Sample code for regression problem:

```
import xgboost as xgb
model=xgb.XGBRegressor()
model.fit(x_train, y_train)
model.score(x_test,y_test)
```

---

4) Light GBM

Before discussing how Light GBM works, let's first understand why we need this algorithm when we have so many others (like the ones we have seen above). Light GBM beats all the other algorithms when the dataset is extremely large. Compared to the other algorithms, Light GBM takes lesser time to run on a huge dataset.

LightGBM is a gradient boosting framework that uses tree-based algorithms and follows leaf-wise approach while other algorithms work in a level-wise approach pattern. The images below will help you understand the difference in a better way.

Code:

```
import lightgbm as lgb
train_data=lgb.Dataset(x_train,label=y_train)
#define parameters
```

```
params = {'learning_rate':0.001}
model= lgb.train(params, train_data, 100)
y_pred=model.predict(x_test)
for i in range(0,185):
  if y_pred[i]>=0.5:
  y_pred[i]=1
else:
  y_pred[i]=0
0.81621621621621621
```

Sample code for regression problem:

```
import lightgbm as lgb
train_data=lgb.Dataset(x_train,label=y_train)
params = {'learning_rate':0.001}
model= lgb.train(params, train_data, 100)
from sklearn.metrics import mean_squared_error
rmse=mean_squared_error(y_pred,y_test)**0.5
```

---

5) CatBoost

Handling categorical variables is a tedious process, especially when you have a large number of such variables. When your categorical variables have too many labels (i.e. they are highly cardinal), performing one-hot-encoding on them exponentially increases the dimensionality and it becomes really difficult to work with the dataset.

Code:

CatBoost algorithm effectively deals with categorical variables. Thus, you should not perform one-hot encoding for categorical variables. Just load the files, impute missing values, and you're good to go.

```
from catboost import CatBoostClassifier
model=CatBoostClassifier()
categorical_features_indices = np.where(df.dtypes != np.float)[0]
model.fit(x_train,y_train,cat_features=([ 0,  1, 2, 3, 4, 10]),eval_set=(x_test, y_test))
model.score(x_test,y_test)
0.80540540540540539
```

Sample code for regression problem:

```
from catboost import CatBoostRegressor
model=CatBoostRegressor()
categorical_features_indices = np.where(df.dtypes != np.float)[0]
```

```
model.fit(x_train,y_train,cat_features=([ 0,  1, 2, 3, 4, 10]),eval_set=(x_test, y_test))
model.score(x_test,y_test)
```

K - Means

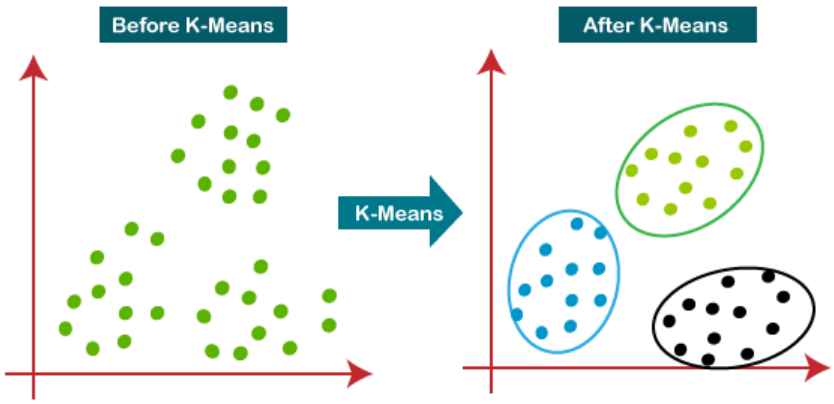
- It is a type of unsupervised algorithm which solves the clustering problem. Its procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters).
- Data points inside a cluster are homogeneous and heterogeneous to peer groups.
- Remember figuring out shapes from ink blots? K means is somewhat similar this activity. You look at the shape and spread to decipher how many different clusters / population are present!

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

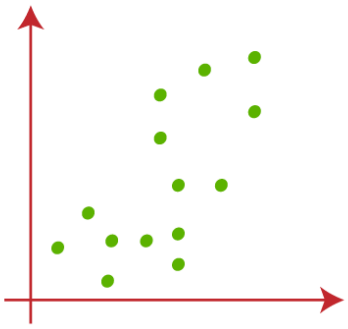
Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

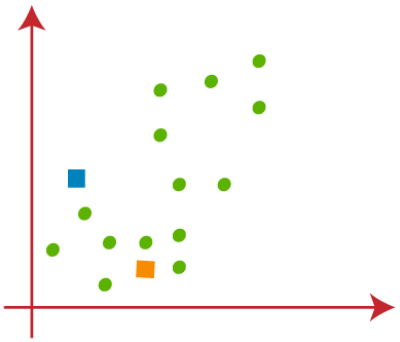
Step-7: The model is ready.

Let's understand the above steps by considering the visual plots:

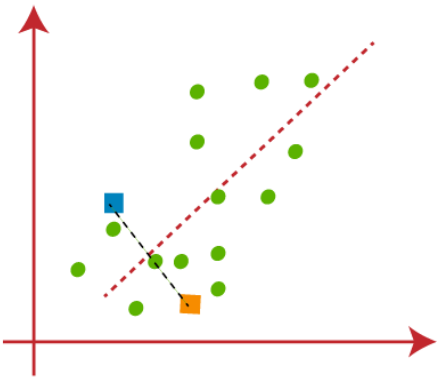
Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:



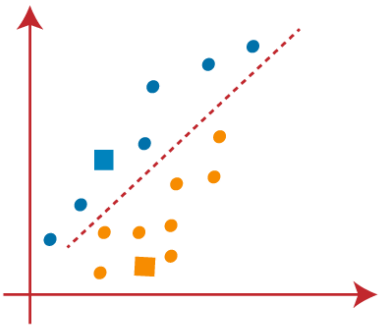
- Let's take number k of clusters, i.e., K=2, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points from the dataset or any other point. So, here we are selecting the below two points as k points, which are not the part of our dataset. Consider the below image:



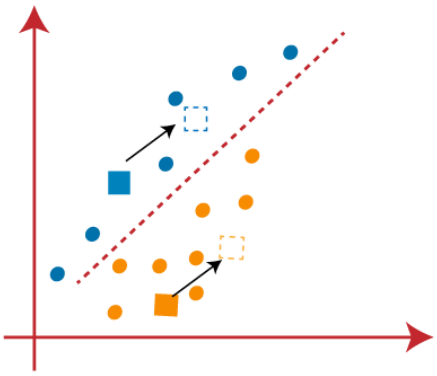
- Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids. Consider below image.



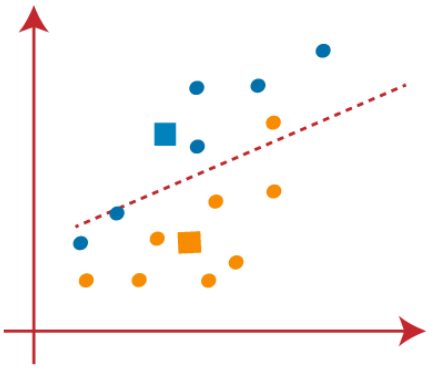
From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.



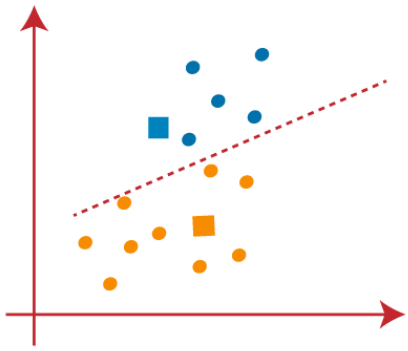
- As we need to find the closest cluster, so we will repeat the process by choosing **a new centroid**. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:



- Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below image:



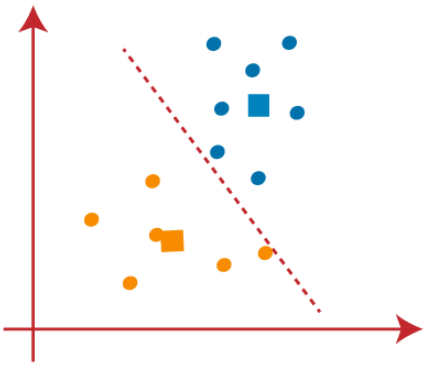
From the above image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids.



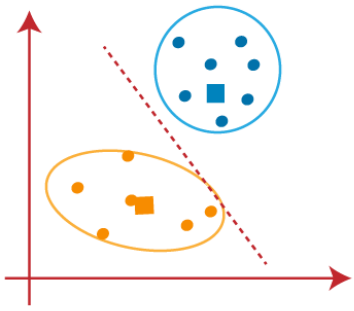
As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points.

- We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image:

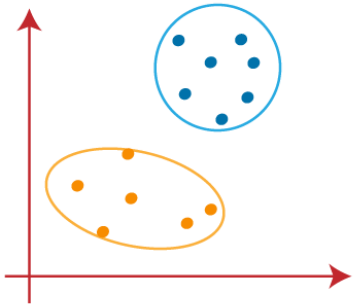
- As we got the new centroids so again will draw the median line and reassign the data points. So, the image will be:



- We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image:



As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image:



How to choose the value of "K number of clusters" in K-means Clustering?

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. But choosing the optimal number of clusters is a big task. There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

Elbow Method

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

$$WCSS= \sum_{Pi \text{ in } Cluster1} distance(P_i\ C_1)^2 +\sum_{Pi \text{ in } Cluster2} distance(P_i\ C_2)^2+\sum_{Pi \text{ in } CLuster3} distance(P_i\ C_3)^2$$

In the above formula of WCSS,

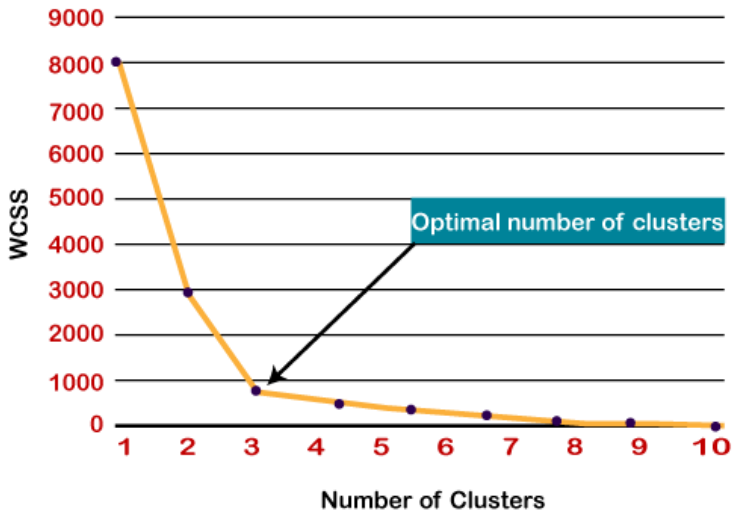
$\sum_{Pi \text{ in } Cluster1} distance(P_i\ C_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method. The graph for the elbow method looks like the below image:



Note: We can choose the number of clusters equal to the given data points. If we choose the number of clusters equal to the data points, then the value of WCSS becomes zero, and that will be the endpoint of the plot.

How K-means forms cluster

- K-means picks k number of points for each cluster known as centroids.
- Each data point forms a cluster based on existing cluster members. Here we have new centroids.
- As we have new centroids, repeat step 2 and 3. Find the closest distance for each data point from new centroids and get associated with new k-clusters.
- Repeat this process until convergence occours i.e. centroids does not change.

How to determine value of K

- In k-means, we have clusters and each cluster has its own centroid. Sum of square of difference between centroid and the data points within a cluster constitutes within sum of square value for that cluster, Also, when the sum of square values for all the cluster are added, it becomes total within sum of square value for the cluster solution.
- We know that as the number of cluster increases, this value keeps on decreasing but if you plot the result you may see that the sum of squared distance decreases sharply up to some value of k, and then much more slowly after that. Here, we can find the optimum number of cluster.

Advantages of K-means clustring

- Easy to implement.
- With a large number of variables, k-means may be computational faster than hierarchical clustring (if k is small).
- K-means may produce higher clusters than hierarchical clustering.

Disadvatnages of k-means clustring

- Difficult to predict the number of clusters (k-value).
- Initial seeds have a string impact on the final results.

Python Implementation of K-means Clustering Algorithm

Import required libraries

In [1]:
```
import pandas as pd
import matplotlib.pyplot as mp
import seaborn as sb
from sklearn.cluster import KMeans
```

Load the dataset using Pandas

In [2]:
```
dataset = pd.read_csv('../input/mall-customers/Mall_Customers.csv')
```

In [3]:
```
dataset.head() # It will fetch the top 5 tuples from the dataset
```

Out[3]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

As i am working with K-Means Clustering so I need only Annual Income & Spending Money Attributes

In [4]:
```
fields = dataset.iloc[:,[3,4]].values
```

In [5]:
```
print(fields)
```

```
[[ 15  39]
 [ 15  81]
 [ 16   6]
 [ 16  77]
 [ 17  40]
 [ 17  76]
 [ 18   6]
 [ 18  94]
 [ 19   3]
 [ 19  72]
 [ 19  14]
 [ 19  99]
 [ 20  15]
 [ 20  77]
 [ 20  13]
 [ 20  79]
 [ 21  35]
 [ 21  66]
 [ 23  29]
```

Choose Number of Clusters

Finding the optimal number of clusters is an important part of this algorithm. A commonly used method for finding optimal K value is Elbow Method.

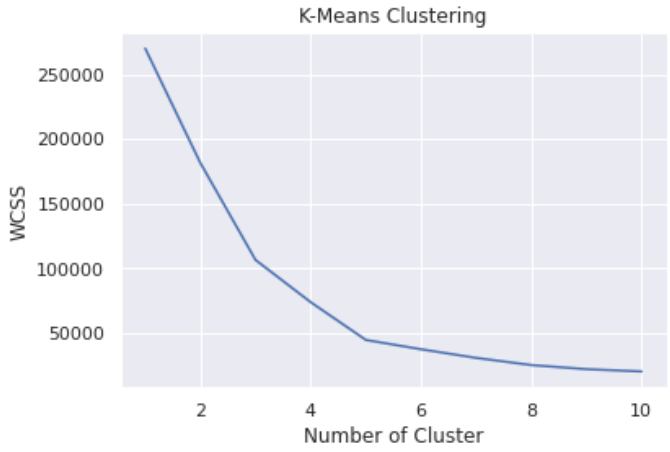
In the Elbow method, we are actually varying the number of clusters (K) from 1 – 10. For each value of K, we are calculating WCSS (Within-Cluster Sum of Square). WCSS is the sum of squared distance between each point and the centroid in a cluster. When we plot the WCSS with the K value, the plot looks like an Elbow. As the number of clusters increases, the WCSS value will start to decrease. WCSS value is largest when K = 1. When we analyze the graph we can see that the graph will rapidly change at a point and thus creating an elbow shape. From this point, the graph starts to move almost parallel to the X-axis. The K value corresponding to this point is the optimal K value or an optimal number of clusters.

In [6]:
```
wcss = [] # empty list
for index in range(1,11): # range 1 to 10 implies that between this range any no. of clusters can be formed
    kmeans = KMeans(n_clusters = index, init = "k-means++", random_state = 2)
    kmeans.fit(fields)

    wcss.append(kmeans.inertia_)
```

**Let's plot the Elbow Method so that i can make out how many clusters will be formed **

In [7]:
```
sb.set()
mp.plot(range(1,11), wcss)
mp.title("K-Means Clustering")
mp.xlabel("Number of Cluster")
mp.ylabel("WCSS")
```

Out[7]: Text(0, 0.5, 'WCSS')



After plotting the Elbow Method , i have make out that 5 clusters will be formed based on the provided datasets

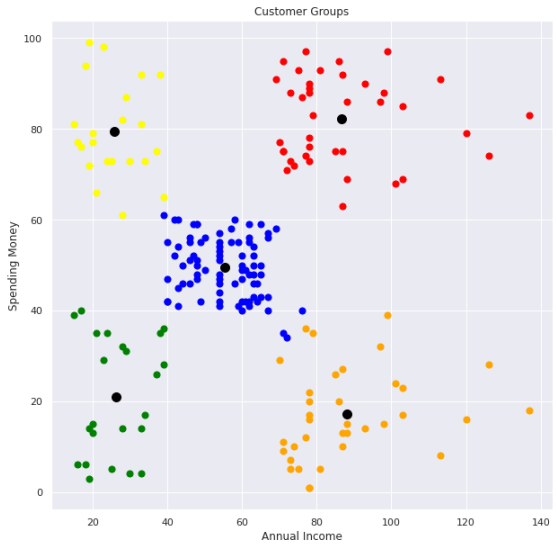
In [8]:
```
kmeans = KMeans(n_clusters=5, init="k-means++",random_state=42)

cluster_values = kmeans.fit_predict(fields)
print(cluster_values)
```

```
[2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2
 3 2 3 2 3 2 0 2 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 4 1 4 0 4 1 4 1 4 0 4 1 4 1 4 1 4 1 4 0 4 1 4 1 4
 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1
 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4]
```

Now, Let's Visualize all the clusters

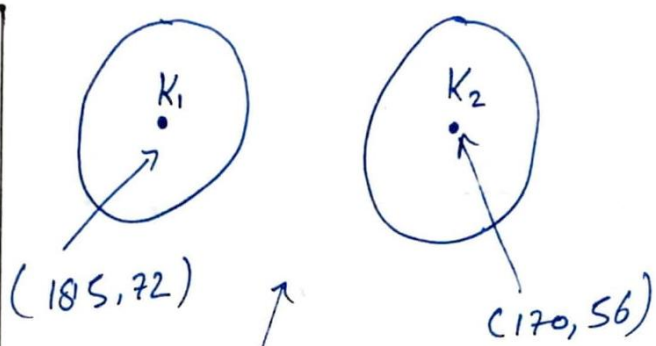
In [9]:
```
mp.figure(figsize=(10,10))
mp.scatter(fields[cluster_values==0,0], fields[cluster_values==0,1], s=50, c='blue', label = "Cluster 1")
mp.scatter(fields[cluster_values==1,0], fields[cluster_values==1,1], s=50, c='orange', label = "Cluster 1")
mp.scatter(fields[cluster_values==2,0], fields[cluster_values==2,1], s=50, c='green', label = "Cluster 1")
mp.scatter(fields[cluster_values==3,0], fields[cluster_values==3,1], s=50, c='yellow', label = "Cluster 1")
mp.scatter(fields[cluster_values==4,0], fields[cluster_values==4,1], s=50, c='red', label = "Cluster 1")

mp.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1], s=100, c="black", label = "Centroids")

mp.title("Customer Groups")
mp.xlabel("Annual Income")
mp.ylabel("Spending Money")
```

Out[9]:
```
Text(0, 0.5, 'Spending Money')
```



K-means Algo-

	x Height	y Weight
1.	185	72
2.	170	52
3.	168	60
4.	179	68
5.	182	72
6.	188	77
7.	180	71
8.	180	70
9.	183	84
10.	180	88.
11.	180	67
12.	177	76



→ These are centroid value

- And other 10 values are observed value

Euclidean Distance =

$$\sqrt{(x_o - x_c)^2 + (y_o - y_c)^2}$$

o = observed value
c = centered value

ED for 3 → $K_1 = \sqrt{(168-185)^2 + (60-72)^2}$

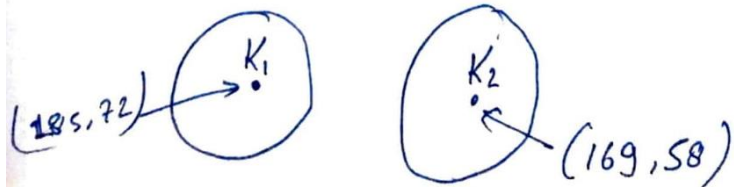
$= 20.80$

→ $K_2 = \sqrt{(168-170)^2 + (60-56)^2}$

$= 4.48$

Next Step → New Centroid Calculation:-

for $K_2 = \left(\frac{170+168}{2}, \frac{60+56}{2}\right) = (169, 58)$



ED for 4 → $K_1 = \sqrt{(179-185)^2 + (68-72)^2}$

$= (6.32)$

→ $K_2 = \sqrt{(179-169)^2 + (68-58)^2}$

$= 14.14$

Now, 4th value gone to the K_1, bcoz in K_1 has less distance comparision to K_2.

Explain thraw Graph :