Supervised and Unsupervised learning :

Supervised learning:

- Supervised learning as the name indicates the presence of a supervisor as a teacher. Basically supervised learning is a learning in which we teach or train the machine using data which is well labeled that means some data is already tagged with the correct answer.
- After that, the machine is provided with a new set of examples(data) so that supervised learning algorithm analyses the training data(set of training examples) and produces a correct outcome from labeled data.
- Supervised learning is the learning of the model where with input variable (say, x) and an output variable (say, Y) and an algorithm to map the input to the output.

Types:

It is Two types:

```
    1. Classification - Here the target variable is Discrete In nature. (E.g. 0,1,2,3,4,5)
    2. Regression - Here the target variable is Continous In nature. (E. g. 0.5,1.23,2.35,3.22)
```

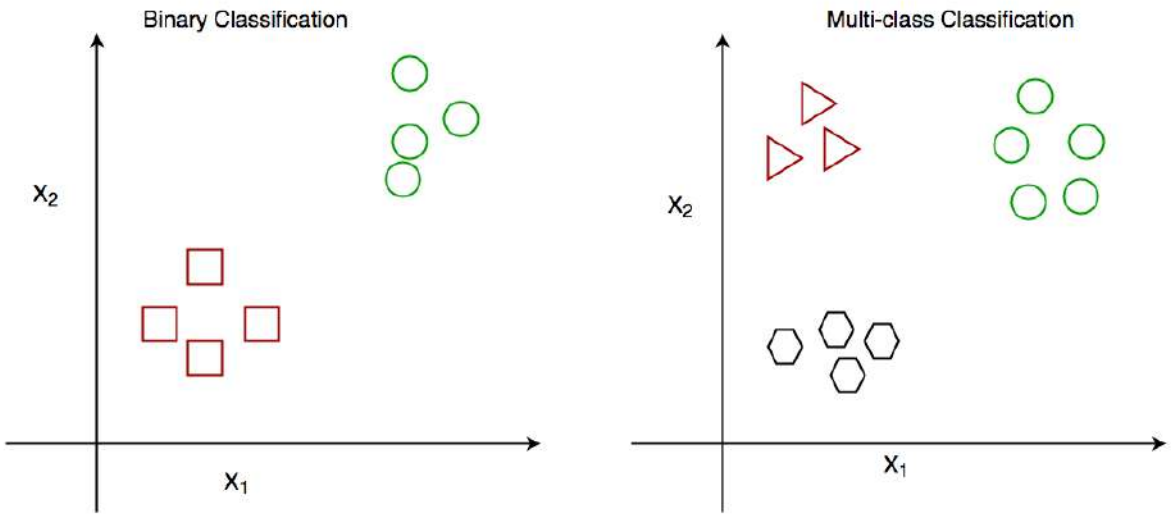
Classificaion:

- A classification problem is when the output variable is a category. it attempts to draw some conclusion from observed values. Given one or more inputs a classification model will try to predict the value of one or more outcomes.
- For example, when filtering emails “spam” or “not spam”, when looking at transaction data, “fraudulent”, or “authorized”. Employee promoted or not.

Classification is of two types:

- Binary Classification : When we have to categorize given data into 2 distinct classes. Example – On the basis of given health conditions of a person, we have to determine whether the person has a certain disease or not. Here target contain only two value.

- Multiclass Classification : The number of classes is more than 2. For Example – On the basis of data about different species of flowers, we have to determine which specie does our observation belong to.



Some Ml Algo Which is used in Classificatio

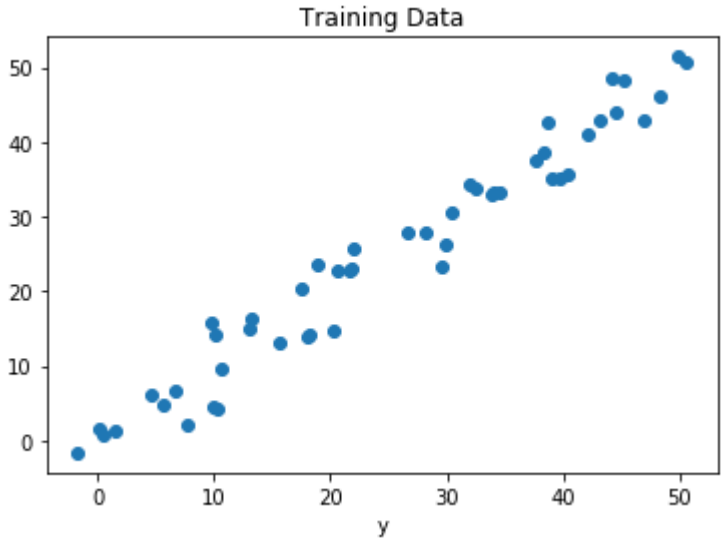
1. LogisticRegression
2. KNeighborsClassifier
3. SVC
4. DecisionTreeClassifier etc

classification metrics :

- Accuracy, Confusion_matrix, Precision, recall, F1-score, ROC, AUC

Regression :

- A regression problem is when the output variable is a real or continuous value, such as “salary” or “weight”. Many different models can be used, the simplest is the linear regression. It tries to fit data with the best hyperplane which goes through the points.
- It can also identify the distribution movement depending on the historical data. Because a regression predictive model predicts a quantity, therefore, the skill of the model must be reported as an error in those predictions



Some MI Algo Which is used in Classificatio:

1. Linear Regression
2. Polynomial regression
3. Ridge regression
4. Lasso regression
5. SVR
6. DecisionTreeRegressor etc

Regression Metrics :

- Mean Squared error (MSE), Root Mean Squared error (RMSE), Root Mean Squared Log error (RMSLE),Mean Absoulute Error (MAE), R-Squared, Adjusted R-Squared,

Unsupervised learning

- Unsupervised learning is where only the input data (say, X) is present and no corresponding output variable is there.
- Unsupervised learning is the training of machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance.
- Here labled data not present means here no target variable.
- Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data.

Source: https://www.geeksforgeeks.org/supervised-unsupervised-learning/ (https://www.geeksforgeeks.org/supervised-unsupervised-learning/)

- Thus the machine has no idea about the features of dogs and cat so we can't categorize it in dogs and cats.
- But it can categorize them according to their similarities, patterns, and differences i.e., we can easily categorize the above picture into two parts.
- First may contain all pics having dogs in it and second part may contain all pics having cats in it. Here you didn't learn anything before, means no training data or examples.
- It allows the model to work on its own to discover patterns and information that was previously undetected. It mainly deals with unlabelled data.

Types:

```
    1. Clustering - discover the inherent groupings in the data
    2. Association - discover rules that describe large portions of your
data
```

Applications of Clustering

- Clustering has a large no. of applications spread across various domains. Some of the most popular applications of clustering are:

```
    * Recommendation engines
    * Market segmentation
    * Social network analysis
    * Search result grouping
    * Medical imaging
    * Image segmentation
    * Anomaly detection
```

In upcoming Notebook i will going to cover all the Clustering Algorithm So stay tunned with me.

What is Simple Linear Regression?

Simple Linear Regression is finding the best relationship between the input variable x (independent variable) and the expected variable y (dependent variable). The linear relationship between these two variables can be represented by a straight line called regression line.

Formula :- $y = b0 + b1x$

What do terms represent?

- y is the response or the target variable
- x is the feature
- $b1$ is the coefficient of x
- $b0$ is the intercept

Estimating ("Learning") Model Coefficients

The coefficients are estimated using the **least-squares criterion**, i.e., the best fit line has to be calculated that minimizes the **sum of squared residuals** (or "sum of squared errors").

Diving into the code

Dividing the code into steps for better understanding:

1. Load the dataset.
2. Visualize the data.
3. Training Simple Linear Regression Model.

Step1 : Load Dataset

In [1]:

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

In [2]:

```
dataset = pd.read_csv('data/Salary_Data.csv')
```

In [3]:

```
dataset.head()
```

Out[3]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

In [17]:

```
dataset.shape
```

Out[17]:

```
(30, 2)
```

Here, 'YearsExperience' is the independent variable and 'Salary' is the dependent variable which will be predicted based on the value of 'YearsExperience'.

In [4]:

```
X = dataset.drop(['Salary'],axis=True)
y = dataset['Salary']
```

In [5]:

```
#Now Split The Data
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=42)
```

In [6]:

```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[6]:

```
((24, 1), (6, 1), (24,), (6,))
```

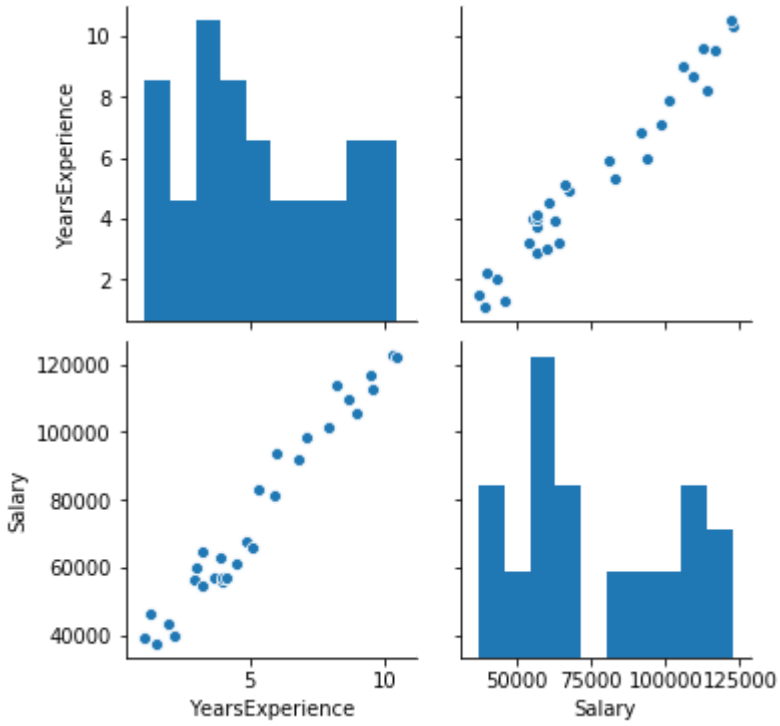
Step 2: Visualize the data.

In [7]:

```
# Visualize Whole Data set
sns.pairplot(dataset)
```

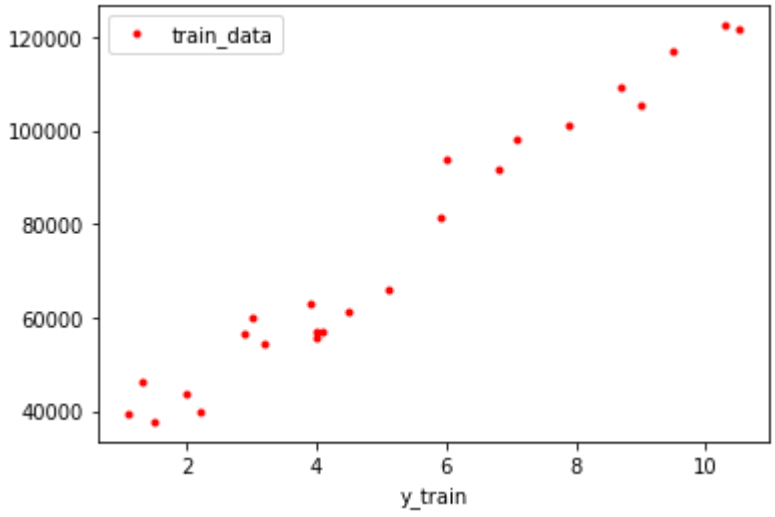
Out[7]:

```
<seaborn.axisgrid.PairGrid at 0x1459f6c6488>
```



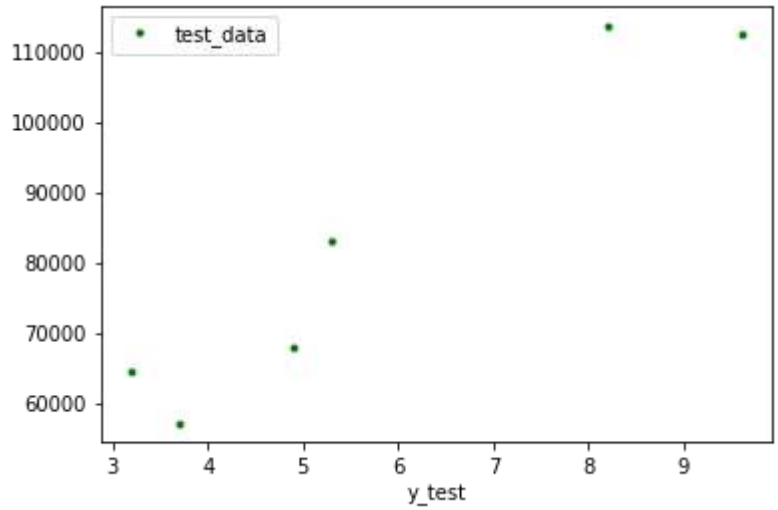
In [8]:

```
# Visualize Only Train data
plt.plot(X_train,y_train,'r.',label='train_data')
plt.xlabel('X_train')
plt.xlabel('y_train')
plt.legend()
plt.show()
```



In [9]:

```
# Visualize Only Test data
plt.plot(X_test,y_test,'g.',label='test_data')
plt.xlabel('X_test')
plt.xlabel('y_test')
plt.legend()
plt.show()
```



In The Above three graph clealr shows that the data is linearly deparable so here we use Linear Regression

Ok, now we hope the dataset is pretty clear by now. We will move to the next step now!

Step 3 :Training Simple Linear Regression Model.

In [10]:

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[10]:

```
LinearRegression()
```

Predicting the Test set results

In [11]:

```
y_pred = regressor.predict(X_test).round(1)
```

In [12]:

```
calculation = pd.DataFrame(np.c_[y_test,y_pred], columns = ["Original Salary","Predict
 Salary"])
calculation
```

Out[12]:

	Original Salary	Predict Salary
0	112635.0	115790.2
1	67938.0	71498.3
2	113812.0	102596.9
3	83088.0	75267.8
4	64445.0	55477.8
5	57189.0	60189.7

Visualising the Training set results

In [16]:

```
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Train set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

Visualising the Training set results

In [14]:

```
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

This is All About Simple Linear Regresion .

What is Multicollinearity

- Multicollinearity occurs when two or more independent variables are highly correlated with one another in a regression model.
- There are two basic kinds of multicollinearity:
    - 1. Structural multicollinearity : This type occurs when we create a model term using other terms.
    - 1. Data multicollinearity : This type of multicollinearity is present in the data itself rather than being an artifact of our model.

What Problems Do Multicollinearity Cause?

- Multicollinearity reduces the precision of the estimate coefficients, which weakens the statistical power of your regression model.
- The coefficient becomes very sensitive to small changes in the model. It’s a disconcerting feeling when slightly different models lead to very different conclusions.

Detecting Multicollinearity

- Here i use two metthod to Detecting Multicollinearity
    - 1. Using OLS - Ordinary Least Squares regression
    - 1. Using VIF - Variable Inflation Factor

Data set where NO Multicollinearity

1. Detecting Multicollinearity using OLS Method

In [1]:

```
import pandas as pd
import statsmodels.api as sm
```

In [2]:

```
df = pd.read_csv('data/Advertising.csv', index_col=0)
```

In [3]:

```
df.head()
```

Out[3]:

	TV	radio	newspaper	sales
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	9.3
4	151.5	41.3	58.5	18.5
5	180.8	10.8	58.4	12.9

In [4]:

```
X = df.drop(['sales'],axis=True)
y = df['sales']
```

- Now See is there any multicolinearity between TV,Radio,Newspaper using OLS
- formula y = b0 + b1(TV)+b2(Radio)+b3(newspaper)
- ##### In Ols we need to compute b0 value but in data set no b0 value so now i am create that b0 in data set. so i add_constant as b0 value

In [5]:

```
X = sm.add_constant(X)
X.head()
```

Out[5]:

	const	TV	radio	newspaper
1	1.0	230.1	37.8	69.2
2	1.0	44.5	39.3	45.1
3	1.0	17.2	45.9	69.3
4	1.0	151.5	41.3	58.5
5	1.0	180.8	10.8	58.4

- Now i am created a const column which measure the b0 value

In [6]:

```
model= sm.OLS(y, X).fit()
```

In [7]:

```
model.summary()
```

Out[7]:

OLS Regression Results

Dep. Variable:	sales	**R-squared:**	0.897
Model:	OLS	**Adj. R-squared:**	0.896
Method:	Least Squares	**F-statistic:**	570.3
Date:	Thu, 17 Sep 2020	**Prob (F-statistic):**	1.58e-96
Time:	12:24:52	**Log-Likelihood:**	-386.18
No. Observations:	200	**AIC:**	780.4
Df Residuals:	196	**BIC:**	793.6
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	2.9389	0.312	9.422	0.000	2.324	3.554
TV	0.0458	0.001	32.809	0.000	0.043	0.049
radio	0.1885	0.009	21.893	0.000	0.172	0.206
newspaper	-0.0010	0.006	-0.177	0.860	-0.013	0.011

Omnibus:	60.414	**Durbin-Watson:**	2.084
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	151.241
Skew:	-1.327	**Prob(JB):**	1.44e-33
Kurtosis:	6.332	**Cond. No.**	454.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

- In model Summary we look only 4 things

- 1. coef 2. std err 3. p value 4. r-squared
  - mainly which have high std err these are Multicollinearity each other
  - As you see here std-err is low so here no Multicollinearity

2. Detecting Multicollinearity using VIF Method

In [8]:

```
# Import library for VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

    return(vif)
```

In [9]:

```
X = df.iloc[:,:-1]
calc_vif(X)
```

Out[9]:

	variables	VIF
0	TV	2.486772
1	radio	3.285462
2	newspaper	3.055245

- VIF starts at 1 and has no upper limit
- VIF = 1, no correlation between the independent variable and the other variables
- VIF exceeding 5 or 10 indicates high multicollinearity between this independent variable and the others
- Here all variable has low VIF Value so no Multicollinearity
- In the above i am use OLS and VIF method to find any Multicollinearity between Variables as a result i found there is no 1. Multicollinearity between any variables

Data set where Multicollinearity exist and How to Overcome

- Here also i use this two method to find Multicollinearity
- but in project you can use any of these which is suitable for you .

1. Detecting Multicollinearity using OLS Method

In [10]:

```
df_salary = pd.read_csv('data/Salary_Data.csv')
df_salary.head()
```

Out[10]:

	YearsExperience	Age	Salary
0	1.1	21.0	39343
1	1.3	21.5	46205
2	1.5	21.7	37731
3	2.0	22.0	43525
4	2.2	22.2	39891

In [11]:

```
X = df_salary.drop(['Salary'],axis=True)
y = df_salary['Salary']
```

In [12]:

```
X = sm.add_constant(X)
X.head()
```

Out[12]:

	const	YearsExperience	Age
0	1.0	1.1	21.0
1	1.0	1.3	21.5
2	1.0	1.5	21.7
3	1.0	2.0	22.0
4	1.0	2.2	22.2

In [13]:

```
model= sm.OLS(y, X).fit()
```

In [14]:

```
model.summary()
```

Out[14]:

OLS Regression Results

Dep. Variable:	Salary	**R-squared:**	0.960
Model:	OLS	**Adj. R-squared:**	0.957
Method:	Least Squares	**F-statistic:**	323.9
Date:	Thu, 17 Sep 2020	**Prob (F-statistic):**	1.35e-19
Time:	12:24:52	**Log-Likelihood:**	-300.35
No. Observations:	30	**AIC:**	606.7
Df Residuals:	27	**BIC:**	610.9
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-6661.9872	2.28e+04	-0.292	0.773	-5.35e+04	4.02e+04
YearsExperience	6153.3533	2337.092	2.633	0.014	1358.037	1.09e+04
Age	1836.0136	1285.034	1.429	0.165	-800.659	4472.686

Omnibus:	2.695	**Durbin-Watson:**	1.711
Prob(Omnibus):	0.260	**Jarque-Bera (JB):**	1.975
Skew:	0.456	**Prob(JB):**	0.372
Kurtosis:	2.135	**Cond. No.**	626.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

- As You see YearsExperience and Age Has High std err so here Multicollinearity occure

- so to fix it we have to remove one variable which has high p value
  - Here Age Has High p value so just drop the age feature

2. Detecting Multicollinearity using VIF Method

In [15]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

    return(vif)
```

In [16]:

```
X = df_salary.iloc[:,:-1]
calc_vif(X)
```

Out[16]:

	variables	VIF
0	YearsExperience	11.24047
1	Age	11.24047

- We can see here that the 'Age' and ' YearsExperience' have a high VIF value, meaning they can be predicted by other independent variables in the dataset.

Fixing Multicollinearity

- 1. Dropping one of the correlated features will help in bringing down the multicollinearity between correlated features.
- 1. combine the correlated variables into one and drop the others. This will reduce the multicollinearity:

Another Method To Know multicollinearity

- Use Correlation matrix

In [17]:

```
df_salary.corr()
```

Out[17]:

	YearsExperience	Age	Salary
YearsExperience	1.000000	0.987258	0.978242
Age	0.987258	1.000000	0.974530
Salary	0.978242	0.974530	1.000000

In [18]:

```
df.corr()
```

Out[18]:

	TV	radio	newspaper	sales
TV	1.000000	0.054809	0.056648	0.782224
radio	0.054809	1.000000	0.354104	0.576223
newspaper	0.056648	0.354104	1.000000	0.228299
sales	0.782224	0.576223	0.228299	1.000000

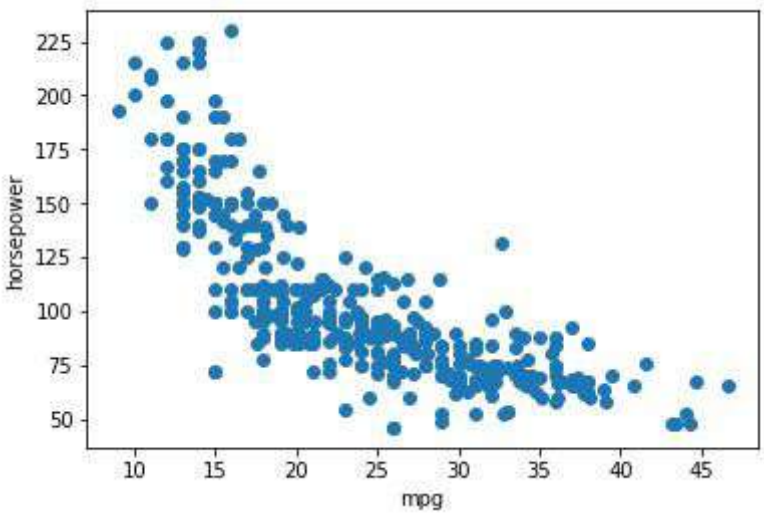
- Here You Can Use Seaborn To plot this matrix

--- This All About multicollinearity If It Helpful to you Please Like Me ---

What is Polynomial Linear Regression?

- In Simple Linear Regression, we use a straight line with formula Y = MX + C.
- It was used to best fit the line to our dataset.
- But in some datasets, it is hard to fit a straight line. Therefore we use a polynomial function of a curve to fit all the data points.
- The formula of the polynomial linear regression is almost similar to that of Simple Linear Regression.

$$y = a + bx + cx^2 + \ldots + nx^n + \ldots$$



- if the dataset look like this type then we are doesnot use Linear Regression

- In this case we are use Polynomial regresion or other Type of regression techinque like (Decision tree,naive Bayes etc)

Implementation of the polynomial linear regression in Python

In [1]:

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

In [2]:

```
dataset = pd.read_csv('data/Position_Salaries.csv')
```

In [3]:

```
dataset.head(10)
```

Out[3]:

	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 [4]:

```
dataset = dataset.drop(['Position'],axis=True)
```

In [5]:

```
dataset.info()
```

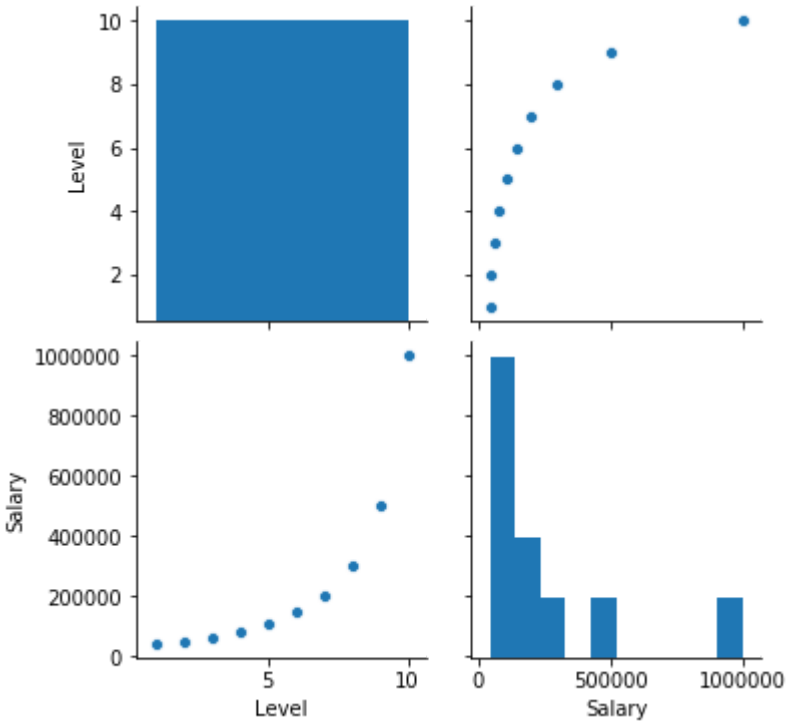
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Level   10 non-null     int64
 1   Salary  10 non-null     int64
dtypes: int64(2)
memory usage: 288.0 bytes
```

In [6]:

```
sns.pairplot(dataset)
```

Out[6]:

```
<seaborn.axisgrid.PairGrid at 0x2870ff66c08>
```



As see this plot is not linearly separable so here we use polynomial regression

In [7]:

```
X = dataset.drop(['Salary'],axis=True)
y = dataset['Salary']
```

In [8]:

```
#Now Split The Data
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=42)
```

In [9]:

```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[9]:

```
((8, 1), (2, 1), (8,), (2,))
```

Training the Linear Regression model

In [10]:

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
```

Out[10]:

```
LinearRegression()
```

Accuracy For Linear Regression

In [11]:

```
print("Training Accuracy :", lin_reg.score(X_train, y_train))
print("Testing Accuracy :", lin_reg.score(X_test, y_test))
```

```
Training Accuracy : 0.6366049276570868
Testing Accuracy : 0.8451346684575975
```

Training the Polynomial Regression model

In [12]:

```
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(X_train)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y_train)
X_poly_test = poly_reg.transform(X_test)
```

Accuracy For Using Polynomial Regression

In [13]:

```
print("Training Accuracy :", lin_reg_2.score(X_poly, y_train))
print("Testing Accuracy :", lin_reg_2.score(X_poly_test, y_test))
```

```
Training Accuracy : 0.9995857211026754
Testing Accuracy : 0.9714666803841844
```

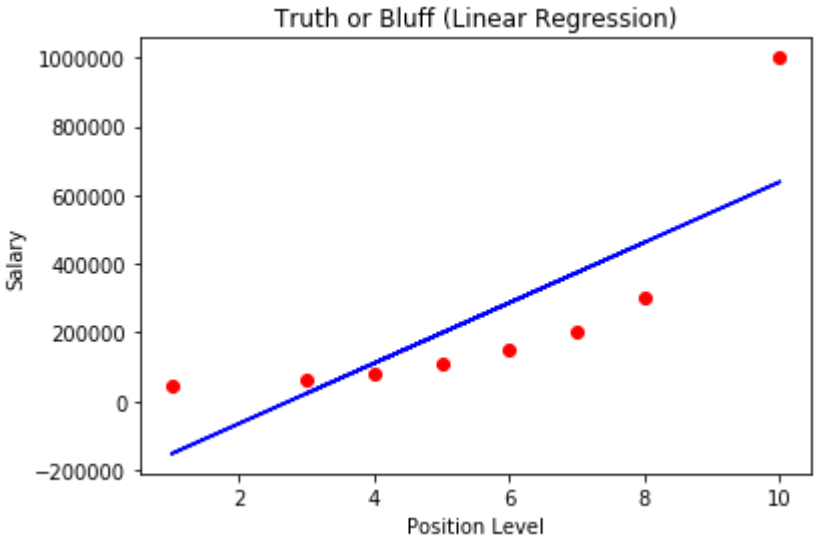
- See in above that when i apply only Linear Regression On this data accuracy is not good, but When i apply polynomial regression then the accuracy is very high.

- So i hope Now you understood when we apply Linear Regression and when we apply Polynomial Regression

Graph For When We Use Linear Regression

In [14]:

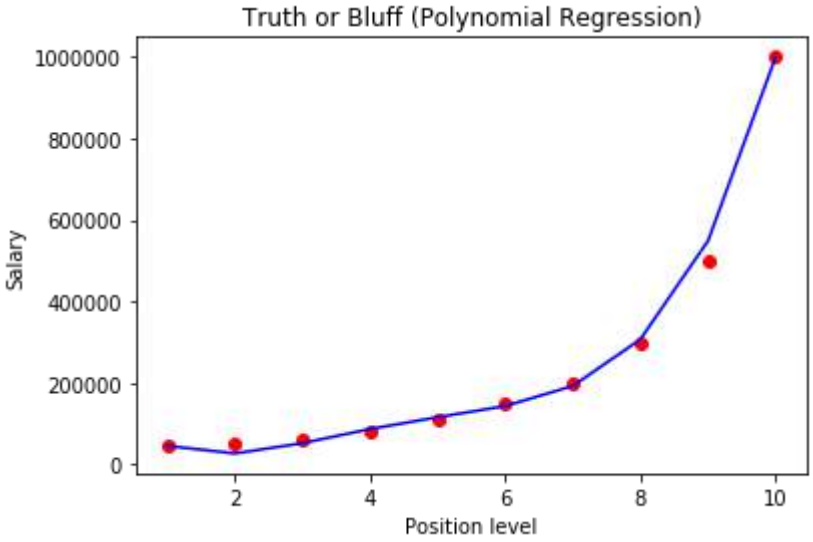
```
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, lin_reg.predict(X_train), color = 'blue')
plt.title('Truth or Bluff (Linear Regression)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```



Graph For When We Use Polynomial Regression

In [15]:

```
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg_2.predict(poly_reg.fit_transform(X)), color = 'blue')
plt.title('Truth or Bluff (Polynomial Regression)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```



** Difference Between Simple And Multiple Linear Regression

- Simple linear regression has only one x and one y variable.
- Multiple linear regression has one y and two or more x variables.
- For instance, when we predict rent based on square feet alone that is simple linear regression.
- When we predict rent based on square feet and age of the building that is an example of multiple linear regression.

An extension of simple linear regression

In simple linear regression there is a one-to-one relationship between the input variable and the output variable. But in multiple linear regression, as the name implies there is a many-to-one relationship, instead of just using one input variable, you use several.

Multiple Linear Regression

Till now, we have created the model based on only one feature. Now, we'll include multiple features and create a model to see the relationship between those features and the label column. This is called **Multiple Linear Regression**.

y = b0 + b1x1+ b2x2... + bnxn

What do terms represent?

- y is the response or the target variable
- x1,x2,x3...xn are the feature as it is multiple
- b1,b2...bn are the coefficient of x1,x2,..xn respectively
- b0 is the intercept

Each x represents a different feature, and each feature has its own coefficient

Implementation

Step1: Import data

In [1]:

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

In [2]:

```
dataset = pd.read_csv('data/50_Startups.csv')
```

In [3]:

```
dataset.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

- See here more than one feature so it is multiplt linear regression
- Here Profit is our Target Feature

In [4]:

```
dataset.shape
```

Out[4]:

```
(50, 5)
```

Step2: Visuallize The Data

In [5]:

```
#sns.pairplot(dataset)
```

In [6]:

```
dataset = dataset.drop('State',axis=True)
```

- Here i simply drop the State feature.
- in next some days i will show how to deal with categorical feature.

In [7]:

```
dataset.head()
```

Out[7]:

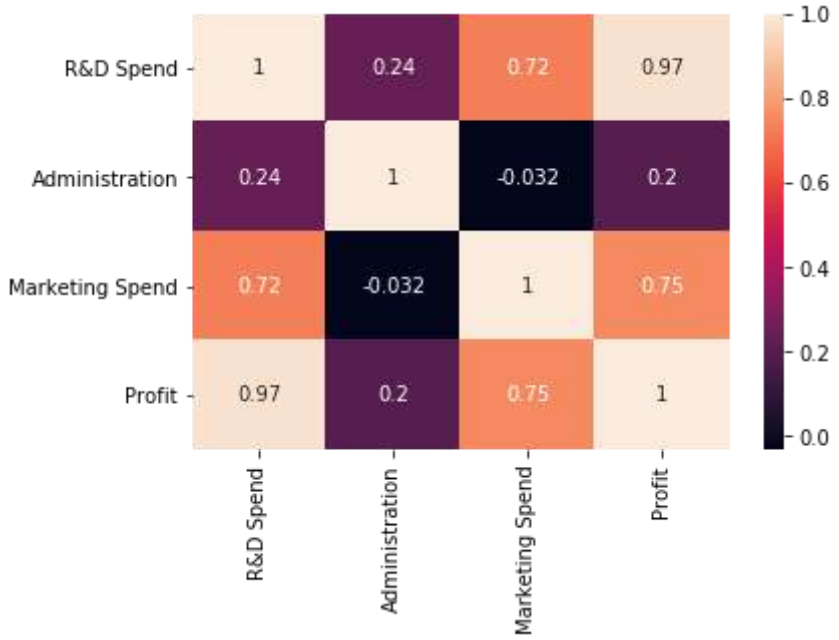
	R&D Spend	Administration	Marketing Spend	Profit
0	165349.20	136897.80	471784.10	192261.83
1	162597.70	151377.59	443898.53	191792.06
2	153441.51	101145.55	407934.54	191050.39
3	144372.41	118671.85	383199.62	182901.99
4	142107.34	91391.77	366168.42	166187.94

In [8]:

```
corr = dataset.corr()
sns.heatmap(corr,annot=True)
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2e6d767ad08>
```



Evaluate The Model

Scalling The Data

In [9]:

```
X = dataset.drop('Profit',axis=True)
y = dataset['Profit']
```

In [10]:

```
X.head() #before standardized data
```

Out[10]:

	R&D Spend	Administration	Marketing Spend
0	165349.20	136897.80	471784.10
1	162597.70	151377.59	443898.53
2	153441.51	101145.55	407934.54
3	144372.41	118671.85	383199.62
4	142107.34	91391.77	366168.42

In [11]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
```

- As here all data are in very big range so we need to take all the data in same range.
- So here i use StandardScaler which take all data in same range
- here it use z score to standardized data
- Z- score formula

$$Z = x-\mu /\sigma$$

- Z = standard score
- x = observed value
- μ = mean of the sample
- σ = standard deviation of the sample
- Some Of the Algorithm Like Tree Base Algorithm not require scalling

In [12]:

```
X #after standardized/scalling data
```

Out[12]:

```
array([[ 2.01641149e+00,  5.60752915e-01,  2.15394309e+00],
       [ 1.95586034e+00,  1.08280658e+00,  1.92360040e+00],
       [ 1.75436374e+00, -7.28257028e-01,  1.62652767e+00],
       [ 1.55478369e+00, -9.63646307e-02,  1.42221024e+00],
       [ 1.50493720e+00, -1.07991935e+00,  1.28152771e+00],
       [ 1.27980001e+00, -7.76239071e-01,  1.25421046e+00],
       [ 1.34006641e+00,  9.32147208e-01, -6.88149930e-01],
       [ 1.24505666e+00,  8.71980011e-01,  9.32185978e-01],
       [ 1.03036886e+00,  9.86952101e-01,  8.30886909e-01],
       [ 1.09181921e+00, -4.56640246e-01,  7.76107440e-01],
       [ 6.20398248e-01, -3.87599089e-01,  1.49807267e-01],
       [ 5.93085418e-01, -1.06553960e+00,  3.19833623e-01],
       [ 4.43259872e-01,  2.15449064e-01,  3.20617441e-01],
       [ 4.02077603e-01,  5.10178953e-01,  3.43956788e-01],
       [ 1.01718075e+00,  1.26919939e+00,  3.75742273e-01],
       [ 8.97913123e-01,  4.58678535e-02,  4.19218702e-01],
       [ 9.44411957e-02,  9.11841968e-03,  4.40446224e-01],
       [ 4.60720127e-01,  8.55666318e-01,  5.91016724e-01],
       [ 3.96724938e-01, -2.58465367e-01,  6.92992062e-01],
       [ 2.79441650e-01,  1.15983657e+00, -1.74312698e+00],
       [ 5.57260867e-02, -2.69587651e-01,  7.23925995e-01],
       [ 1.02723599e-01,  1.16918609e+00,  7.32787791e-01],
       [ 6.00657792e-03,  5.18495648e-02,  7.62375876e-01],
       [-1.36200724e-01, -5.62211268e-01,  7.74348908e-01],
       [ 7.31146008e-02, -7.95469167e-01, -5.81939297e-01],
       [-1.99311688e-01,  6.56489139e-01, -6.03516725e-01],
       [ 3.53702028e-02,  8.21717916e-01, -6.35835495e-01],
       [-3.55189938e-02,  2.35068543e-01,  1.17427116e+00],
       [-1.68792717e-01,  2.21014050e+00, -7.67189437e-01],
       [-1.78608540e-01,  1.14245677e+00, -8.58133663e-01],
       [-2.58074369e-01, -2.05628659e-01, -9.90357166e-01],
       [-2.76958231e-01,  1.13055391e+00, -1.01441945e+00],
       [-2.26948675e-01,  2.83923813e-01, -1.36244978e+00],
       [-4.01128925e-01, -6.59324033e-01,  2.98172434e-02],
       [-6.00682122e-01,  1.31053525e+00, -1.87861793e-03],
       [-6.09749941e-01, -1.30865753e+00, -4.54931587e-02],
       [-9.91570153e-01,  2.05924691e-01, -8.17625734e-02],
       [-6.52532310e-01, -2.52599402e+00, -1.15608256e-01],
       [-1.17717755e+00, -1.99727037e+00, -2.12784866e-01],
       [-7.73820359e-01, -1.38312156e+00, -2.97583276e-01],
       [-9.89577015e-01, -1.00900218e-01, -3.15785883e-01],
       [-1.00853372e+00, -1.32079581e+00, -3.84552407e-01],
       [-1.10210556e+00, -9.06937535e-01, -5.20595959e-01],
       [-1.28113364e+00,  2.17681524e-01, -1.44960468e+00],
       [-1.13430539e+00,  1.20641936e+00, -1.50907418e+00],
       [-1.60035036e+00,  1.01253936e-01, -1.72739998e+00],
       [-1.59341322e+00, -1.99321741e-01,  7.11122474e-01],
       [-1.62236202e+00,  5.07721876e-01, -1.74312698e+00],
       [-1.61043334e+00, -2.50940884e+00, -1.74312698e+00],
       [-1.62236202e+00, -1.57225506e-01, -1.36998473e+00]])
```

In [13]:

```
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
= 0)
```

In [14]:

```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[14]:

```
((40, 3), (10, 3), (40,), (10,))
```

Build Model

In [15]:

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[15]:

```
LinearRegression()
```

In [16]:

```
y_pred = regressor.predict(X_test).round(1)
```

In [17]:

```
calculation = pd.DataFrame(np.c_[y_test,y_pred], columns = ["Original Salary","Predict
 Salary"])
calculation.head(5)
```

Out[17]:

	Original Salary	Predict Salary
0	103282.38	103901.9
1	144259.40	132763.1
2	146121.95	133567.9
3	77798.83	72911.8
4	191050.39	179627.9

In [18]:

```
print("Training Accuracy :", regressor.score(X_train, y_train))
print("Testing Accuracy :", regressor.score(X_test, y_test))
```

```
Training Accuracy : 0.9499572530324031
Testing Accuracy : 0.9393955917820571
```

In [19]:

```
regressor.intercept_
```

Out[19]:

```
111297.71256204927
```

In [20]:

```
regressor.coef_
```

Out[20]:

```
array([35391.2501208 ,   815.21987542,  4202.06618916])
```

Test The Model

In [21]:

```
feature = [165349.20,136897.80,471784.10]
scale_feature = sc.transform([feature])
scale_feature
```

Out[21]:

```
array([[2.01641149, 0.56075291, 2.15394309]])
```

In [22]:

```
y_pred_test = regressor.predict(scale_feature)
y_pred_test  #By Using Sklearn Library
```

Out[22]:

```
array([192169.18440985])
```

In [23]:

```
# Here I use b1x1+b2x2+b3x3+b0 BY MANUAL
35391.2501208*2.01641149+815.21987542*0.56075291+4202.06618916*2.15394309+ 111297.71256
204927
```

Out[23]:

```
192169.1843003897
```

- Now above you see manual and automatic prediction on the same data in this way linear regression predict the data

Regularization

- it is an extremely important concept in machine learning. It’s a way to prevent overfitting, and thus, improve the likely generalization performance of a model by reducing the complexity of the final estimated model.
- In regularization, what we do is normally we keep the same number of features, but reduce the magnitude of the coefficients.
- Main Objective of regularization is to scale down the coeficient Value.

Ex:- Real Value

0.9+1.2x1+20x2+39x3

- as you all see there are 1.2,20,39 are three coefficient. we easily say which have high coefficient this effect highly our target variable as x3 has high coefficient it has higly corelate with target variable.
- so but this may cause overfit in real life problem. so we need to scale down these coefficient value to decrease the complexity of model

After Scale down(By appling some regularization techinque)

0.9+0.7x1+2x2+5x3

- see here i apply regularization and scaled down the value of coefficient. as this is low value so here the model give good accuracy. so we always tries to scaled down the coeficient value.

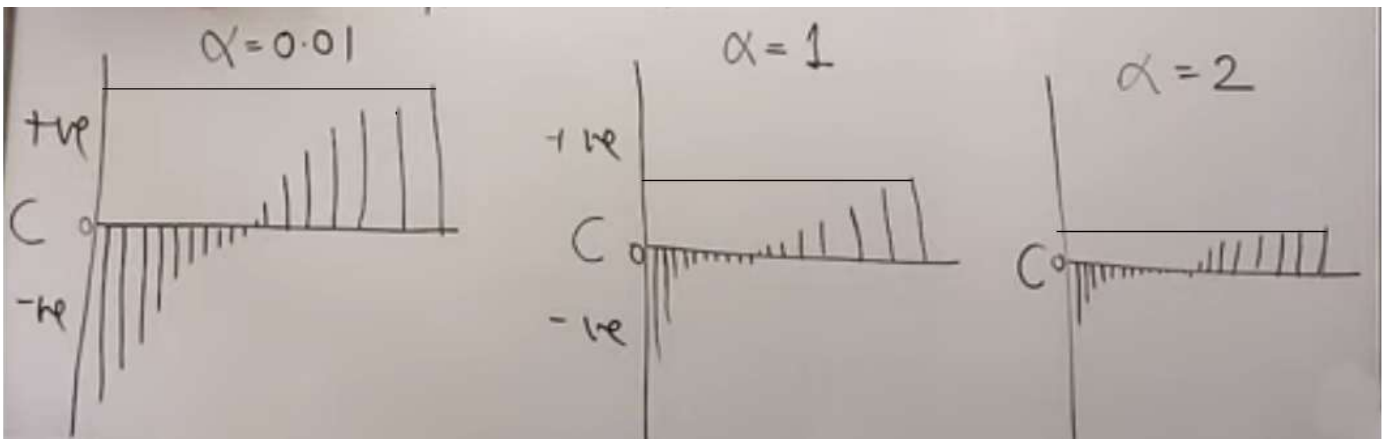
- Now lets dicuss how we scaled down these value.
- In regularization two type of method

```
1. Ridge Regression
2. Lasso Regression
```

Ridge Regression :

Formula : Ridge = Loss + α $||W||^2$

- Whrer Loss = Difference between predict and actual value (Or Cost Function)
- W = slope
- α = constnt

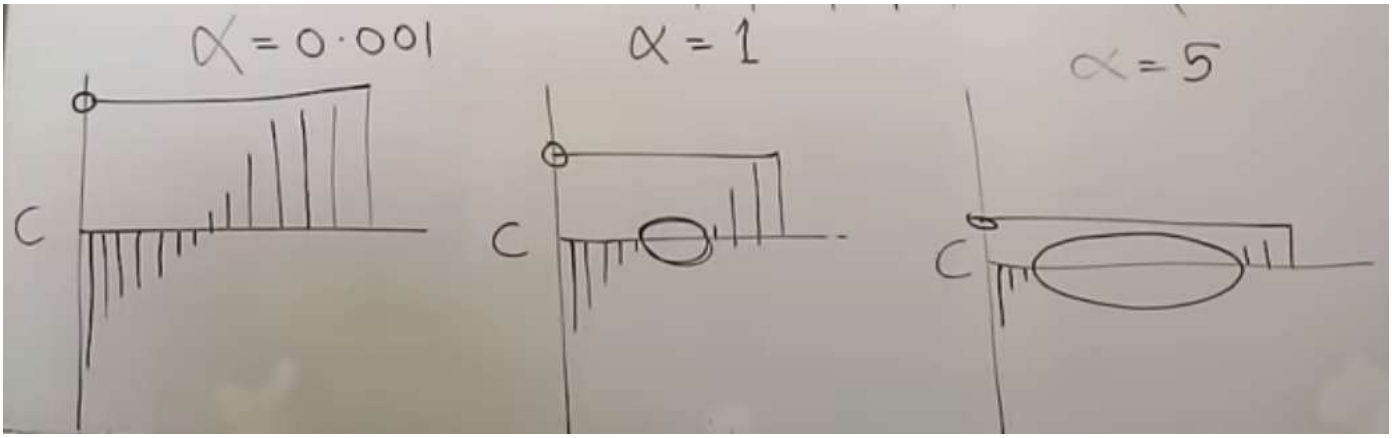


- See in this picture how coeficient value reduced by giving the differnet α Value.

Lasso Regression :

Formula: Lasso = Loss + α $||W||$

- Whrer Loss = Difference between predict and actual value (Or Cost Function)
- W = slope
- α = constnt
- In this we can consider only absolute value.



- As you see in above picture it can reduce the magnitude as well as it reduce the feature. so Lasso is also used for feature selection.

Note :

1. Regularization is mainly used for Scaled down the coeficient value so that overfit not happen in model.
2. Regularization reduce complexity of model
3. in Ridge Regression magnitude of coefficient is almost zero , but in Lasso magnitude of coefficient is exaclty zero.
4. we can find appropriate α value by doing Cross validation.
5. α value is always 0 to any +ve number.
6. Lasso is used for feature selection.
7. Ridge Also called as L2 Regularization, and Lasso Is L1 Regularization.

Now Implement In Python

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston

import warnings
warnings.simplefilter('ignore')
```

In [2]:

```
df=load_boston()
dataset = pd.DataFrame(df.data)
dataset.columns=df.feature_names
dataset["Price"]=df.target
dataset.head()
```

Out[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	

In [3]:

```
X=dataset.iloc[:,:-1] ## independent features
y=dataset.iloc[:,-1] ## dependent features
```

Linear Regression

- First i am going to apply linear regression and find out the MSE Value. Then i apply both Lasso And Rigde to comapre which has less MSE.

In [4]:

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

lin_regressor=LinearRegression()
mse=cross_val_score(lin_regressor,X,y,scoring='neg_mean_squared_error',cv=5)
lin_regressor.fit(X,y)
mean_mse=np.mean(mse)
print(mean_mse)
```

```
-37.13180746769922
```

- here i take neg_mean_squared_error instead of mean_squared_error, there no differnece between these

Ridge Regression

In [5]:

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

ridge=Ridge()

#here i do CV for find alpha value
parameters={'alpha':[1e-15,1e-10,1e-8,1e-3,1e-2,1,5,10,20,30,35,40,45,50,55,100]}
ridge_regressor=GridSearchCV(ridge,parameters,scoring='neg_mean_squared_error',cv=5)
ridge_regressor.fit(X,y)

print(ridge_regressor.best_params_)
print(ridge_regressor.best_score_)
```

```
{'alpha': 100}
-29.90570194754033
```

See here Rigde Regression give -29 Mse value which is lower than Linear Regression

Lasso Regression

In [6]:

```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
lasso=Lasso()

#here i do CV for find alpha value
parameters={'alpha':[1e-15,1e-10,1e-8,1e-3,1e-2,1,5,10,20,30,35,40,45,50,55,100]}
lasso_regressor=GridSearchCV(lasso,parameters,scoring='neg_mean_squared_error',cv=5)

lasso_regressor.fit(X,y)
print(lasso_regressor.best_params_)
print(lasso_regressor.best_score_)
```

```
{'alpha': 1}
-35.531580220694856
```

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=0.3, random_state=0)
```

In [8]:

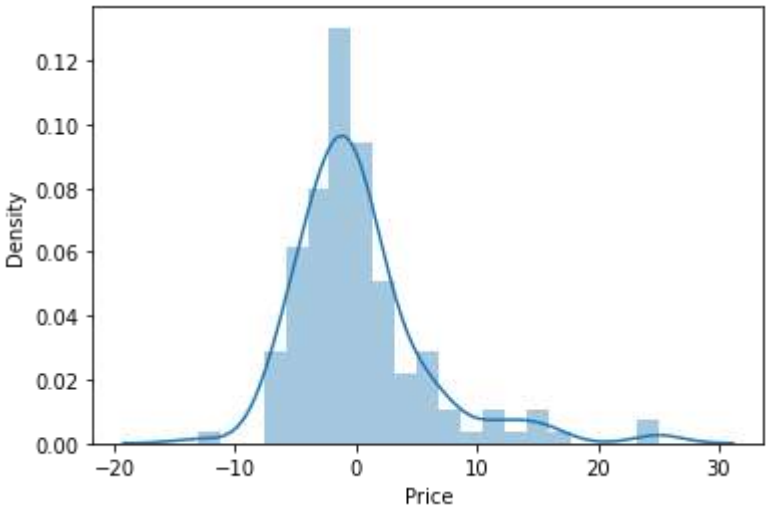
```
prediction_lasso=lasso_regressor.predict(X_test)
prediction_ridge=ridge_regressor.predict(X_test)
```

In [9]:

```
#graph for lasso
sns.distplot(y_test-prediction_lasso)
```

Out[9]:

```
<AxesSubplot:xlabel='Price', ylabel='Density'>
```

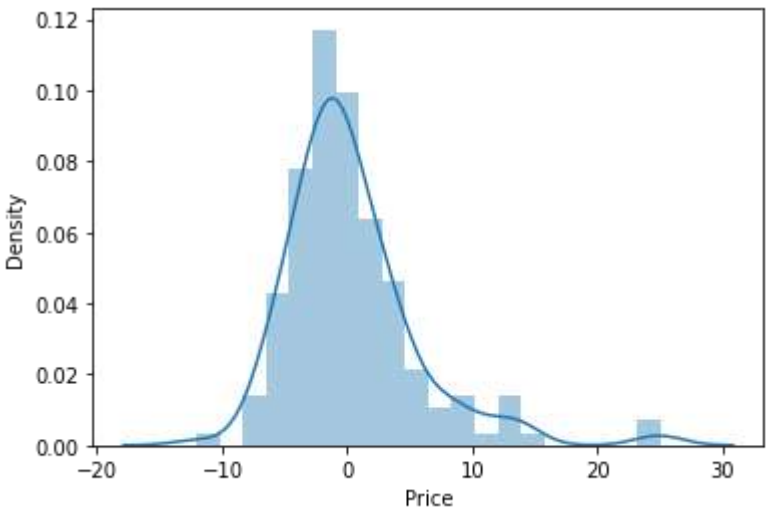


In [10]:

```
# graph for ridge
sns.distplot(y_test-prediction_ridge)
```

Out[10]:

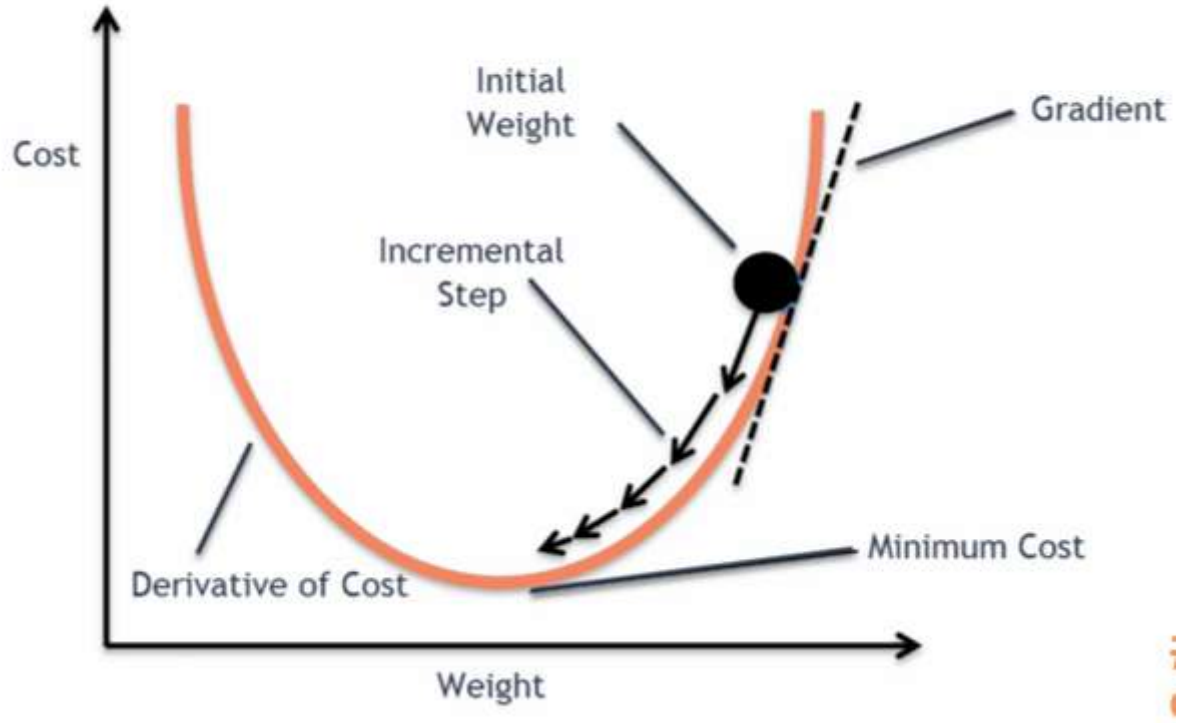
```
<AxesSubplot:xlabel='Price', ylabel='Density'>
```



- See ridge gives almost perfect gaussian curve than lasso, as ridge has samll mse value. in this way we can implement all ridge and lasso.

Gardient Descent

- Gradient Descent is an optimization algorithm used for minimizing the cost function in various machine learning algorithms. It is basically used for updating the parameters of the learning model.
- Lets i give you a example : Suppose you are lost in the mountains in a dense fog; you can only feel the slope of the ground below your feet. A good strategy to get to the bottom of the valley quickly is to go downhill in the direction of the steepest slope. This is exactly what Gradient Descent does: it measures the local gradient of the error function with regards to the parameter vector θ, and it goes in the direction of descending gradient. Once the gra- dient is zero, you have reached a minimum!



Steps of gradient descent

Following the mountain example, In each position, the agent only knows two things:

- the gradient (for that position, or parameters) and the width of the step to take (learning rate).
- With that information, the current value of each parameter is updated. With the new parameter values, the gradient is re-calculated and the process is repeated until reach convergence or local minima.
- Let's revise how the gradient descent algorithm works at each step:

Repeat until hit convergence:

1. Given the gradient, calculate the change in the parameters with the learning rate.
2. Re-calculate the new gradient with the new value of the parameter.
3. Repeat step 1.

Here is the formula of gradient descent algorithm:

$$
\begin{aligned}
&Repeat\ \{ \\
&\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\
&\}
\end{aligned}
$$

Convergence is a name given to the situation where the loss function does not improve significantly, and we are stuck in a point near to the minima.

Types of gradient Descent:

1. Batch Gradient Descent: This is a type of gradient descent which processes all the training examples for each iteration of gradient descent. But if the number of training examples is large, then batch gradient descent is computationally very expensive. Hence if the number of training examples is large, then batch gradient descent is not preferred. Instead, we prefer to use stochastic gradient descent or mini-batch gradient descent.

2. Stochastic Gradient Descent: This is a type of gradient descent which processes 1 training example per iteration. Hence, the parameters are being updated even after one iteration in which only a single example has been processed. Hence this is quite faster than batch gradient descent. But again, when the number of training examples is large, even then it processes only one example which can be additional overhead for the system as the number of iterations will be quite large.

3. Mini Batch gradient descent: This is a type of gradient descent which works faster than both batch gradient descent and stochastic gradient descent. Here b examples where b<m are processed per iteration. So even if the number of training examples is large, it is processed in batches of b training examples in one go. Thus, it works for larger training examples and that too with lesser number of iterations.

Convergence trends in different variants of Gradient Descents:

In case of Batch Gradient Descent, the algorithm follows a straight path towards the minimum.If the cost function is convex, then it converges to a global minimum and if the cost function is not convex, then it converges to a local minimum. Here the learning rate is typically held constant.

In case of stochastic gradient Descent and mini-batch gradient descent, the algorithm does not converge but keeps on fluctuating around the global minimum. Therefore in order to make it converge, we have to slowly change the learning rate. However the convergence of Stochastic gradient descent is much noisier as in one iteration, it processes only one training example.

Here I show Gradient Descent In Linear Regression

In linear regression, the model targets to get the best-fit regression line to predict the value of y based on the given input value (x). While training the model, the model calculates the cost function which measures the Root Mean Squared error between the predicted value (pred) and true value (y). The model targets to minimize the cost function.

To minimize the cost function, the model needs to have the best value of pred and y. Initially model selects m and c values randomly and then itertively update these value in order to minimize the cost function untill it reaches the minimum.

By the time model achieves the minimum cost function, it will have the best pred and y values. Using these finally updated values of pred and y in the hypothesis equation of linear equation, model predicts the value of y in the best manner it can.

##

$$J = \frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)^2$$

$$minimize\frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)^2$$

Gradient Descent Algorithm For Linear Regression

Cost Function

$$J(\Theta_0, \Theta_1) = \frac{1}{2m}\sum_{i=1}^{m}[h_\Theta(x_i) - y_i]^2$$

Predicted Value — True Value

Gradient Descent

$$\Theta_j = \Theta_j - \alpha\frac{\partial}{\partial\Theta_j}J(\Theta_0, \Theta_1)$$

Learning Rate

Now,

$$\frac{\partial}{\partial\Theta}J_\Theta = \frac{\partial}{\partial\Theta}\frac{1}{2m}\sum_{i=1}^{m}[h_\Theta(x_i) - y]^2$$

$$= \frac{1}{m}\sum_{i=1}^{m}(h_\Theta(x_i) - y)\frac{\partial}{\partial\Theta_j}(\Theta x_i - y)$$

$$= \frac{1}{m}(h_\Theta(x_i) - y)x_i$$

Therefore,

$$\Theta_j := \Theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}[(h_\Theta(x_i) - y)x_i]$$

Where

-> θj : Weights of the hypothesis.

-> hθ(xi) : predicted y value for ith input.

-> j : Feature index number (can be 0, 1, 2, ..., n).

-> i : Learning Rate of Gradient Descent.

Now Time To Implement

In [1]:

```
# Making the imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (10,5)
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
from math import sqrt
```

In [2]:

```
# Preprocessing Input data
cols = ['a','b']
data = pd.read_csv('data.csv',header=None,names=cols)
```

In [3]:

```
data.head()
```

Out[3]:

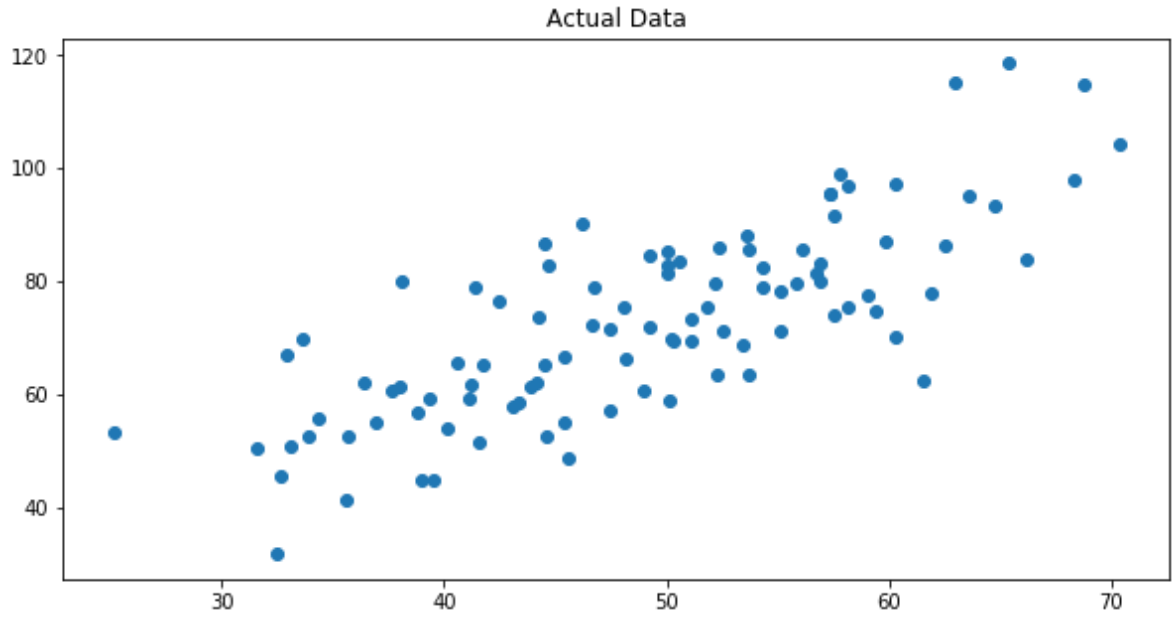
	a	b
0	32.502345	31.707006
1	53.426804	68.777596
2	61.530358	62.562382
3	47.475640	71.546632
4	59.813208	87.230925

In [4]:

```
x = data.drop('b',axis=1)
y = data['b']
```

In [5]:

```
plt.scatter(x, y)
plt.title('Actual Data')
plt.show()
```



In [6]:

```
##Simple Linear Regression

from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x,y)
```

Out[6]:

```
LinearRegression()
```

In [7]:

```
y_pred_lr= lr.predict(x)
```

In [8]:

```
print("Intercept is",lr.intercept_)
print("Coeficient is",lr.coef_[0])
```

```
Intercept is 7.991020982270399
Coeficient is 1.3224310227553597
```

Useing Gradient Descent

In [9]:

```
X = data.iloc[:, 0]
Y = data.iloc[:, 1]
```

In [10]:

```
m = 0
c = 0

L = 0.0001  # The learning Rate
epochs = 1000  # The number of iterations to perform gradient descent

n = float(len(X)) # Number of elements in X

# Performing Gradient Descent
for i in range(epochs):
    Y_pred = m*X + c  # The current predicted value of Y
    D_m = (-2/n) * sum(X * (Y - Y_pred))  # Derivative wrt m
    D_c = (-2/n) * sum(Y - Y_pred)  # Derivative wrt c
    m = m - L * D_m  # Update m
    c = c - L * D_c  # Update c

# print (m, c)
print("Intercept is",m)
print("Coeficient is",c)
```

```
Intercept is 1.4777440851894448
Coeficient is 0.08893651993741342
```

In [11]:

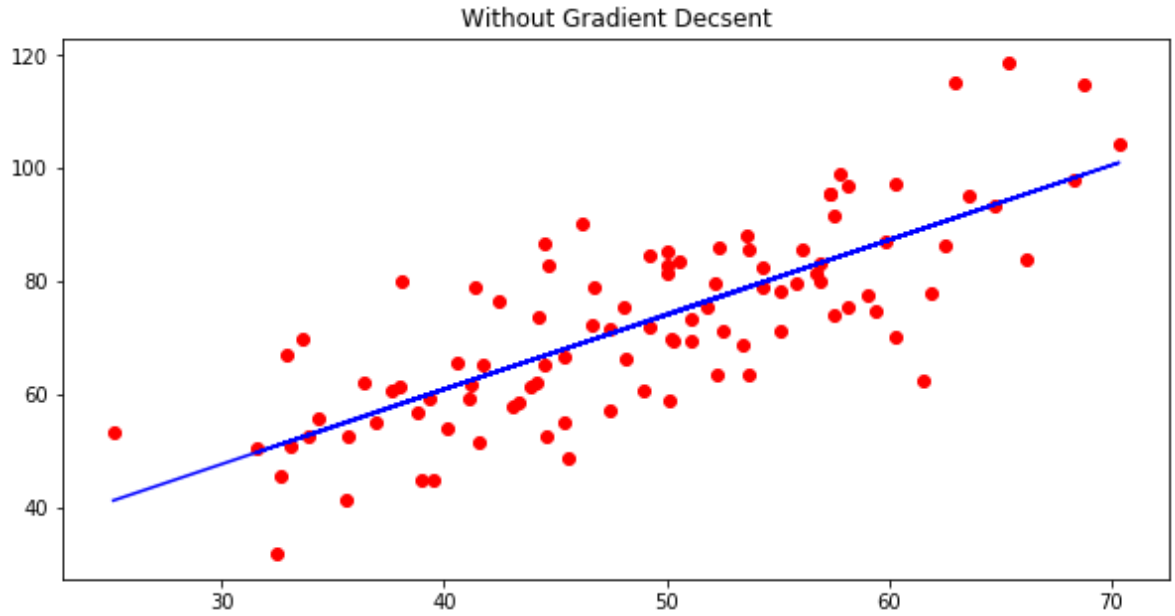
```
# Making predictions
Y_pred = m*X + c
```

See Result In Graph

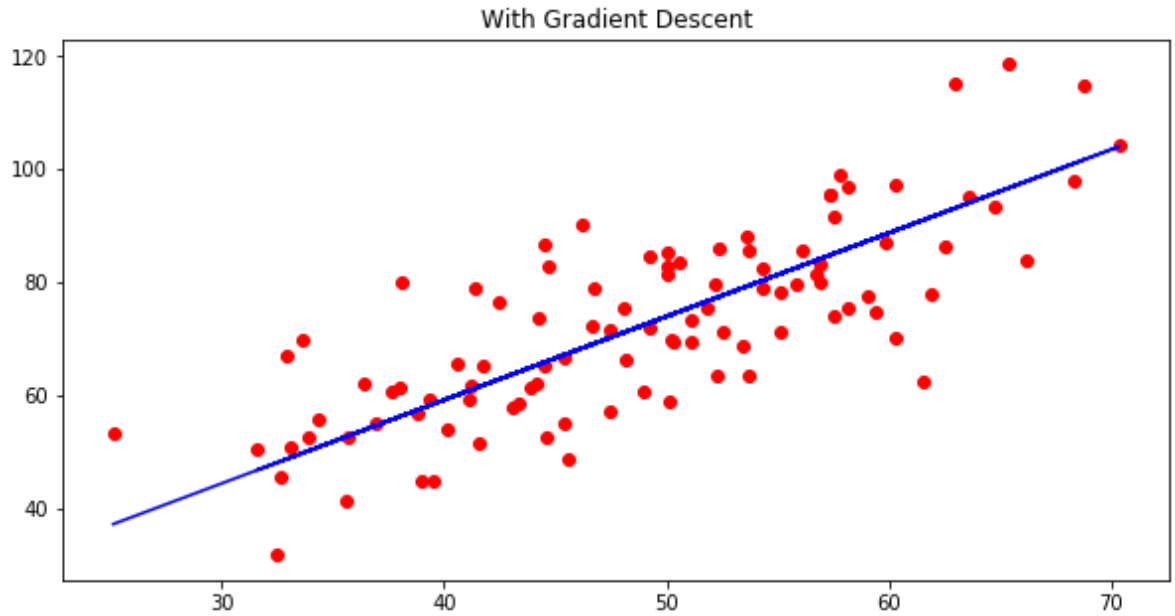
In [12]:

```
print("--------------- Without Gradient Decsent --------------------")
plt.scatter(x,y, color = 'red')
plt.plot(x, lr.predict(x), color = 'blue')
plt.title("Without Gradient Decsent")
plt.show()

print("--------------- With Gradient Decsent --------------------")
plt.scatter(X,Y, color = 'red')
plt.plot(X, Y_pred, color = 'blue')
plt.title("With Gradient Descent")
plt.show()
```

--------------- Without Gradient Decsent --------------------



--------------- With Gradient Decsent --------------------



Logistic Regression

- It's a classification algorithm, that is used where the response variable is categorical. The idea of Logistic Regression is to find a relationship between features and probability of particular outcome.
  - E.g. When we have to predict if a student passes or fails in an exam when the number of hours spent studying is given as a feature, the response variable has two values, pass and fail.
- If the probability is more than 50%, it assigns the value in that particular class else if the probability is less than 50%, the value is assigned to the other class. Therefore, we can say that logistic regression acts as a binary classifier.

Working of a Logistic Model

For linear regression, the model is defined by: $y = \beta_0 + \beta_1 x$ - (i)

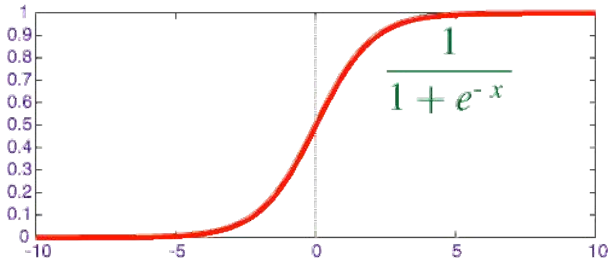
and for logistic regression, we calculate probability, i.e. y is the probability of a given variable x belonging to a certain class. Thus, it is obvious that the value of y should lie between 0 and 1.

But, when we use equation(i) to calculate probability, we would get values less than 0 as well as greater than 1. That doesn't make any sense . So, we need to use such an equation which always gives values between 0 and 1, as we desire while calculating the probability.

So here we Use Sigmoid Function

Sigmoid function

We use the sigmoid function as the underlying function in Logistic regression. Mathematically and graphically, it is shown as:



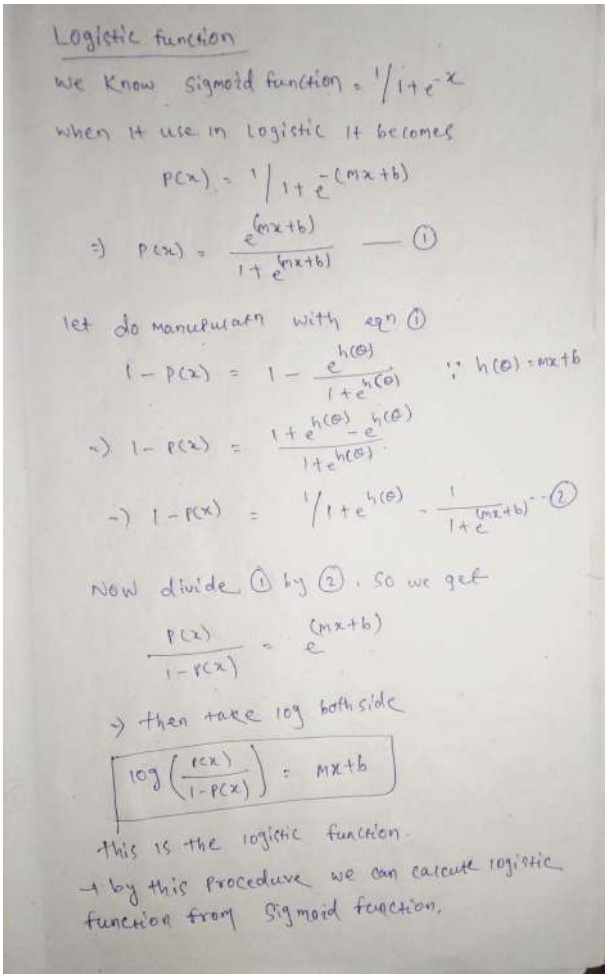
Why do we use the Sigmoid Function?

1) The sigmoid function's range is bounded between 0 and 1. Thus it's useful in calculating the probability for the Logistic function.

2) It's derivative is easy to calculate than other functions which is useful during gradient descent calculation.

3) It is a simple way of introducing non-linearity to the model.

Now Logistic function On Sigmoid Function



Logit Function

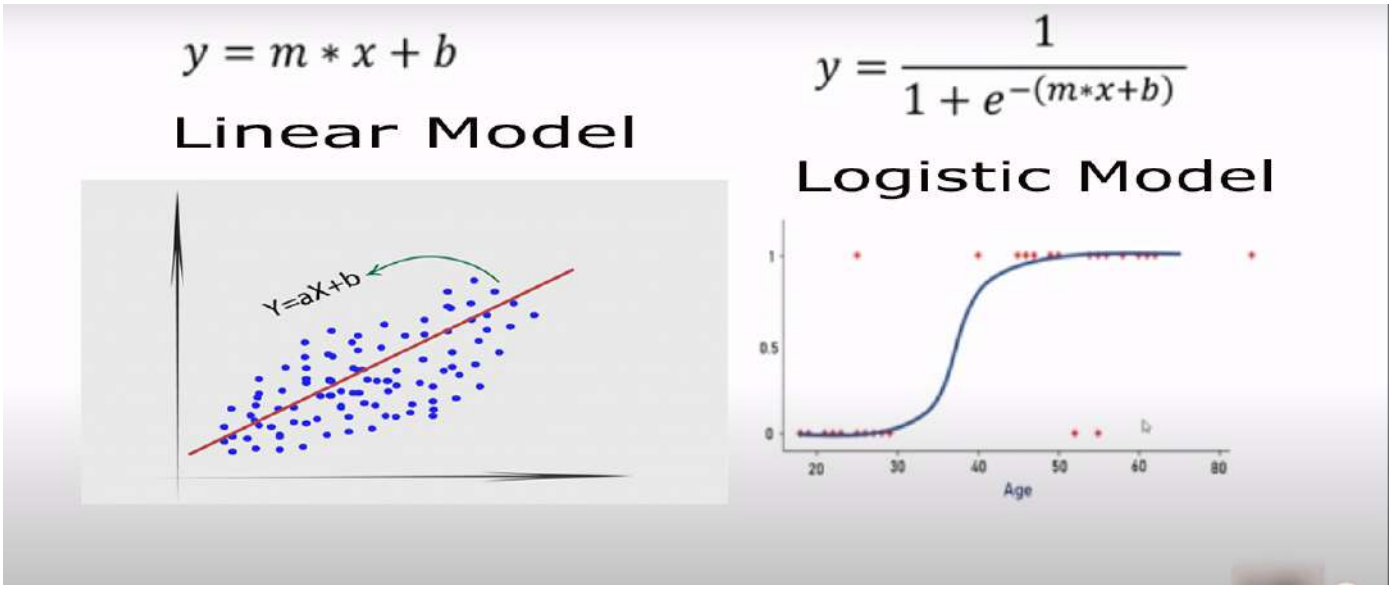
- Logistic regression can be expressed as:

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X.$$

- where, the left hand side is called the logit or log-odds function, and p(x)/(1-p(x)) is called odds.
- The odds signifies the ratio of probability of success to probability of failure. Therefore, in Logistic Regression, linear combination of inputs are mapped to the log(odds) - the output being equal to 1.
- The cost function for the whole training set is given as :

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

Logistic And Linear Model



Pratical Demonstrate Of Logistic Regression

Importing the libraries

In [1]:

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

import warnings
warnings.filterwarnings("ignore")
```

Importing the dataset

In [2]:

```
dataset = pd.read_csv('Social_Network_Ads.csv')
```

In [3]:

```
dataset.head()
```

Out[3]:

	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

In [4]:

```
X = dataset.drop(['Purchased','User ID','Gender'],axis=1)
y = dataset['Purchased']
```

In [5]:

```
X.shape,y.shape
```

Out[5]:

```
((400, 2), (400,))
```

Splitting the dataset into the Training set and Test set

In [6]:

```
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 =
```

Feature Scaling

In [7]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Training the Logistic Regression model on the Training set

In [8]:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(C=1.0)
classifier.fit(X_train, y_train)
```

Out[8]:

```
LogisticRegression()
```

Predicting the Test set results

In [9]:

```
y_pred = classifier.predict(X_test)
```

In [10]:

```
calculation = pd.DataFrame(np.c_[y_test,y_pred], columns = ["Original Purchased","Predict P
calculation
```

Out[10]:

	Original Purchased	Predict Purchased
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
95	1	0
96	0	0
97	1	0
98	1	1
99	1	1

100 rows × 2 columns

Visualising the Training set results

In [11]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.sh
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.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 a 2-D array with a single row if you really want
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 a 2-D array with a single row if you really want
to specify the same RGB or RGBA value for all points.
```

Visualising the Test set results

In [12]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.sh
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.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 a 2-D array with a single row if you really want
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 a 2-D array with a single row if you really want
to specify the same RGB or RGBA value for all points.
```

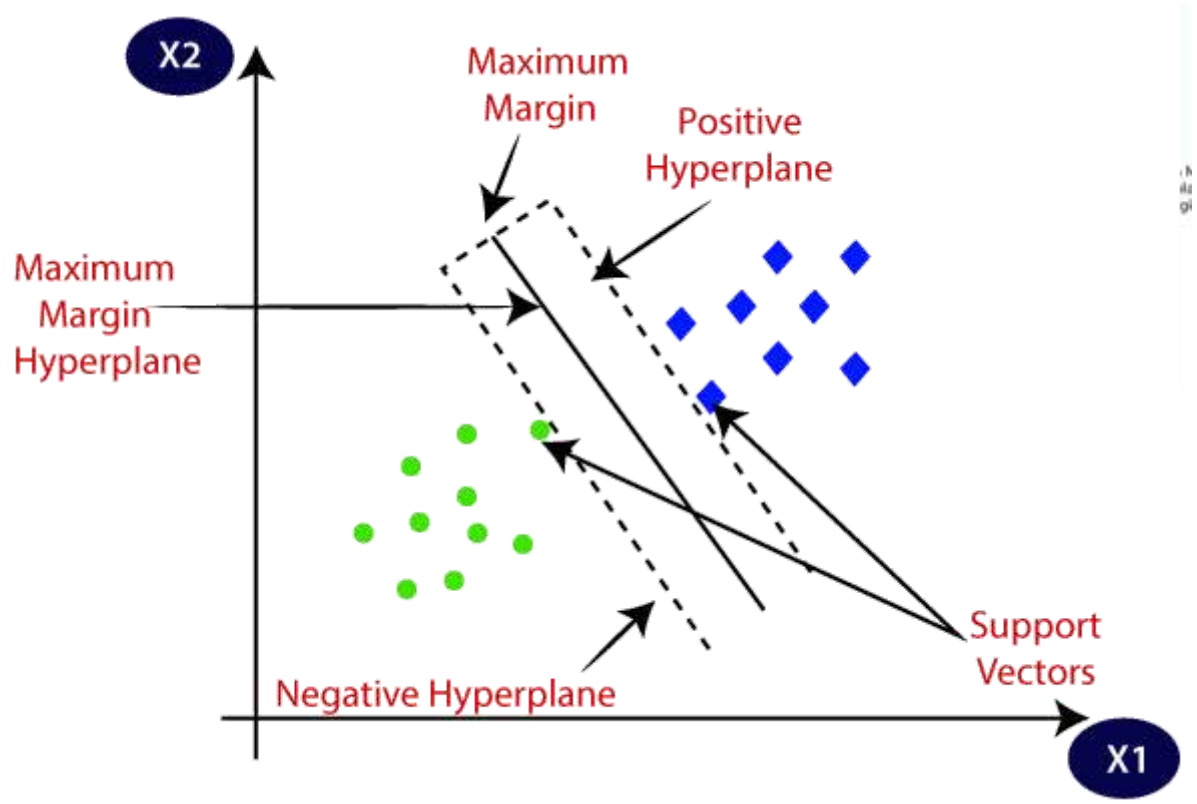
Support Vector Machine (SVM)

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

Befor going to SVM work first know some genral things:

- 1. Support Vector
- 2. Hyper Plane
- 3. Marginal Distance
- 4. Kernel
- 5. Linear and Non Linear Separable Data

1. Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane.
2. Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.
3. Marginal Distance is the two parallel line that create w.r.t most near point of +ve and -ve to plane distance. we always maximise the marginal distance.
4. See the below figure you better understood all these three term.

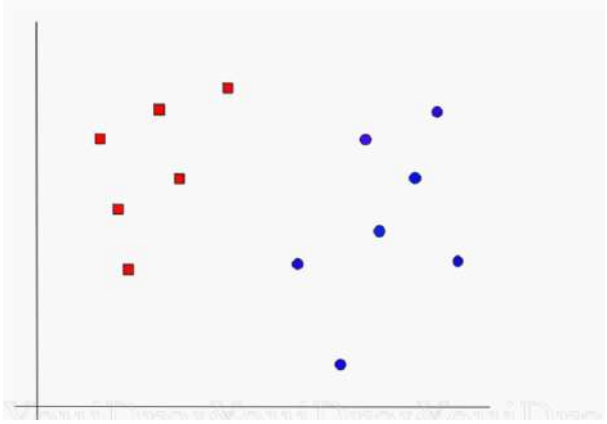


4. The function of kernel is to take data as input and transform it into the required form. Different SVM algorithms use different types of kernel functions. These functions can be different types. For example linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid.

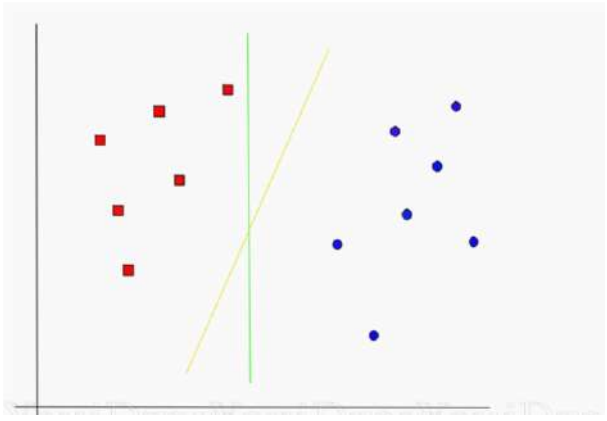
To Know More About Kernel click this:https://data-flair.training/blogs/svm-kernel-functions/#:~:text=SVM%20Kernel%20Functions,it%20into%20the%20required%20form.&text=For%20example%(RBF)%2C%20and%20sigmoid (https://data-flair.training/blogs/svm-kernel-functions/#:~:text=SVM%20Kernel%20Functions,it%20into%20the%20required%20form.&text=For%20example%(RBF)%2C%20and%20sigmoid).

- Below i demonstrate linear and non linear separable dataset.

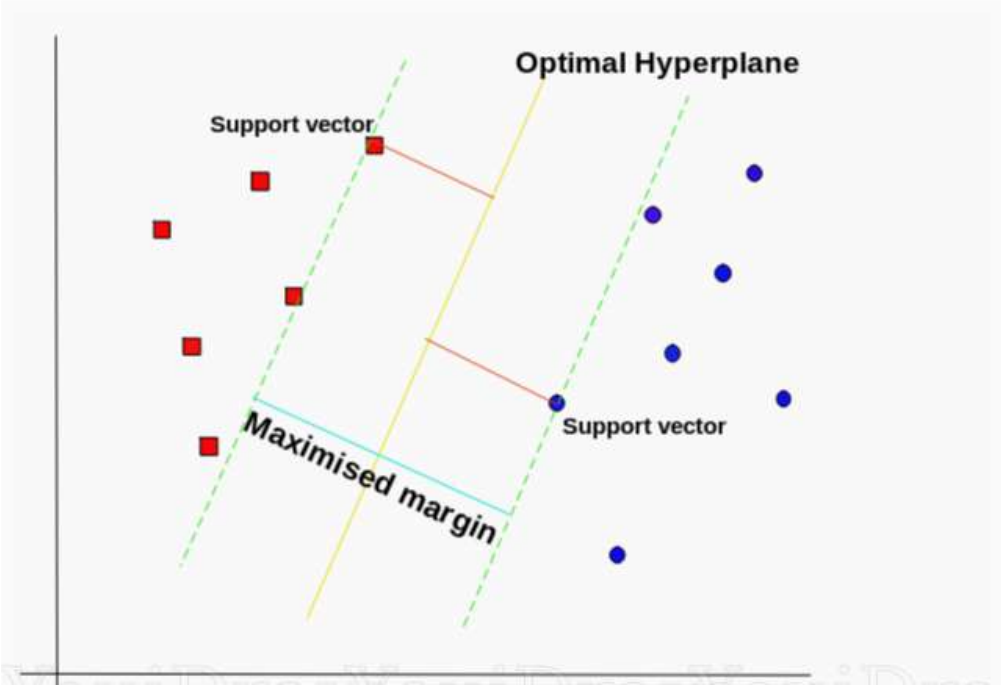
Linear Separable data



- Suppose you have a dataset as shown above and you need to classify the red rectangles from the blue. So your task is to find an ideal line that separates this dataset in two classes.
- It is not so difficult as i learnt before when there is a linear separable data set you just draw a line with the help of linear regression then it will be separable easily.

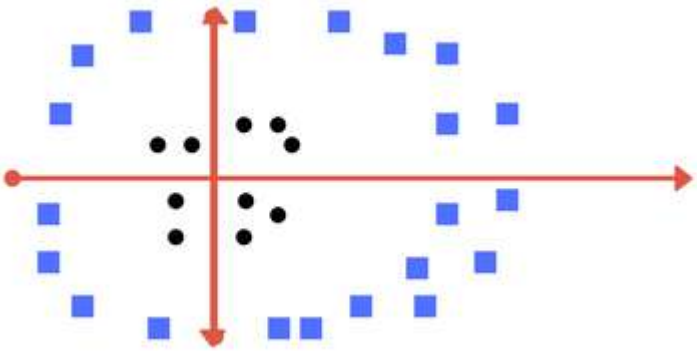


- Two lines here, the green colored line and the yellow colored line. Which line according to you best separates the data?
- If you selected the yellow line then it correct , because thats the line we are looking for. It’s visually quite intuitive in this case that the yellow line classifies better. But, we need something concrete to fix our line.
- Here we see how linear regrssion separate the dataset. Now Times to see How SVM find the best line.
- According to the SVM algorithm that find the points closest to the line from both the classes.These points are called support vectors. Now, we compute the distance between the line and the support vectors.
- This distance is called the margin. Our goal is to maximize the margin. The hyperplane for which the margin is maximum is the optimal hyperplane.

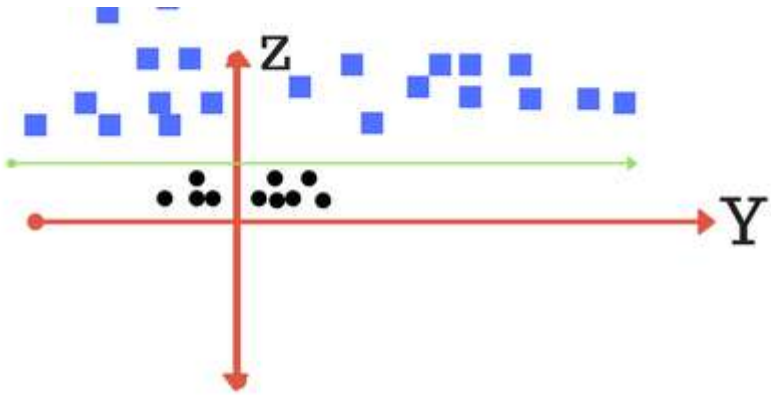


No-Linear Separable Data

- SVM Can work both linear and non- linear dataset. above i dicuss about how svm works in linear data now dicuss about works in non linear data.



- Now consider what if we had data as shown in image below? This is a type of non linear data. Clearly See, there is no line that can separate the two classes in this x-y plane. So what do we do?
- We apply transformation and add one more dimension as we call it z-axis. Lets assume value of points on z plane, $w = x^2 + y^2$ - So, basically z co-ordinate is the square of distance of the point from origin. Let's plot the data on z-axis..
- After Transform



- These Transform Done By SVM kernels.

Math Intution About SVM

SVM Math Intuition

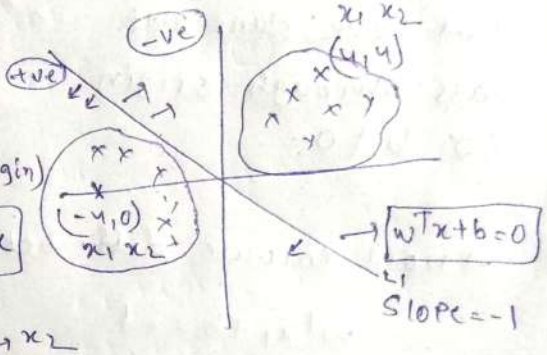
→ In Logistic regression we only plot a single line but in SVM there is a hyperplane. so now learn how to derive this Plane.

ex-1

here $m = -1$

$b = 0$ (as it pass through origin)

So My eqn = $y = w^T x$



$$y = w^T x$$

$$= \begin{bmatrix} -1 \\ 0 \end{bmatrix} \begin{bmatrix} -4 & 0 \\ \uparrow x_1 & \uparrow x_2 \end{bmatrix}$$

→ do matrix multiplication

$= 4 \Rightarrow$ +ve value

$\Rightarrow$ going to allways +ve.

→ when ever going below in the place it is +ve,

Now calculate above the plane

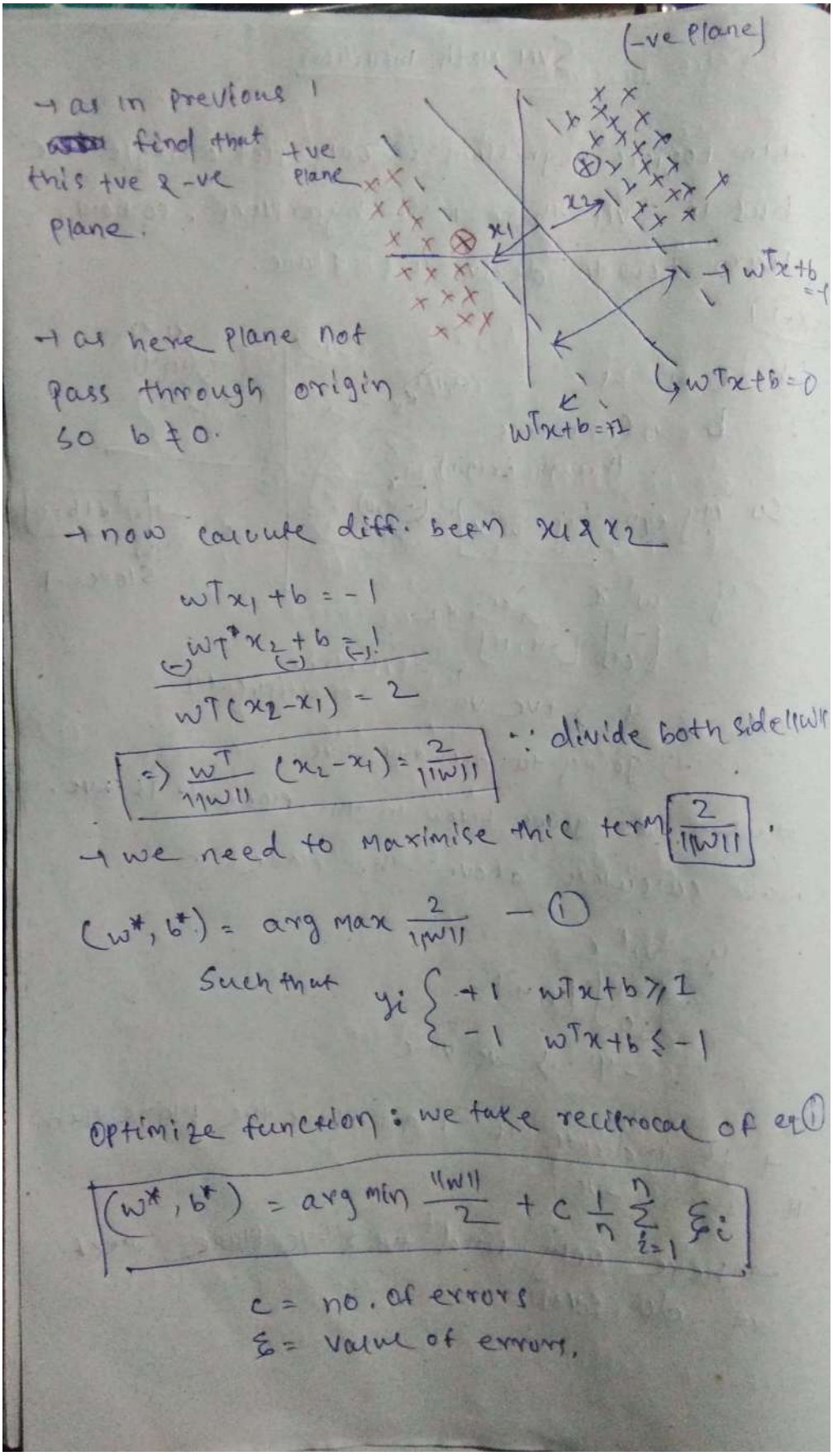
$$y = w^T x$$

$$= \begin{bmatrix} -1 \\ 0 \end{bmatrix} [4, 4]$$

$= -4 \Rightarrow$ -ve value

→ It indicate if I go above the plane then it always -ve.

→ here only find +ve & -ve Plane. next we do SVM.



Let's Do an Example

In [1]:

```
## Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

Importing the dataset

In [2]:

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

In [3]:

```
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 =
```

Feature Scaling

In [4]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Training the SVM model on the Training set

In [5]:

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
```

Out[5]:

```
SVC(kernel='linear', random_state=0)
```

Predicting the Test set results

In [6]:

```
y_pred = classifier.predict(X_test)
```

Making the Confusion Matrix

In [7]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

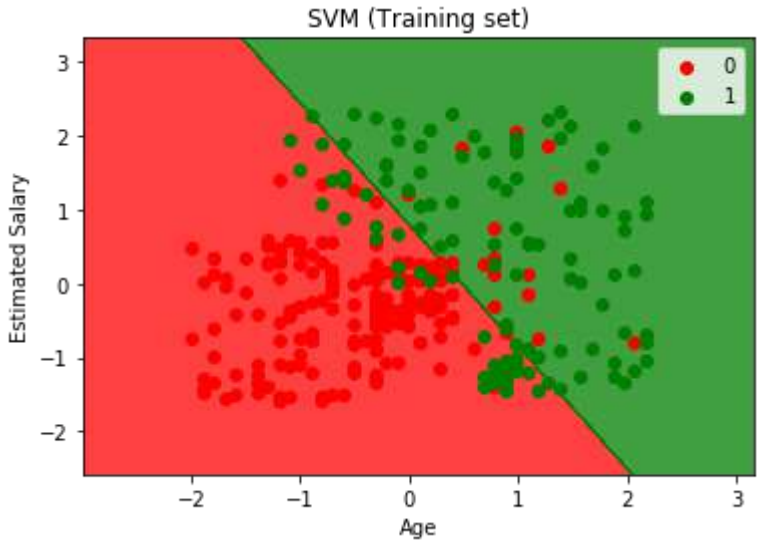
```
[[66  2]
 [ 8 24]]
```

Visualising the Training set results

In [8]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.sh
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.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 a 2-D array with a single row if you really want
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 a 2-D array with a single row if you really want
to specify the same RGB or RGBA value for all points.
```

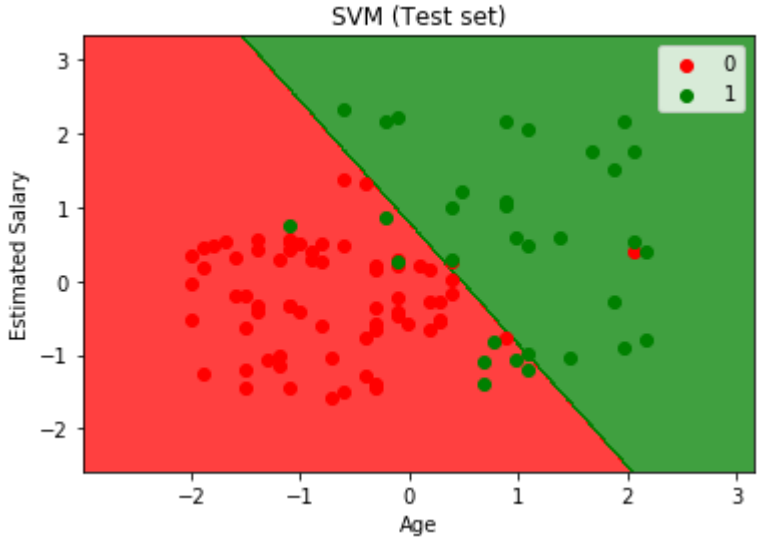


Visualising the Test set results

In [9]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.sh
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.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 a 2-D array with a single row if you really want
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 a 2-D array with a single row if you really want
to specify the same RGB or RGBA value for all points.
```



KNN (K- Nearest Neighbors) Algorithm

1. What is K- Nearest neighbors?

- It is Supervised machine learning algorithm as target variable is known.
- Non parametric as it does not make an assumption about the underlying data distribution pattern.
- It is Lazy algorithm as KNN does not have a training step. All data points will be used only at the time of prediction.
- It Used for both Classification and Regression

2. What is K is K nearest neighbors?

- K is a number used to identify similar neighbors for the new data point.
- KNN takes K nearest neighbors to decide where the new data point with belong to. This decision is based on feature similarity.

3. How do we chose the value of K?

- We can evaluate accuracy of KNN classifier using K fold cross validation.
- Using Elbow Method To Find Right Value OF K.

Before Going To Working of Knn I am showing you to all distance that Knn Use

All Distance:

There are Three Types Of Distance

- Euclidian Distance (L2 Norm)
- Manhattan Distance (L1 Norm)
- Minkowski Distance (Lp Norm)
- Hamming Distance

Euclidian Distance (L2 Norm)

- Euclidean distance is the square root of the sum of squared distance between two points. It is also known as L2 norm.

$$\text{Euclidean Distance between } (x_1, y_1) \text{ and} (x_2, y_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Manhattan Distance (L1 Norm)

- Manhattan distance is the sum of the absolute values of the differences between two points. Also known as L1 norm.

Manhattan Distance between (x_1, y_1) and(x_2, y_2) = $|x_1 - x_2| + |y_1 - y_2|$

Minkowski Distance (Lp Norm)

- Minkowski distance is the used to find distance similarity between two points. When p=1, it becomes Manhattan distance and when p=2, it becomes Euclidean distance.

Minkowski Distance between (x_1, y_1) and(x_2, y_2) = $(|x_1 - x_2|^p + |y_1 - y_2|^p)^{1/p}$

p is any real value. Usually set to a value between 1 and 2

Hamming Distance

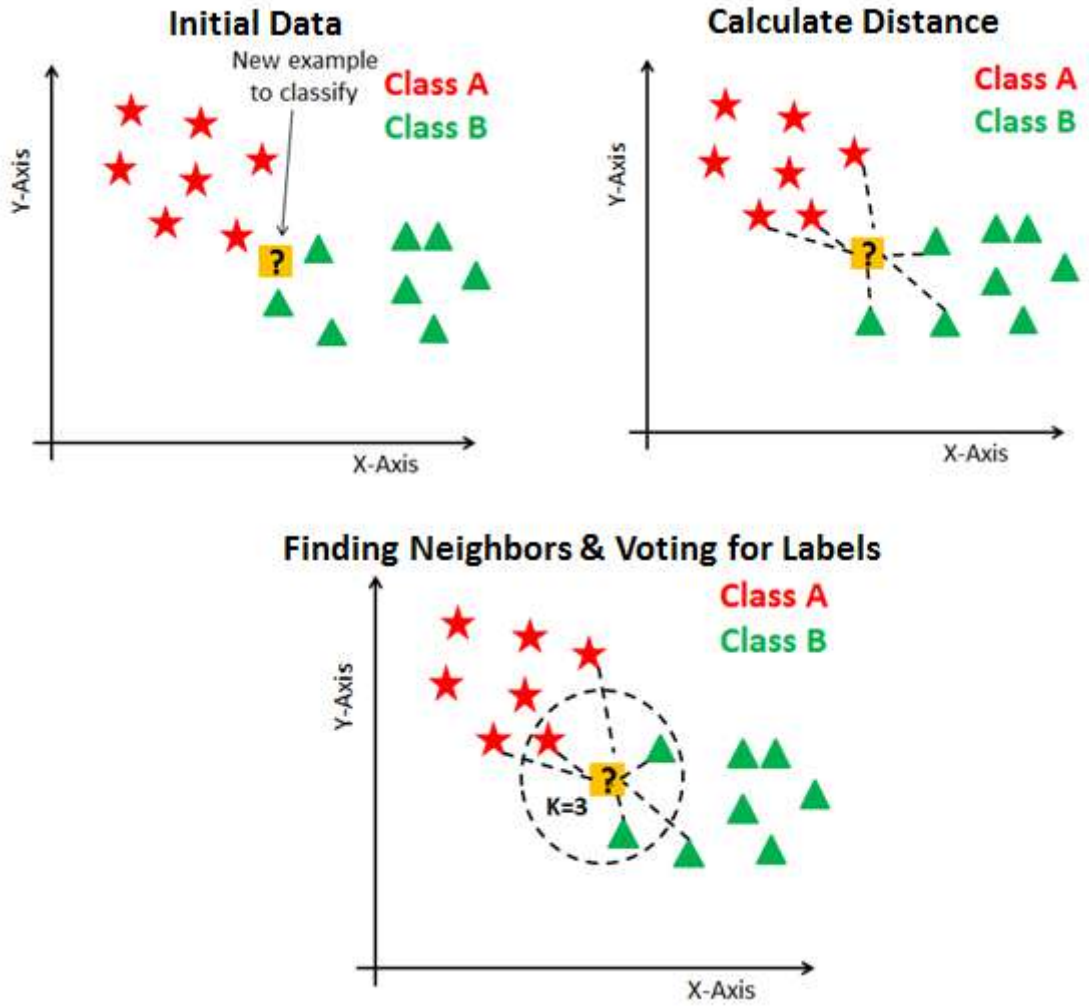
- Hamming distance is used for categorical variables. In simple terms it tells us if the two categorical variables are same or not.

Hemming distance between categorical variables Large and Medium = 1

Hemming distance between categorical variables Large and Large = 0

5. Now Discuss How does KNN work?

- Step 1: Choose a value for K. K should be an odd number.
- Step2: Find the distance of the new point to each of the training data.
- Step 3:Find the K nearest neighbors to the new data point.
- Step 4:
  - For classification, count the number of data points in each category among the k neighbors. New data point will belong to class that has the most neighbors.
  - For regression, value for the new data point will be the average of the k neighbors.



Now See Some Pratical demonstration About KNN

- Here I Only Show Knn Use On Classification Problem.
- You Can Use KNN For Regression Problem Implementation Is same on both Case.

Importing Libraries

In [1]:

```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

Load the data

In [2]:

```
data = pd.read_csv('data_cleaned.csv')
data.shape
```

Out[2]:

```
(891, 25)
```

In [3]:

```
data.head()
```

Out[3]:

	Survived	Age	Fare	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	SibSp_0	Sib
0	0	22.0	7.2500	0	0	1	0	1	0	
1	1	38.0	71.2833	1	0	0	1	0	0	
2	1	26.0	7.9250	0	0	1	1	0	1	
3	1	35.0	53.1000	1	0	0	1	0	0	
4	0	35.0	8.0500	0	0	1	0	1	1	

5 rows × 25 columns

Segregating variables: Independent and Dependent Variables

In [4]:

```
#seperating independent and dependent variables
x = data.drop(['Survived'], axis=1)
y = data['Survived']
x.shape, y.shape
```

Out[4]:

```
((891, 24), (891,))
```

Scaling the data

In [5]:

```
## Importing the MinMax Scaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_scaled = scaler.fit_transform(x)
```

In [6]:

```
x = pd.DataFrame(x_scaled, columns = x.columns)
```

In [7]:

```
x.head()
```

Out[7]:

	Age	Fare	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	SibSp_0	SibSp_1
0	0.271174	0.014151	0.0	0.0	1.0	0.0	1.0	0.0	1.0
1	0.472229	0.139136	1.0	0.0	0.0	1.0	0.0	0.0	1.0
2	0.321438	0.015469	0.0	0.0	1.0	1.0	0.0	1.0	0.0
3	0.434531	0.103644	1.0	0.0	0.0	1.0	0.0	0.0	1.0
4	0.434531	0.015713	0.0	0.0	1.0	0.0	1.0	1.0	0.0

5 rows × 24 columns

In [8]:

```
# Importing the train test split function
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y = train_test_split(x,y, random_state = 56, stratify=y)
```

Implementing KNN Classifier

In [9]:

```
#importing KNN classifier and metric F1score
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.metrics import f1_score
```

In [10]:

```
# Creating instance of KNN
clf = KNN(n_neighbors = 10, metric='euclidean') #here use k=10

#See In metric We use all The above Distance To Calculate. here i Use euclidean distance .

# Fitting the model
clf.fit(train_x, train_y)

# Predicting over the Train Set and calculating F1
test_predict = clf.predict(test_x)
k_1 = f1_score(test_predict, test_y)
```

Elbow for Finding K-Value

In [11]:

```
## Function For Finding K-Value
def Elbow(K):
    #initiating empty list
    test_error = []

    #training model for evey value of K
    for i in K:
        #Instance oh KNN
        clf = KNN(n_neighbors = i)
        clf.fit(train_x, train_y)
        # Appending F1 scores to empty list claculated using the predictions
        tmp = clf.predict(test_x)
        tmp = f1_score(tmp,test_y)
        error = 1-tmp
        test_error.append(error)

    return test_error
```

In [12]:

```
#Defining K range
k = range(6, 20, 2)
```

In [13]:

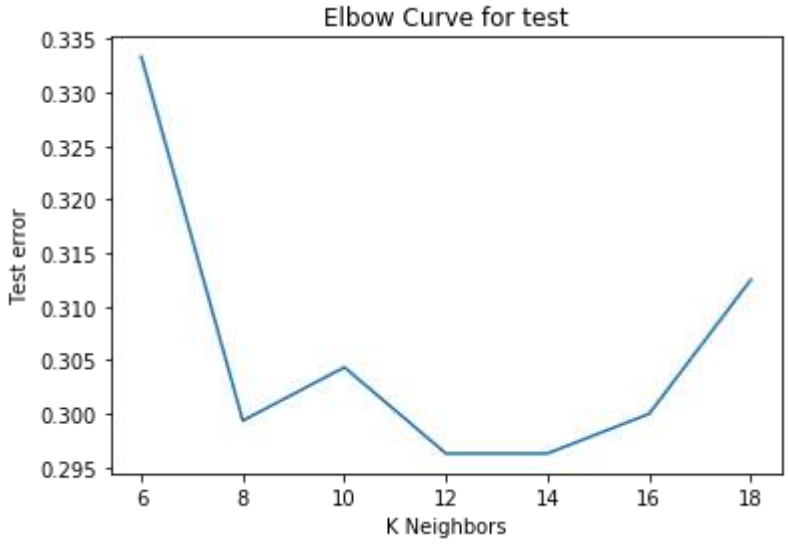
```
# calling above defined function
test = Elbow(k)
```

In [14]:

```
# plotting the Curves
plt.plot(k, test)
plt.xlabel('K Neighbors')
plt.ylabel('Test error')
plt.title('Elbow Curve for test')
```

Out[14]:

```
Text(0.5, 1.0, 'Elbow Curve for test')
```



- Here You See The Curve Down At 27 Then it goes up.

- You need to choose the value of K When the graph never comes down.
- And Always Take Odd Number For K.
- Here I take K=27

In [15]:

```
# Creating instance of KNN
clf = KNN(n_neighbors = 27) #after find K-value by elbow method

# Fitting the model
clf.fit(train_x, train_y)

# Predicting over the Train Set and calculating F1
test_predict = clf.predict(test_x)
k_2 = f1_score(test_predict, test_y)
```

In [16]:

```
print("Before Elbow Method Knn Score: ",k_1)
print("After Elbow Method to put right K-Value Knn Score: ",k_2)
```

```
Before Elbow Method Knn Score:  0.6956521739130435
After Elbow Method to put right K-Value Knn Score:  0.751592356687898
```

- See Here Before Randomly Put K Value The Score is not well. But After Doing Elbow Method And Find Appropriate K-Value Then the score increase to 75.
- So Now you understood how imporatant K-Value For Knn Algorithm.
- This All About Knn Algorithm

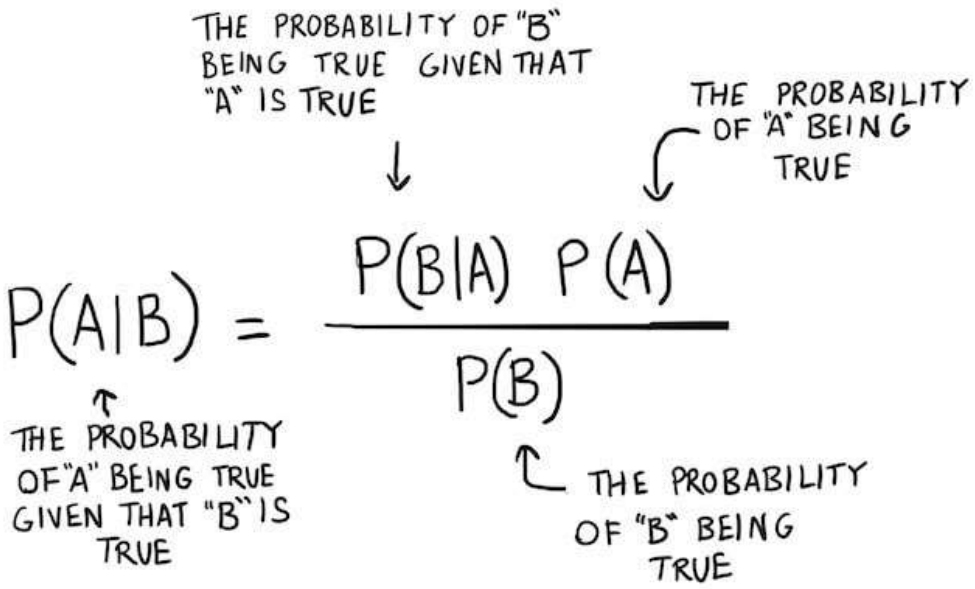
Naive Bayes

- Another very popular Supervised Classification algorithm is Naive Bayes.
- Before diving deep lets understand What does Naive and Bayes signify.

1. This algorithm is called “Naive” because it makes a naive assumption that each feature is independent of other features which is not true in real life.
2. As for the “Bayes” part, it refers to the statistician and philosopher, Thomas Bayes and the theorem named after him, Bayes’ theorem, which is the base for Naive Bayes Algorithm.

Befor Going Naive Bayes we see about Bayes Therom ¶

Bayes Theorm



Where,

- P(A|B) is the probability of hypothesis A given the data B. This is called the posterior probability.
- P(B|A) is the probability of data B given that the hypothesis A was true.
- P(A) is the probability of hypothesis A being true (regardless of the data). This is called the prior probability of A.
- P(B) is the probability of the data (regardless of the hypothesis).
- P(A|B) or P(B|A) are conditional probabilities P(B|A) = P(A and B)/P(A)

Types of Naive Bayes Classifier:

1. Multinomial Naive Bayes
2. Bernoulli Naive Bayes
3. Gaussian Naive Bayes

a. Multinomial Naive Bayes: This is mostly used for document classification problem, i.e whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document.

b. Bernoulli Naive Bayes: This is similar to the multinomial naive bayes but the predictors are boolean variables. The parameters that we use to predict the class variable take up only values yes or no, for example if a word occurs in the text or not.

c. Gaussian Naive Bayes : When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution.

How Naive Bayes algorithm works?

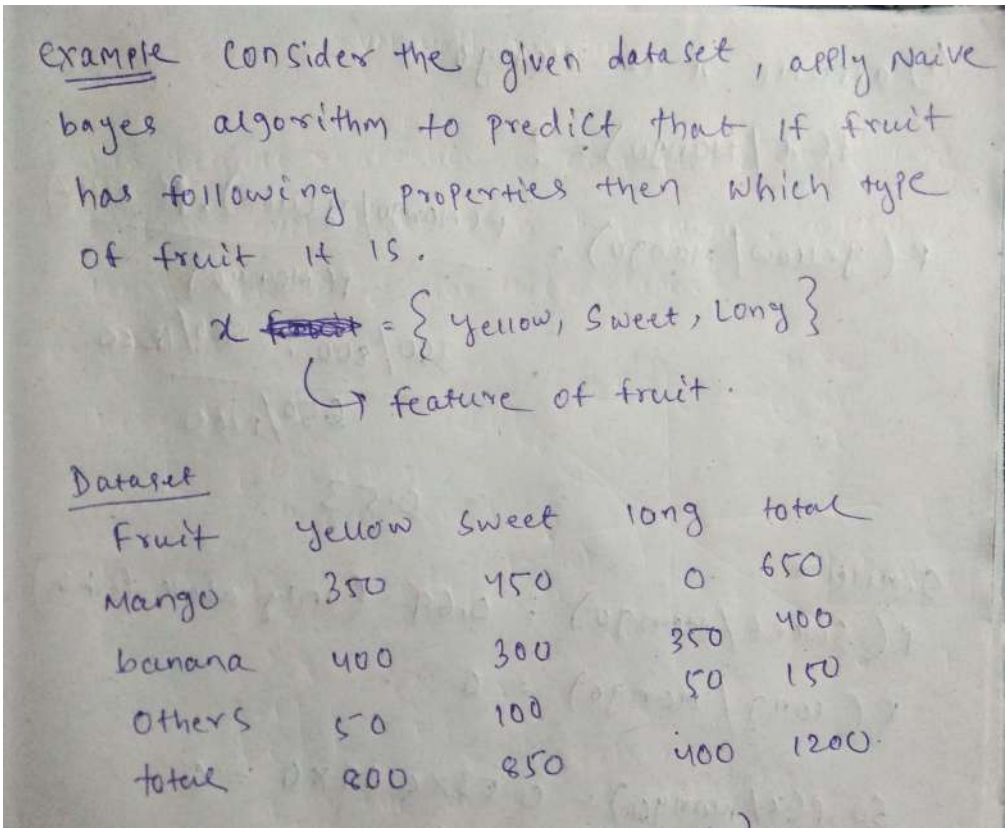
Let's understand it using an clasic example.

Below I have a training data set of weather and corresponding target variable 'Play' (suggesting possibilities of playing). Now, we need to classify whether players will play or not based on weather condition. Let's follow the below steps to perform it.

- Step 1: Convert the data set into a frequency table
- Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.
- Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

See the below example

This is the problem statement :



Here is the Answer:

① assume x is Mango

$P(x/Mango) = ?$

$$P(yellow/Mango) = \frac{P(Mango/yellow) \cdot P(yellow)}{P(Mango)}$$

$$= \frac{350/800 \cdot 800/1200}{650/1200}$$

$$= 0.53$$

Similarly

$P(Sweet/Mango) = 0.69$ (apply above formula)

$P(long/Mango) = 0$

So $P(x/Mango) = 0.53 \times 0.69 \times 0 = 0$

$P(x/Mango) = 0$

② assume x is banana.

$$P(yellow/banana) = \frac{P(banana/yellow) \cdot P(yellow)}{P(banana)}$$

$$= \frac{480/800 \cdot \frac{800}{1200}}{400/1200} = 1$$

$P(Sweet/banana) = 0.75$

$P(long/banana) = 0.875$

So $P(x/banana) = 1 \times 0.75 \times 0.87$

$= 0.65$

$P(x/banana = 0.65$

③ assume x is others

$P(yellow/others) = 0.33$

$P(Sweet/others) = 0.66$

$P(long/others) = 0.33$

$P(x/others) = 0.33 \times 0.66 \times 0.33 = 0.072$

So by this three we found 0.65 is large so I conclude x is belongs to banana.

In this way naive bays work.

Applications of Naive Bayes Algorithm :

- Naive Bayes is widely used for text classification
- Another example of Text Classification where Naive Bayes is mostly used is Spam Filtering in Emails
- Other Examples include Sentiment Analysis ,Recommender Systems etc

Now Implement This Algorithm

Importing the libraries

In [0]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

In [0]:

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

In [0]:

```
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 =
```

Feature Scaling

In [0]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Training the Naive Bayes model on the Training set

In [5]:

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

Out[5]:

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

Predicting the Test set results

In [0]:

```
y_pred = classifier.predict(X_test)
```

Making the Confusion Matrix

In [7]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

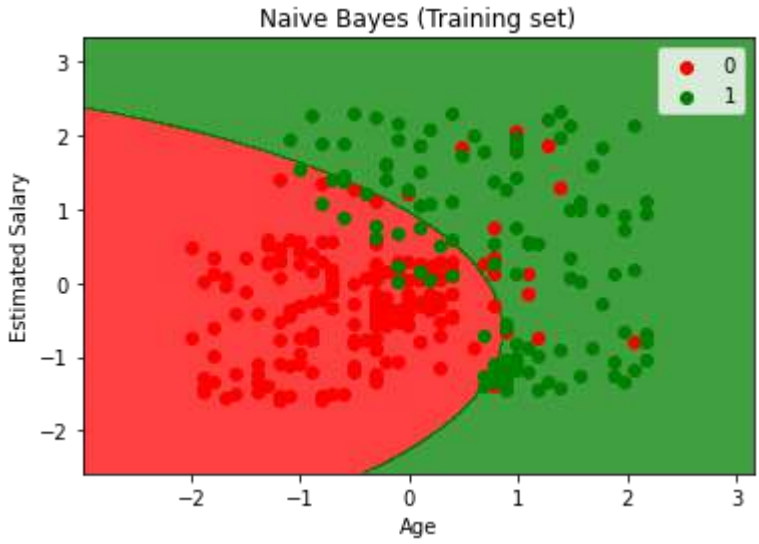
```
[[65  3]
 [ 7 25]]
```

Visualising the Training set results

In [8]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.sh
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naive Bayes (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.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 a 2-D array with a single row if you really want
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 a 2-D array with a single row if you really want
to specify the same RGB or RGBA value for all points.
```

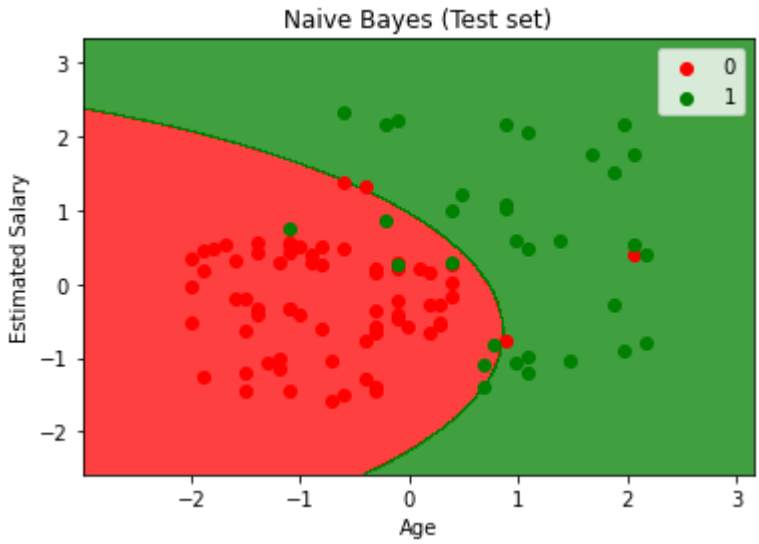


Visualising the Test set results

In [9]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.sh
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naive Bayes (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.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 a 2-D array with a single row if you really want
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 a 2-D array with a single row if you really want
to specify the same RGB or RGBA value for all points.
```



Decision Tree

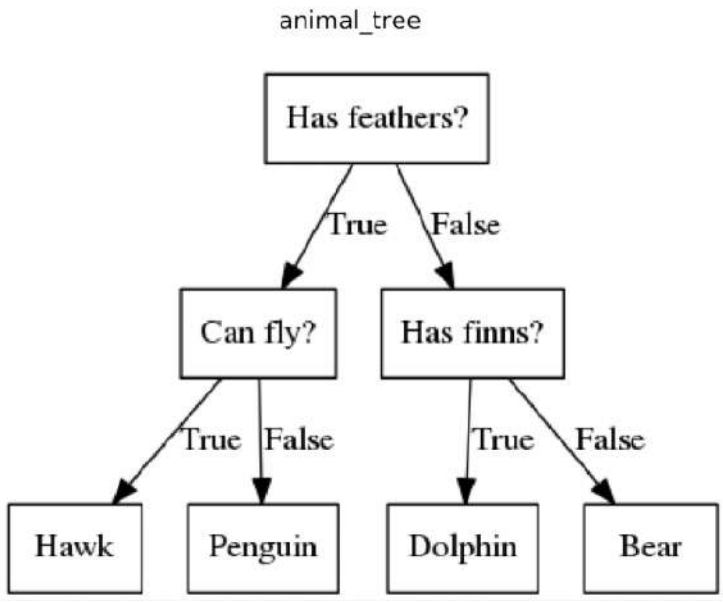
- Decision tree is one of the most popular machine learning algorithms used all along.
- Decision trees are used for both classification and regression problems.

1. Why Decision trees?

We have couple of other algorithms there, so why do we have to choose Decision trees??

- Decision tress often mimic the human level thinking so its so simple to understand the data and make some good interpretations.
- Decision trees actually make you see the logic for the data to interpret(not like black box algorithms like SVM,NN,etc..)
- A decision tree follows a set of if-else conditions to visualize the data and classify it according to the conditions.

For example :

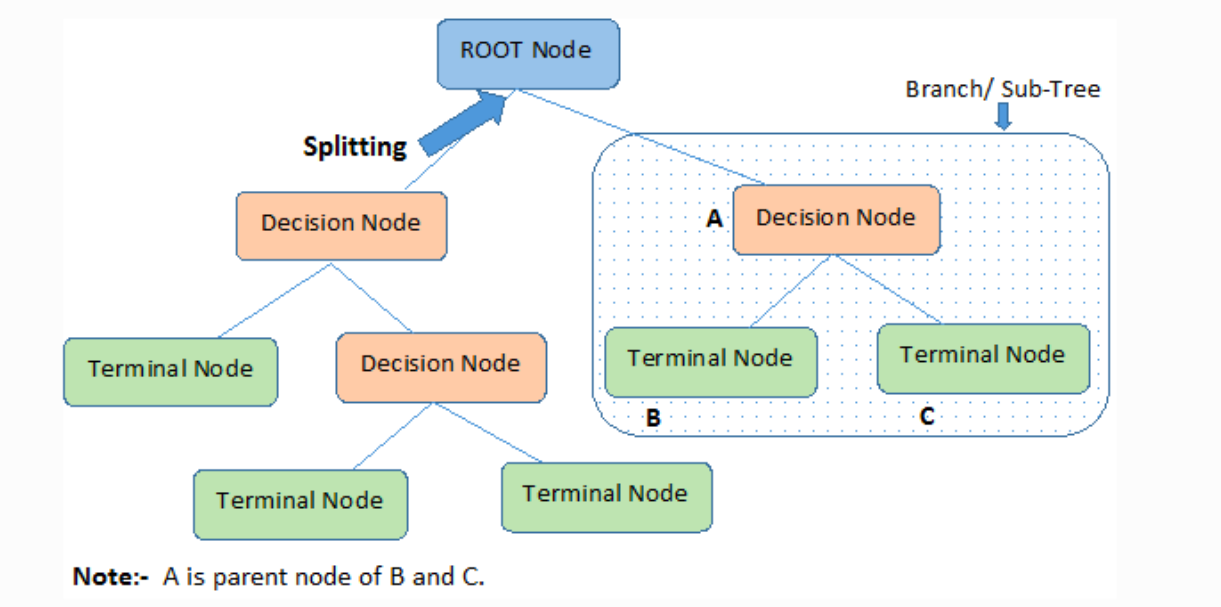


- A decision tree is a tree where each node represents a feature(attribute), each link(branch) represents a decision(rule) and each leaf represents an outcome(categorical or continues value).

Important terminology In DT

- Root Node: This attribute is used for dividing the data into two or more sets. The feature attribute in this node is selected based on Attribute Selection Techniques.
- Branch or Sub-Tree: A part of the entire decision tree is called branch or sub-tree.
- Splitting: Dividing a node into two or more sub-nodes based on if-else conditions.
- Decision Node: After splitting the sub-nodes into further sub-nodes, then it is called as the decision node.

- Leaf or Terminal Node: This is the end of the decision tree where it cannot be split into further sub-nodes.
- Pruning: Removing a sub-node from the tree is called pruning.



Types of Decision Trees

- Types of decision tree is based on the type of target variable we have. It can be of two types:

1. Categorical Variable Decision Tree: Decision Tree which has categorical target variable then it called as categorical variable decision tree. Example:- In above scenario of student problem, where the target variable was “Student will play cricket or not” i.e. YES or NO.
2. Continuous Variable Decision Tree: Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree.

Three Important Term In DT

- Entropy(H)
- Information Gain(IG)
- Gini Impurity

Entropy

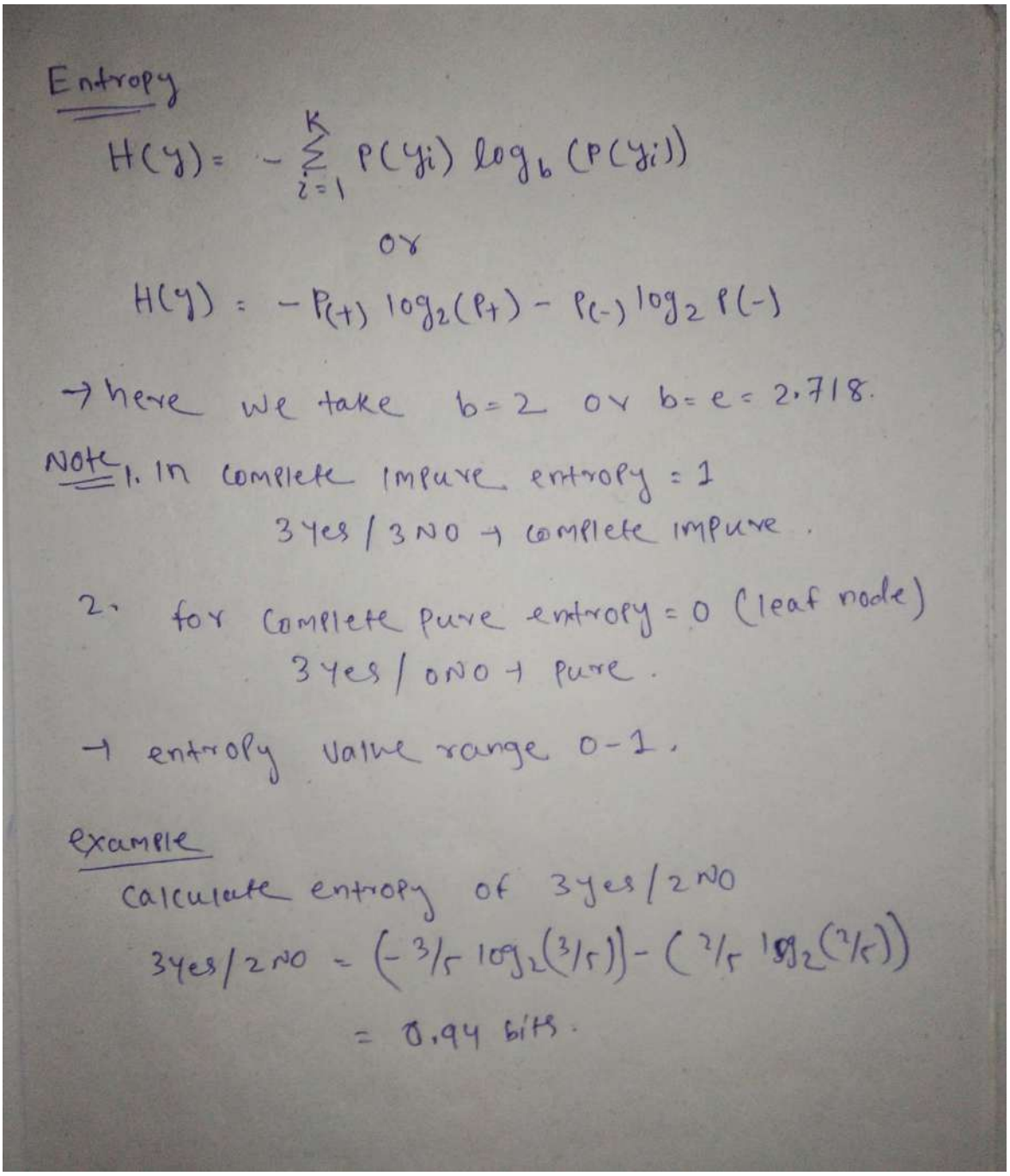
-> It is select the best feature/attribute in DT for Splitting Purpose

-> It helps to measure purity of split

-> Pure split is either get yes or no but not both.

-> Eg. 3yes/2No - it is not pure But 3Yes/0No - it is pure

-> we keep split whenever pure split not come.



2. Information Gain(IG)

-> Information gain is calculated by comparing the entropy of the dataset before and after a transformation.

-> If there are many entropy then we compute average of all entropy and compare to find which is best this done by information gain

-> Information gain is the reduction in entropy or surprise by transforming a dataset and is often used in training decision trees.

$$GAIN_{split} = Entropy(p) - \left(\sum_{i=1}^{k} \frac{n_i}{n} Entropy(i) \right)$$

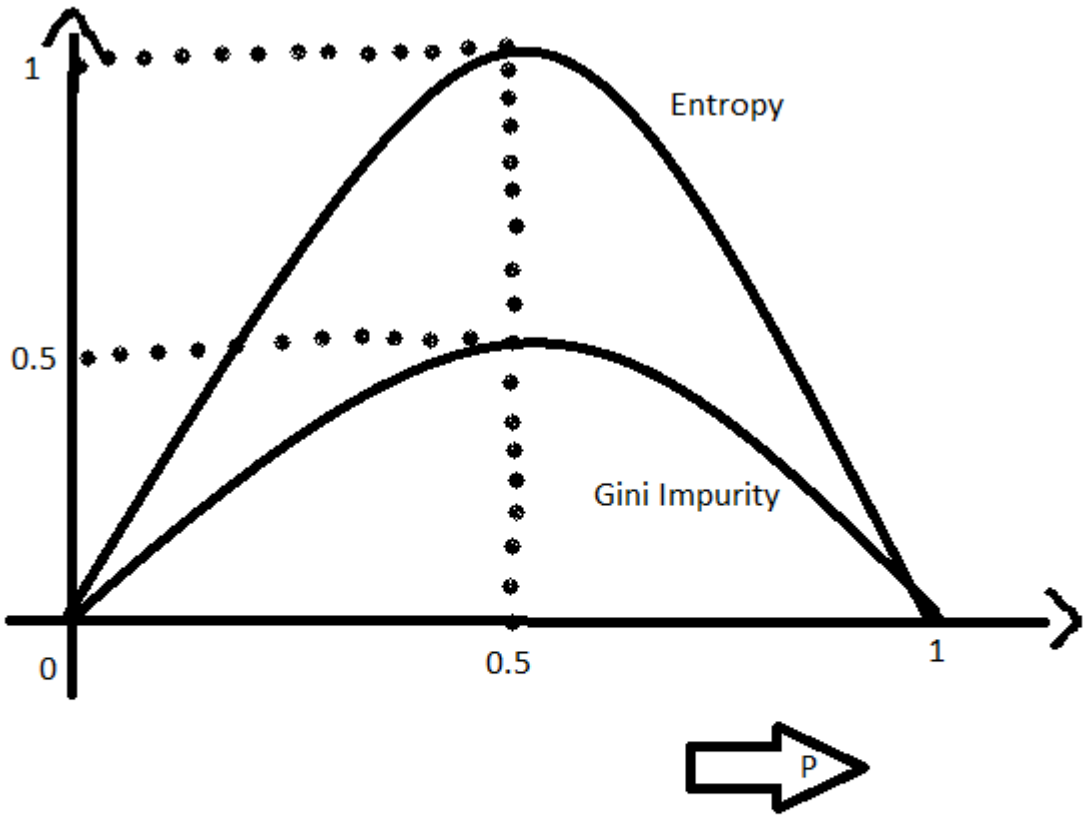
3. Gini Impurity

-> Both Entropy and Gini Impurity Calculate the purity of split in DT.

-> But Gini Impurity is better than Entropy. So we use Gini Impurity to find purity of split.

-> Gini Impurity is a measurement of the likelihood of an incorrect classification of a new instance of a random variable, if that new instance were randomly classified according to the distribution of class labels from the data set.

Lets plot the Entropy Vs Gini impurity



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

$$Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

-> Entropy Max Value is 1 but Gini Impurity Max Value is 0.5.

-> As Gini Impurity has no logarithmic function so it is more efficient than Entropy.

Tree Pruning

Tree pruning is the method of trimming down a full tree (obtained through the above process) to reduce the complexity and variance in the data. Just as we regularised linear regression, we can also regularise the decision tree model by adding a new term.

Post-pruning

Post-pruning, also known as backward pruning, is the process where the decision tree is generated first and then the non-significant branches are removed. Cross-validation set of data is used to check the effect of pruning and tests whether expanding a node will make an improvement or not. If any improvement is there then we continue by expanding that node else if there is reduction in accuracy then the node not be expanded and should be converted in a leaf node.

Pre-pruning

Pre-pruning, also known as forward pruning, stops the non-significant branches from generating. It uses a condition to decide when should it terminate splitting of some of the branches prematurely as the tree is generated.

How does the Decision Tree algorithm Work?

- In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.
- For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM). Here it use Entropy or Gini Impurity.

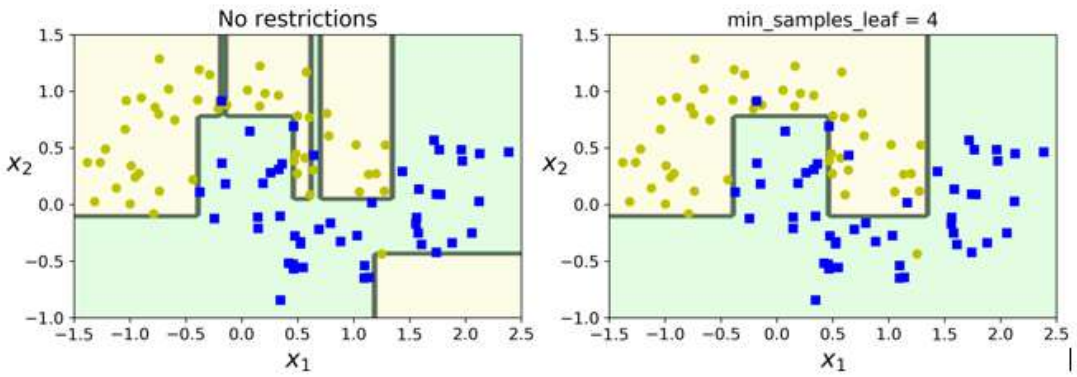
Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

See An Example Of Above Algorithm

- Here YOu see the overview of DT, On the left, the Decision Tree is trained with the default hyperparameters (i.e., no restrictions), and on the right the Decision Tree is trained with min_sam ples_leaf=4. It is quite obvious that the model on the left is overfitting, and the model on the right will probably generalize better.



Lets Implement With Python

Importing the libraries

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

Importing the dataset

In [2]:
```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

In [3]:
```
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, randc
```

Feature Scaling

In [4]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Training the Decision Tree Classification model on the Training set

In [5]:
```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

Out[5]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
```

Predicting the Test set results

In [6]:
```
y_pred = classifier.predict(X_test)
```

Making the Confusion Matrix

In [7]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[62  6]
 [ 3 29]]
```

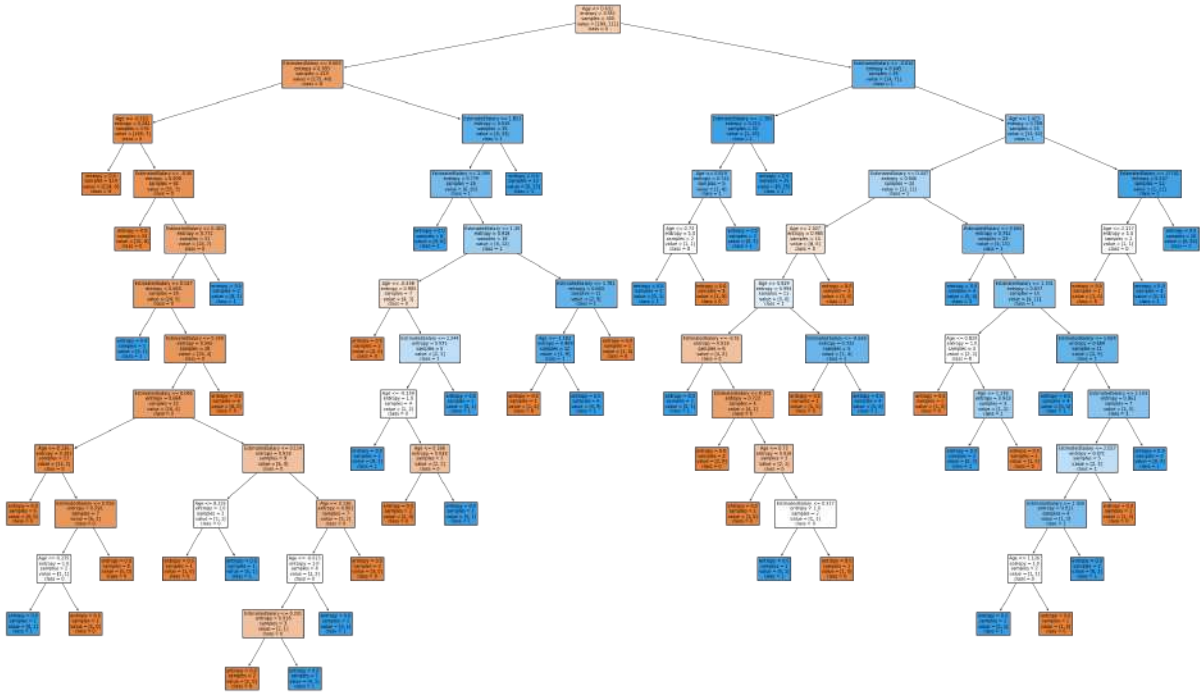
In [15]:
```
dataset.head()
```

Out[15]:

	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

In [10]:
```
from sklearn import tree
```

In [20]:
```
fn=['Age','EstimatedSalary']
cn=['0','1']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (50,30))
tree.plot_tree(classifier,
               feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('imagename.png')
```



Visualising the Training set results

In [8]:
```
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).res
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.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 a 2-D array with a single row if you really want to spec
ify 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 a 2-D array with a single row if you really want to spec
ify the same RGB or RGBA value for all points.
```

- The above output is completely different from the rest classification models. It has both vertical

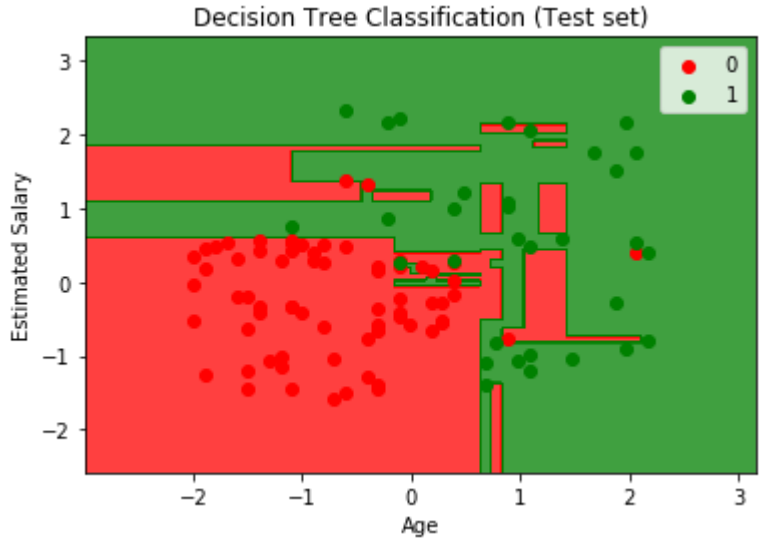
and horizontal lines that are splitting the dataset according to the age and estimated salary variable.

- As we can see, the tree is trying to capture each dataset, which is the case of overfitting.

Visualising the Test set results

In [9]:
```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).res
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Decision Tree Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.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 a 2-D array with a single row if you really want to spec
ify 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 a 2-D array with a single row if you really want to spec
ify the same RGB or RGBA value for all points.
```

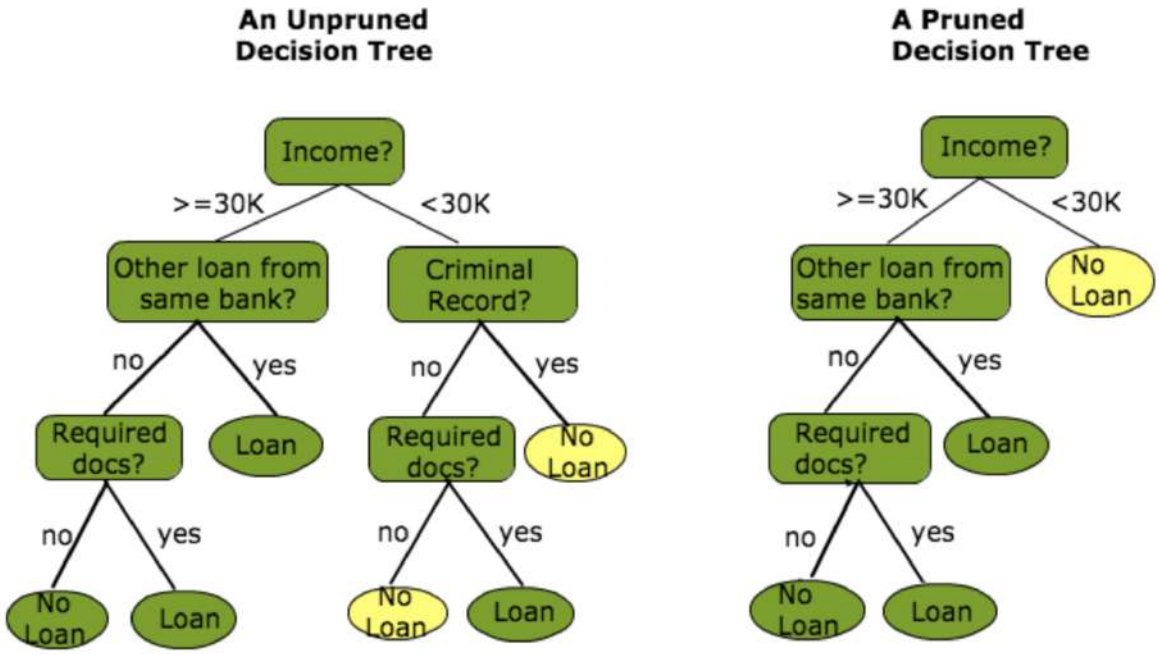


As we can see in the above image that there are some green data points within the red region and vice versa. So, these are the incorrect predictions.

Pruning

- Machine learning is a problem of trade-offs. The classic issue is over-fitting versus under-fitting. Over-fitting happens when a model fits on training data so well and it fails to generalize well.ie, it also learns noises on top of the signal. Under-fitting is an opposite event: the model is too simple to find the patterns in the data.
- Pruning is a technique used to deal with overfitting, that reduces the size of DTs by removing sections of the Tree that provide little predictive or classification power.
- The goal of this procedure is to reduce complexity and gain better accuracy by reducing the effects of overfitting and removing sections of the DT that may be based on noisy or erroneous data. There are two different strategies to perform pruning on DTs:

```
  1. Pre-prune: When you stop growing DT branches when information b
ecomes unreliable.
  2. Post-prune: When you take a fully grown DT and then remove leaf
nodes only if it results in a better model performance. This way, yo
u stop removing nodes when no further improvements can be made.
```



Example On A Dataset

In [1]:
```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn.tree import DecisionTreeClassifier
```

In [2]:
```
X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

clf = DecisionTreeClassifier(random_state=0)
clf.fit(X_train,y_train)
```

Out[2]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
```

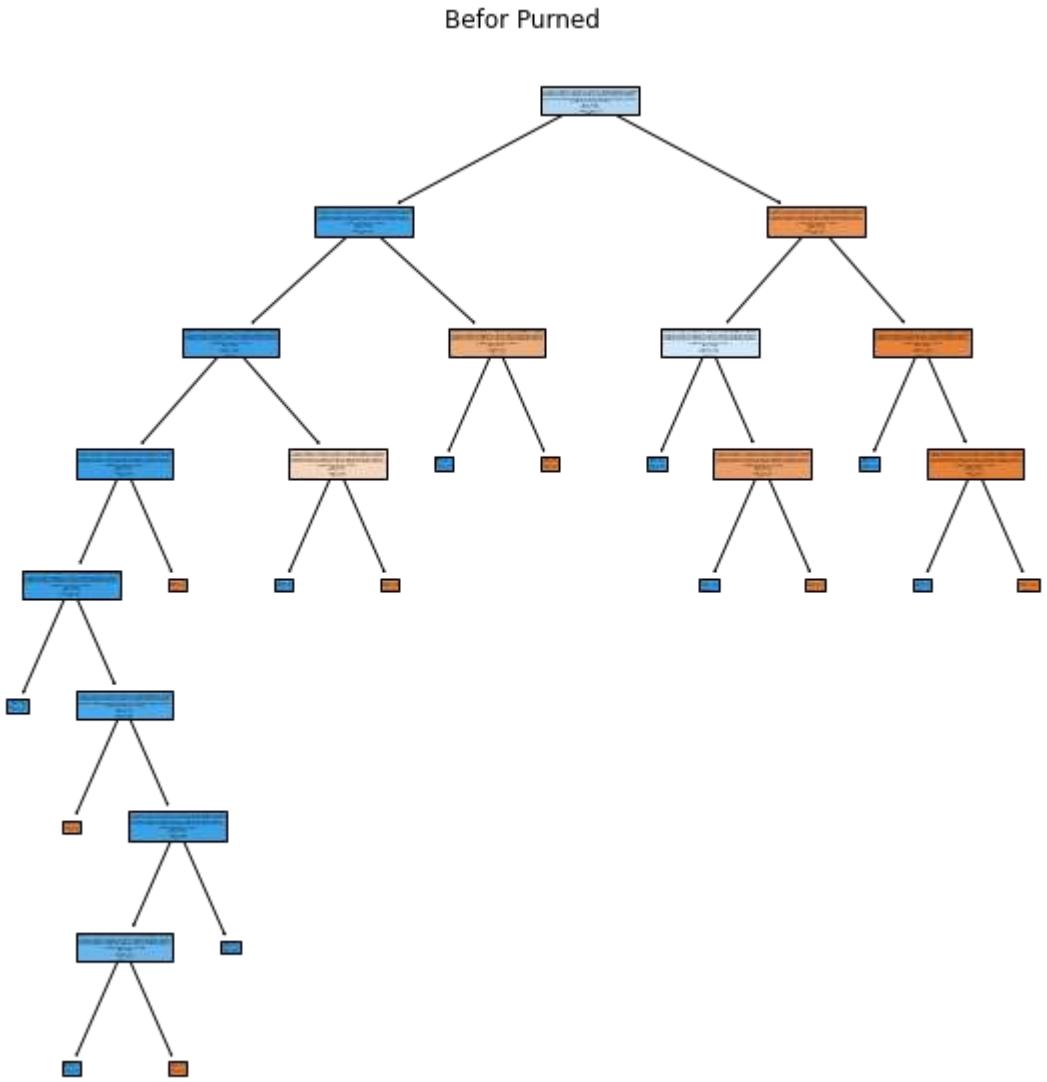
In [3]:
```
pred=clf.predict(X_test)
from sklearn.metrics import accuracy_score
print("Training Accuracy :", clf.score(X_train, y_train))
print("Testing Accuracy :", accuracy_score(y_test,pred))
```

```
Training Accuracy : 1.0
Testing Accuracy : 0.8811188811188811
```

We can see that in our train data we have 100% accuracy. But in test data model is not well generalizing. We have just 88% accuracy. Over model is clearly overfitting. We will avoid overfitting through pruning. We will do cost complexity prunning

In [4]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [5]:
```
from sklearn import tree
plt.figure(figsize=(10,10))
features = X
classes = ['0','1']
tree.plot_tree(clf,feature_names=features,class_names=classes,filled=True)
plt.title('Befor Pruning')
plt.show()
```



So here We use Two techniques Post and Pre pruning

Methood 1: Pre Pruning techniques

- Pre pruning is nothing but stoping the growth of decision tree on an early stage. For that we can limit the growth of trees by setting constrains. We can limit parameters like max_depth , min_samples etc.
- An effective way to do is that we can grid search those parameters and choose the optimum values that gives better performace on test data.
- it is our manually hyperparameter technique
- As of now we will control these parameters

```
        - max_depth: maximum depth of decision tree
        - min_sample_split: The minimum number of samples required to
    split an internal node:
        - min_samples_leaf: The minimum number of samples required to
    be at a leaf node.
```

In [6]:
```
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
params = {'max_depth': [2,4,6,8,10,12],
          'min_samples_split': [2,3,4],
          'min_samples_leaf': [1,2]}

clf = tree.DecisionTreeClassifier()
gcv = GridSearchCV(estimator=clf,param_grid=params)
gcv.fit(X_train,y_train)
```

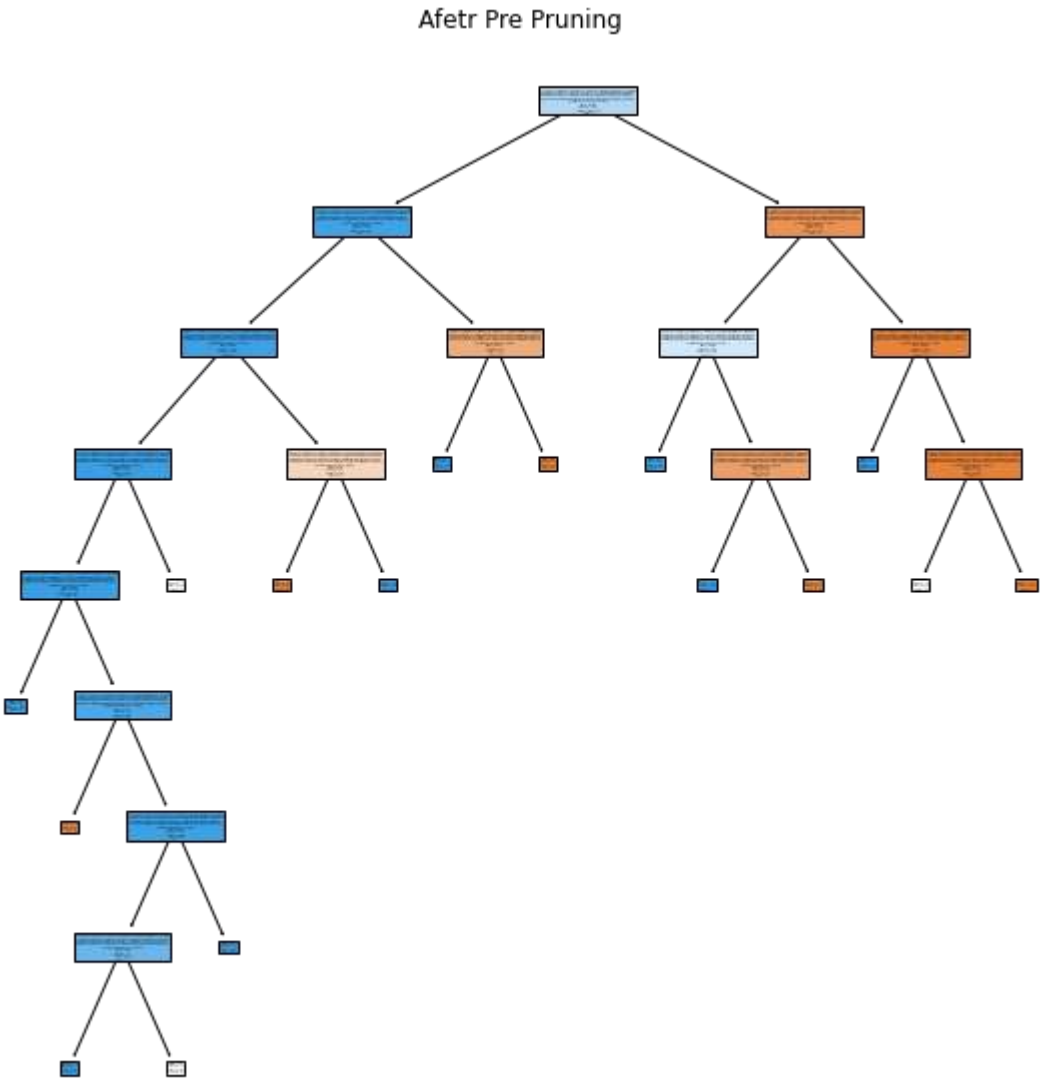
Out[6]:
```
GridSearchCV(cv=None, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [2, 4, 6, 8, 10, 12],
                         'min_samples_leaf': [1, 2],
                         'min_samples_split': [2, 3, 4]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [7]:
```
model = gcv.best_estimator_
model.fit(X_train,y_train)
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

print(f'Train score {accuracy_score(y_train_pred,y_train)}')
print(f'Test score {accuracy_score(y_test_pred,y_test)}')
```

```
Train score 0.9929577464788732
Test score 0.9230769230769231
```

We can see that tree is pruned and there is less improvement in test accuracy.But still there is still scope of improvement. So now we Go towards Post Pruning.

In [8]:
```
from sklearn import tree
plt.figure(figsize=(10,10))
features = X
classes = ['0','1']
tree.plot_tree(model,feature_names=features,class_names=classes,filled=True)
plt.title('Afetr Pre Pruning')
plt.show()
```



Method 2: Post Pruning Example

- There are several post pruning techniques. Cost complexity pruning is one of the important among them.

Cost Complexity Pruning:

```
    - Decision trees can easily overfit. One way to avoid it is to l
imit the growth of trees by setting constrains. We can limit paramet
ers like max_depth , min_samples etc. But a most effective way is to
use post pruning methods like cost complexity pruning. This helps to
improve test accuracy and get a better model.

    - Cost complexity pruning is all about finding the right paramet
er for alpha.We will get the alpha values for this tree and will che
ck the accuracy with the pruned trees.
```

In [9]:
```
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
```

- cost_complexity_pruning_path function gives you two values one is alphas and another is impurity
- If you use alpha value you will be know how depth your tree is.
- ccp_alphas is going to find out the weak point with respect to the leaf node and it return a list of values

In [10]:
```
ccp_alphas
```

Out[10]:
```
array([0.        , 0.00226647, 0.00464743, 0.0046598 , 0.0056338 ,
       0.00704225, 0.00784194, 0.00911402, 0.01144366, 0.018988  ,
       0.02314163, 0.03422475, 0.32729844])
```

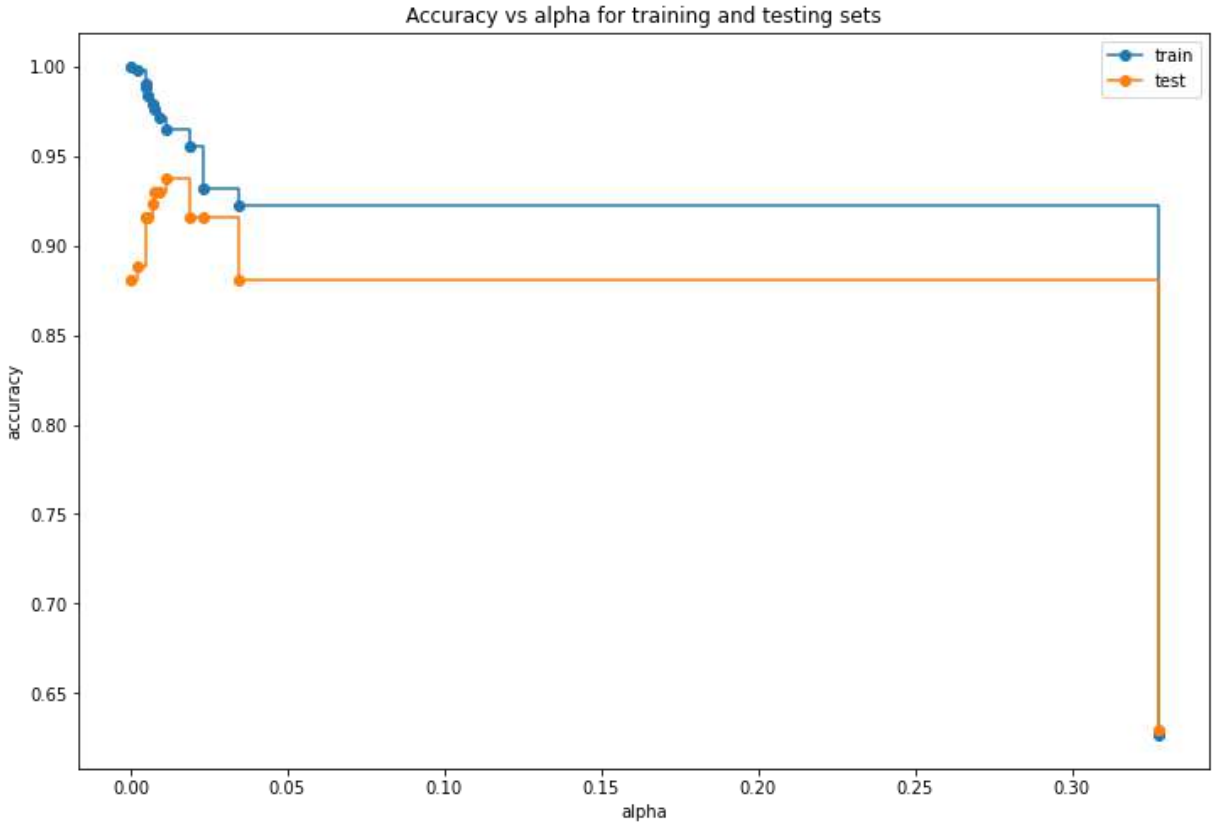
In [11]:
```
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)
print("Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
     clfs[-1].tree_.node_count, ccp_alphas[-1]))
```

```
Number of nodes in the last tree is: 1 with ccp_alpha: 0.3272984419327777
```

For the remainder of this example, we remove the last element in clfs and ccp_alphas, because it is the trivial tree with only one node. Here we show that the number of nodes and tree depth decreases as alpha increases.

Accuracy vs alpha for training and testing sets

In [12]:
```
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]


fig, ax = plt.subplots(figsize=(12,8))
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker='o', label="train",
        drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker='o', label="test",
        drawstyle="steps-post")
ax.legend()
plt.show()
```



- When ccp_alpha is set to zero and keeping the other default parameters of :class:DecisionTreeClassifier, the tree overfits, leading to a 100% training accuracy and 88% testing accuracy.
- As alpha increases, more of the tree is pruned, thus creating a decision tree that generalizes better.
- In this example, setting ccp_alpha=0.03 maximizes the testing accuracy. after that the accuracy is going to reduce.

In [13]:
```
clf = DecisionTreeClassifier(random_state=0, ccp_alpha=0.03)
clf.fit(X_train,y_train)
```

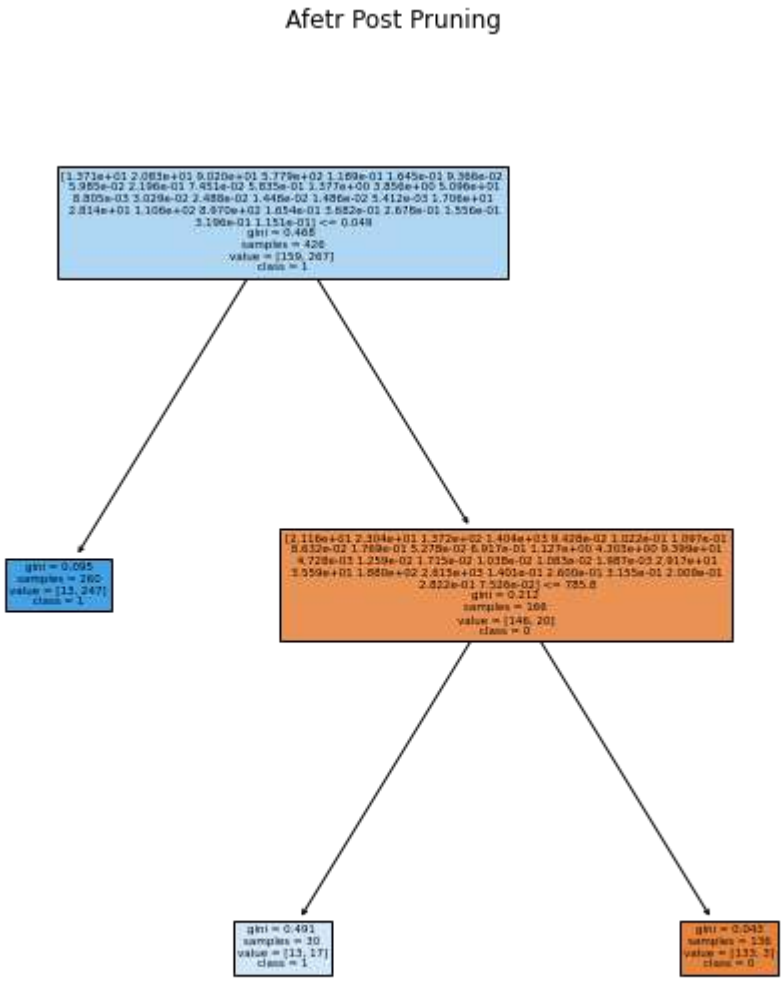
Out[13]:
```
DecisionTreeClassifier(ccp_alpha=0.03, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
```

In [14]:
```
pred=clf.predict(X_test)
from sklearn.metrics import accuracy_score
print("Training Accuracy :", clf.score(X_train, y_train))
print("Testing Accuracy :", accuracy_score(y_test,pred))
```

```
Training Accuracy : 0.931924882629108
Testing Accuracy : 0.916083916083916
```

- Now see the problem of overfit is gone and our model now perfect work.

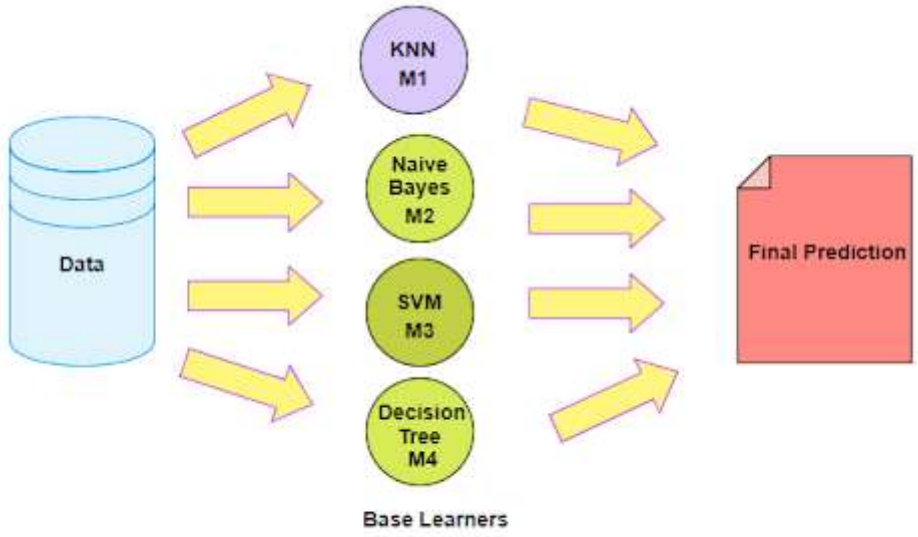
In [15]:
```
from sklearn import tree
plt.figure(figsize=(10,10))
features = X
classes = ['0','1']
tree.plot_tree(clf,feature_names=features,class_names=classes,filled=True)
plt.title('Afetr Post Pruning')
plt.show()
```



- Now you see that the size of decision tree significantly got reduced.
- postpruning is much efficient than prepruning. This all about Purning.

What is Ensemble Learning?

- Ensemble methods combine several trees base algorithms to construct better predictive performance than a single tree base algorithm.
- The main principle behind the ensemble model is that a group of weak learners come together to form a strong learner, thus increasing the accuracy of the model.
- When we try to predict the target variable using any machine learning technique, the main causes of difference in actual and predicted values are noise, variance, and bias. Ensemble helps to reduce these factors (except noise, which is irreducible error).



- As you see in the picture here it combaine 4 diffenet algorithm , and by this 4 algorithm it predict the output. Now you can imagine here it combaine diffenet algorithm and predict so it more powerful than single model.

Now Let's Dicuss About All Types Of Ensemble

Simple Ensemble

1. Max Voting
2. Averaging
3. Weighted Average

1. 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.

```
model1 = tree.DecisionTreeClassifier()
model2 = KNeighborsClassifier()
```

```
model3= LogisticRegression()

model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)

pred1=model1.predict(x_test)
pred2=model2.predict(x_test)
pred3=model3.predict(x_test)

final_pred = np.array([])
for i in range(0,len(x_test)):
    final_pred = np.append(final_pred, mode([pred1[i], pred2[i], pred3[i]]))
```

2. 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. Averaging can be used for making predictions in regression problems or while calculating probabilities for classification problems.

For example, in the below case, the averaging method would take the average of all the values.

i.e. (5+7+8+3+5+4)/6 = 5.33

```
model1 = tree.DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()

model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)

pred1=model1.predict_proba(x_test)
pred2=model2.predict_proba(x_test)
pred3=model3.predict_proba(x_test)

finalpred=(pred1+pred2+pred3)/3
```

3. 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.

The result is calculated as [(5*0.23) + (4*0.23) + (5*0.18) + (4*0.18) + (4*0.18)] = 4.41.

```
model1 = tree.DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()

model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)
```

```
pred1=model1.predict_proba(x_test)
pred2=model2.predict_proba(x_test)
pred3=model3.predict_proba(x_test)

finalpred=(pred1*0.3+pred2*0.3+pred3*0.4)
```

Advanced Ensemble techniques

- Now Dicuss Adance Ensemble Techniques
    1. Bagging
    2. Boosting
    3. stacking
    4. Blending
    5. cascadeing

1. Bagging

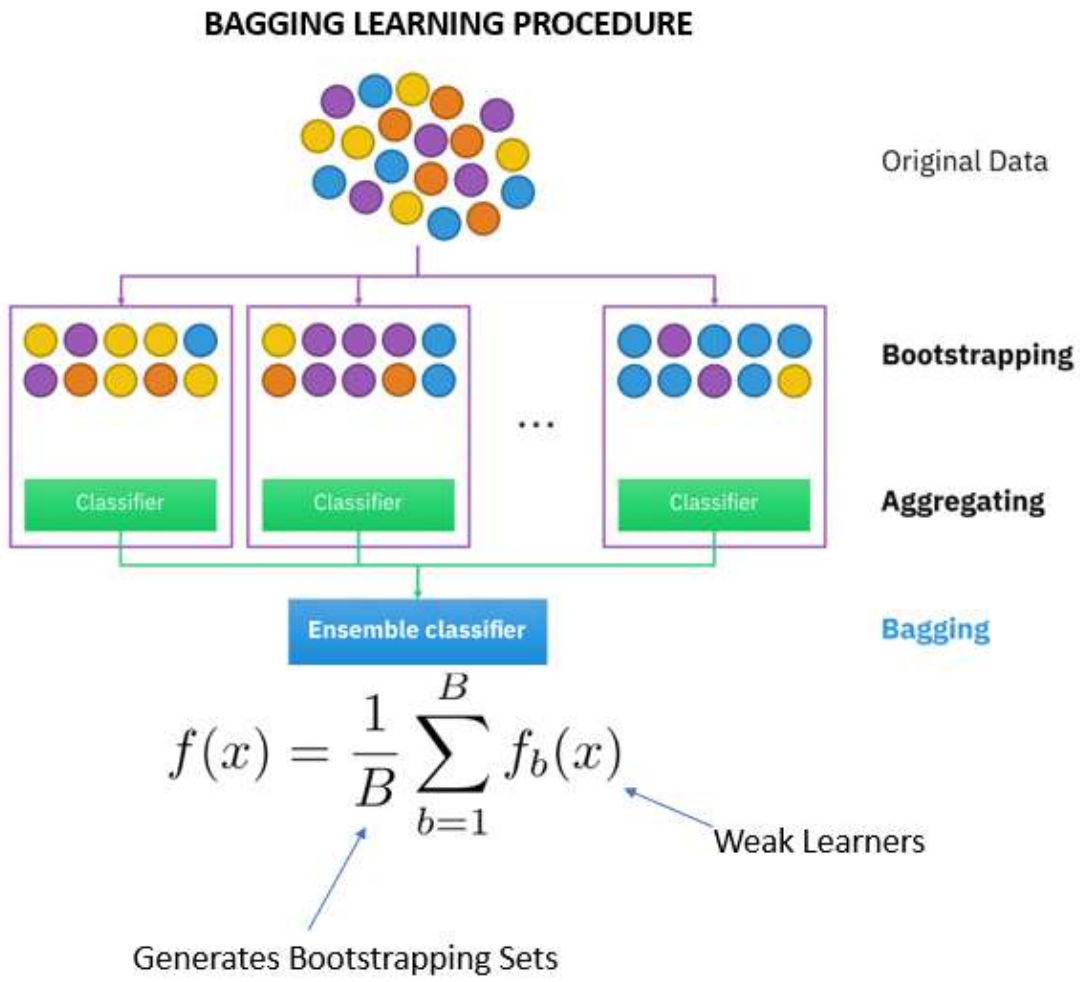
- It is also know as Bootstrapped Aggregation
- Bagging is mostly used to reduce the variance in a model. A simple example of bagging is the Random Forest algorithm.
- Bagging is one of the earliest, interesting and a very powerful ensemble algorithm.
- The core idea of bagging is to use bootstrapped replicas of the original dataset and use them to train different classifiers.
- Bagging (or Bootstrap Aggregating) technique uses these subsets (bags) to get a fair idea of the distribution (complete set). The size of subsets created for bagging may be less than the original set.

How Bagging works on training dataset ?

Since Bagging resamples the original training dataset with replacement, some instance(or data) may be present multiple times while others are left out.

```
Original training dataset: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Resampled training set 1: 2, 3, 3, 5, 6, 1, 8, 10, 9, 1
Resampled training set 2: 1, 1, 5, 6, 3, 8, 9, 10, 2, 7
Resampled training set 3: 1, 5, 8, 9, 2, 10, 9, 7, 5, 4
```

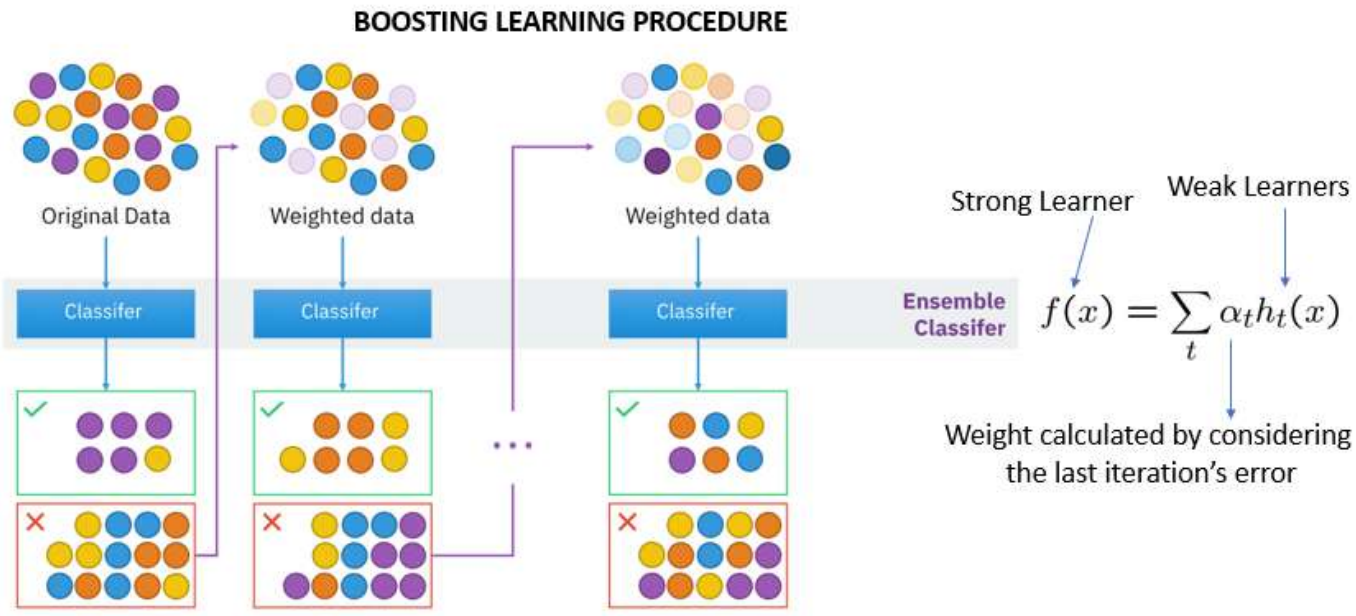
Its Process Flow:



Eg. Random Forest Classifier

2. Boosting

- The second ensemble technique that we are going to discuss today is called Boosting.
- Boosting is used to convert weak base learners to strong ones. Weak learners generally have a very weak correlation with the true class labels and strong learners have a very high correlation between the model and the true class labels.
- AdaBoost was the first really successful boosting algorithm developed for the purpose of binary classification. AdaBoost is short for Adaptive Boosting and is a very popular boosting technique which combines multiple “weak classifiers” into a single “strong classifier”.



Eg. AdaBoost ,GBM,XGBM,Light GBM,CatBoost

Similarities Between Bagging and Boosting –

- Both are ensemble methods to get N learners from 1 learner.
- Both generate several training data sets by random sampling.
- Both make the final decision by averaging the N learners (or taking the majority of them i.e Majority Voting).
- Both are good at reducing variance and provide higher stability.

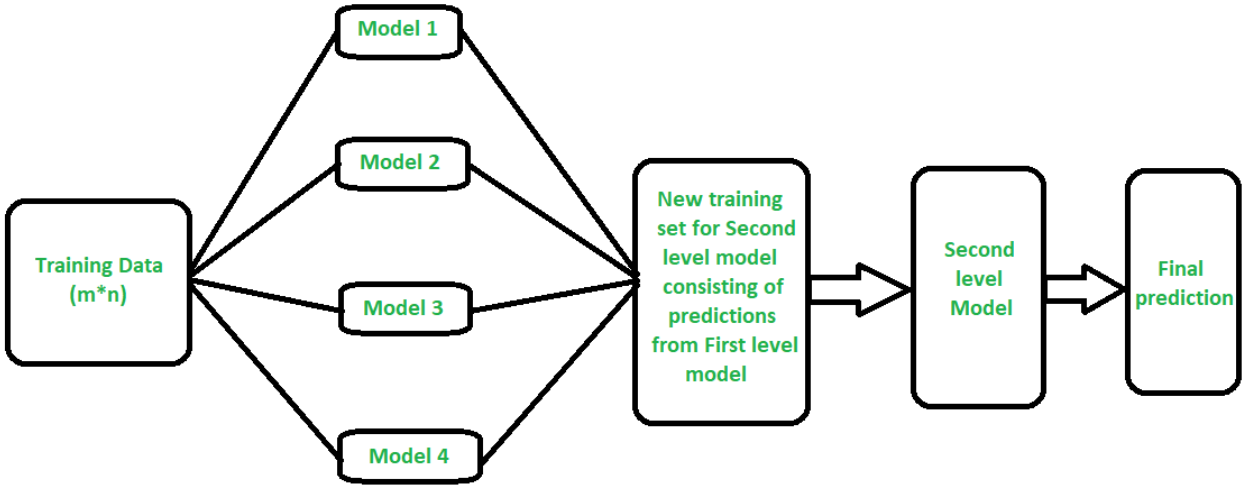
Where You Use bagging And Boosting

- If the difficulty of the single model is over-fitting, then Bagging is the best option.
- If the problem is that the single model gets a very low performance, Boosting could generate a combined model with lower errors as it optimises the advantages and reduces pitfalls of the single model.

3. Stacking

- Stacking is a way to ensemble multiple classifications or regression model.
- Stacking is an ensemble learning technique which is used to combine the predictions of diverse classification models into one single model also known as the meta-classifier.
- Stacking (sometimes called Stacked Generalization) is a different paradigm. The point of stacking is to explore a space of different models for the same problem.
- The idea is that you can attack a learning problem with different types of models which are capable to learn some part of the problem, but not the whole space of the problem. So, you can build multiple different learners and you use them to build an intermediate prediction, one prediction for each learned model.
- Then you add a new model which learns from the intermediate predictions the same target.
- This final model is said to be stacked on the top of the others, hence the name. Thus, you might improve your overall performance, and often you end up with a model which is better than any individual intermediate model. Notice however, that it does not give you any guarantee, as is often the case with any machine learning technique.

Stacking Procedure



How stacking works?

- We split the training data into K-folds just like K-fold cross-validation.
- A base model is fitted on the K-1 parts and predictions are made for Kth part.
- We do for each part of the training data.
- The base model is then fitted on the whole train data set to calculate its performance on the test set.
- We repeat the last 3 steps for other base models.
- Predictions from the train set are used as features for the second level model.
- Second level model is used to make a prediction on the test set.

Eg. Voting Classifier

4.BeInding

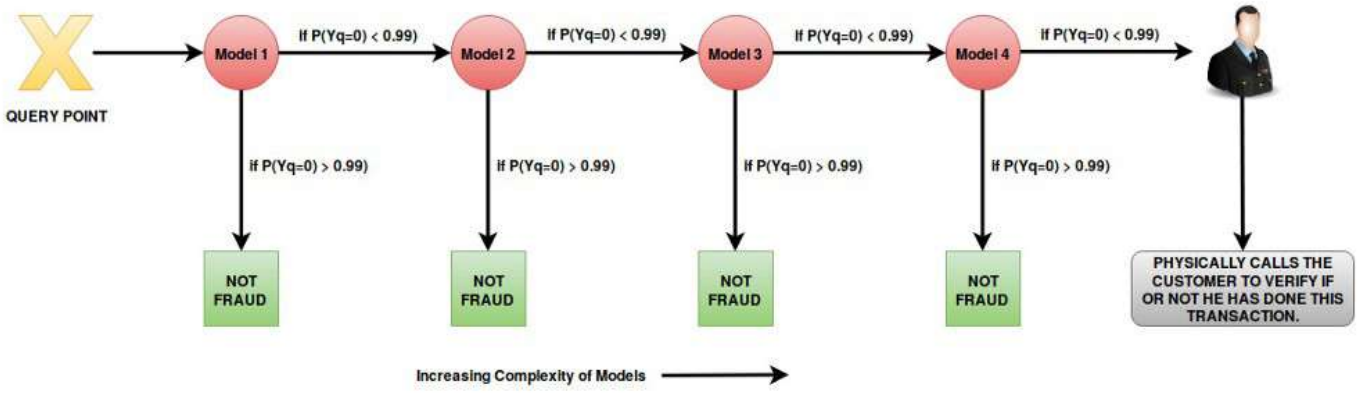
Blending is a similar approach to stacking.

- Blending is technique where we can do weighted averaging of final result.
- Blending is also an ensemble technique that can help us to improve performance and increase accuracy. It follows the same approach as stacking but uses only a holdout (validation) set from the train set to make predictions.
- Here is a detailed explanation of the blending process:
  - The train set is split into training and validation sets.
  - We train the base models on the training set.
  - We make predictions only on the validation set and the test set.
  - The validation predictions are used as features to build a new model.
  - This model is used to make final predictions on the test set using the prediction values as features.
- The difference between stacking and blending is that Stacking uses out-of-fold predictions for the train set of the next layer (i.e meta-model), and Blending uses a validation set (let's say, 10-15% of the training set) to train the next layer.

4. Cascading

- Cascading is one of the most powerful ensemble learning algorithm which is used by Machine Learning engineers and scientists when they want to be absolutely dead sure about the accuracy of a result.

- For example, suppose we want to build a machine learning model which would detect if a credit card transaction is fraudulent or not.
- If you think about it, it's a binary classification problem where a class label 0 means the transaction is not fraud & a class label 1 means the transaction is fraudulent.
- In such a model, it's very risky to put our faith completely on just one model. So what we do is build a sequence of models (or a cascade of models) to be absolutely sure about the fact that the transaction is not fraudulent.
- Cascade models are mostly used when the cost of making a mistake is very very high. I will try to explain cascading with the help of a simple diagram.



DIFFERENT STAGES OF QUERING CASCADE CLASSIFIERS IN A FOUR MODEL CASCADE SYSTEM

Look at the above diagram. Given that we have a transaction query point Xq, we will feed it to Model 1. Model 1 can be anything a random forest, or a logistic regression model or maybe a support vector machine. It can be anything! Basically what Model 1 does is that it predicts class probabilities to determine to which class do a given query point has higher chances of belonging to.

Let's say class label 1 means the transaction is fraudulent, and class label 0 means the transaction is not fraud. Typically, the predicted probabilities is given by this — P(Yq=0) and P(Yq=1), where Yq is our actual class label. Now let's assume that P(Yq=0), i.e. the probability of the transaction to be not fraudulent is very high. If you think carefully, if P(Yq=0) is extremely high, we will say that the transaction is not fraud. Let's assume we have set a threshold of 99%. It means if and only if P(Yq=0) > 0.99, we will declare the final prediction to be not fraudulent. However, if P(Yq=0) < 0.99 we are not very sure if or not it's a fraudulent transaction although though there is a high chance that the transaction is not fraudulent. In such a case, when P(Yq=0) < 0.99, we want to be really really sure that the transaction is not fraudulent. We need to be absolutely careful because if our model fails to detect a fraudulent transaction we might lose millions of dollars! So even when we are slightly unsure, we will train another Model 2. Model 2 does the same thing, it receives the query point and predicts P(Yq=0). Just like in stage 1, if P(Yq=0) > 0.99, we will declare the transaction to be not fraudulent and terminate the loop. But again if we get P(Yq=0) < 0.99, we aren' sure! Hence, we will pass the query point to another Model 3 in the cascade which does the same thing.

In a typical cascading system the complexity of models increases as we add more and more models to the cascade. Please note that all the models in a cascade are super powerful and has a very high accuracy on unseen data. However, it might happen that none of the models can give us a value of P(Yq=0) > 0.99. In such a case, typically there is a human being who sits at the end of a cascade. This person will personally call the customer and ask him whether or not he has done the transaction. Now, we are absolutely certain that the transaction is not a fraud one when the customer says that he is the one who has done the transaction.

@@ Credit:- https://medium.com/@saugata.paul1010/ensemble-learning-bagging-boosting-stacking-and-cascading-classifiers-in-machine-learning-9c66cb271674 (https://medium.com/@saugata.paul1010/ensemble-learning-bagging-boosting-stacking-and-cascading-classifiers-in-machine-learning-9c66cb271674)

Random Forest

1. What is Random Forest algorithm?

- It is a bagging or bootstraping Technique. For understanding bagging see my preious notebook.
- First, Random Forest algorithm is a supervised classification algorithm. We can see it from its name, which is to create a forest by some way and make it random.
- There is a direct relationship between the number of trees in the forest and the results it can get: the larger the number of trees, the more accurate the result.
- The difference between Random Forest algorithm and the decision tree algorithm is that in Random Forest, the process es of finding the root node and splitting the feature nodes will run randomly.

2. Why Random Forest algorithm?

1. Overfitting is one critical problem that may make the results worse, but for Random Forest algorithm, if there are enough trees in the forest, the classifier won't overfit the model.
2. Random Forest can handle missing values, and the last advantage is that the Random Forest classifier can be modeled for categorical values.
3. No nedd for feature scalling in random forest.

3. How Random Forest algorithm works?

- There are two step
    1. Creation Of Random Forest
    2. Prediction Of Random Forest

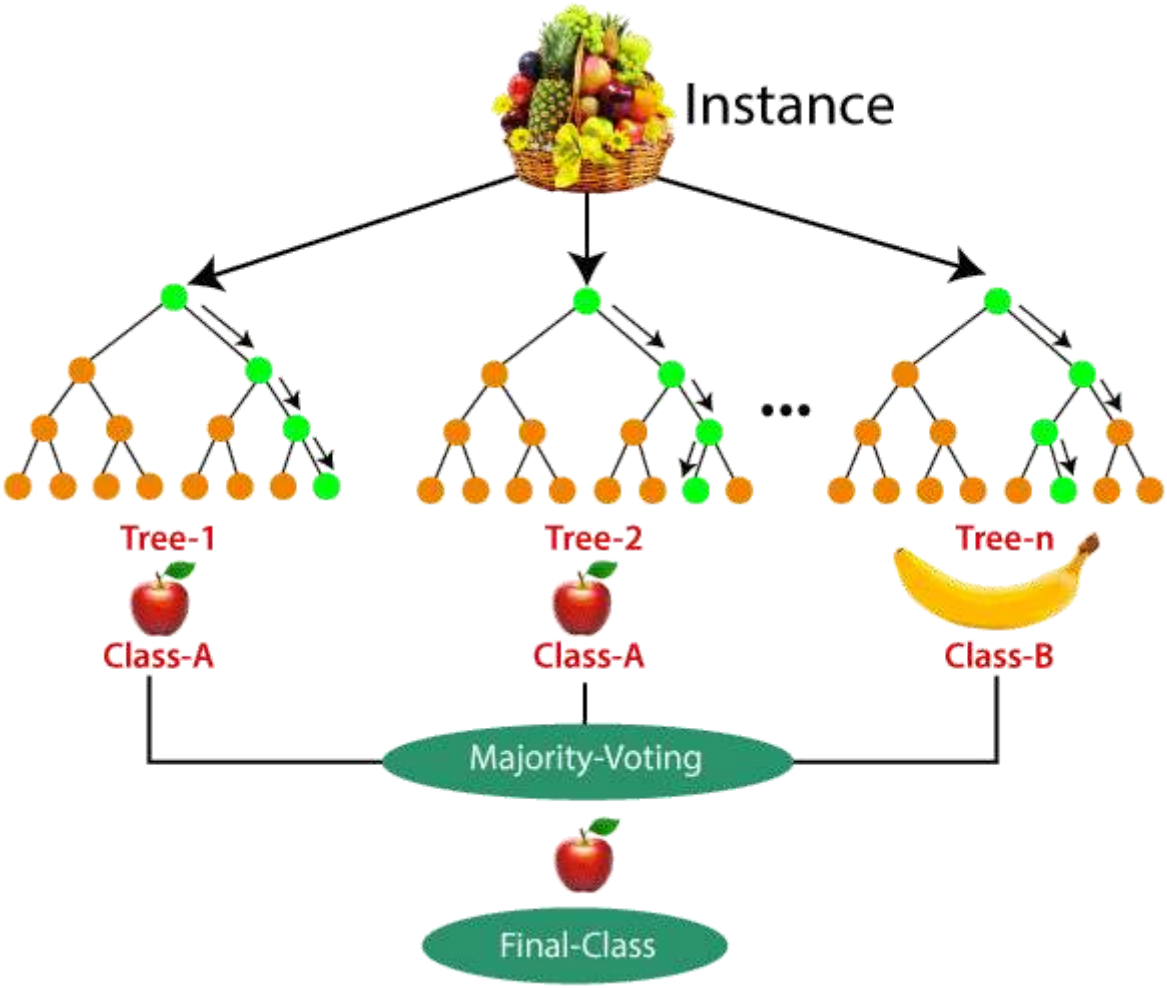
- 1. Created Random Forest

- Randomly select "K" features from total "m" features where k << m
- Among the "K" features, calculate the node "d" using the best split point
- Split the node into daughter nodes using the best split
- Repeat the a to c steps until "l" number of nodes has been reached
- Build forest by repeating steps a to d for "n" number times to create "n" number of trees

- 2. Prediction Random Forest

- Takes the test features and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome (target)
- Calculate the votes for each predicted target
- Consider the high voted predicted target as the final prediction from the random forest algorithm

See the Diagram:



- Here Main thing is that it apply majority vote on dataset , as you see above have 2apple and one banana, so to apply majority vote apply wins, so it predict apple.

Before Going Randomforest Implementation I am shown you how bagging Works:

In [1]:

```
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [2]:

```
from sklearn.datasets import load_wine
dataset = load_wine()
X = dataset.data
y = dataset.target
```

In [3]:

```
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=3)
```

In [4]:

```
#Only Knn classifier

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
knn.score(X_test, y_test)
```

Out[4]:

```
0.6388888888888888
```

In [5]:

```
#bagging over our KNN classifier
bag_knn = BaggingClassifier(KNeighborsClassifier(n_neighbors=5),
                            n_estimators=10, max_samples=0.5,
                            bootstrap=True, random_state=3,oob_score=True)
bag_knn.fit(X_train, y_train)
bag_knn.score(X_test, y_test)
```

Out[5]:

```
0.6944444444444444
```

- See here if i use only Knn classifier then the accuracy 63.88 , But when i use bagging over our KNN classifier and see our score improves to 69.44. So now you think how powerful bagging is?

Now Implement Randomforest

Importing the libraries

In [6]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

In [7]:

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

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 = 0.25, random_state =
```

Feature Scaling

In [9]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Training the Random Forest Classification model on the Training set

In [10]:

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state
classifier.fit(X_train, y_train)
```

Out[10]:

```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

Predicting the Test set results

In [11]:

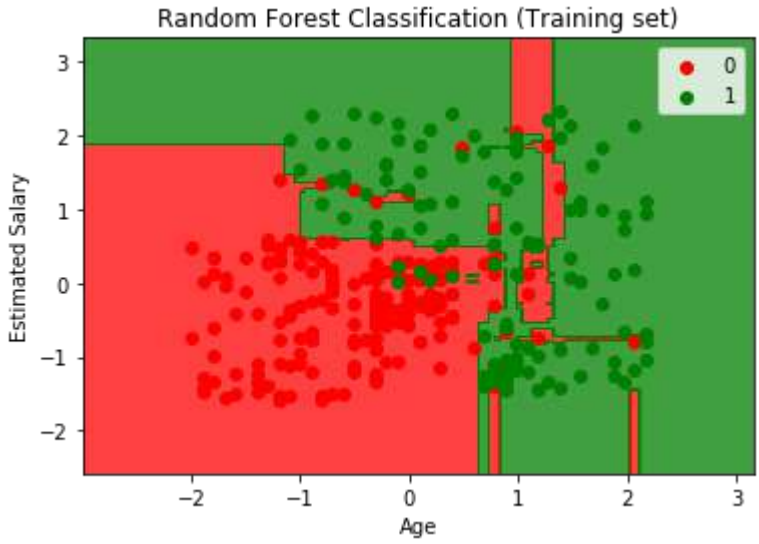
```
y_pred = classifier.predict(X_test)
```

Visualising the Training set results

In [12]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.sh
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Random Forest Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.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 a 2-D array with a single row if you really want
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 a 2-D array with a single row if you really want
to specify the same RGB or RGBA value for all points.
```

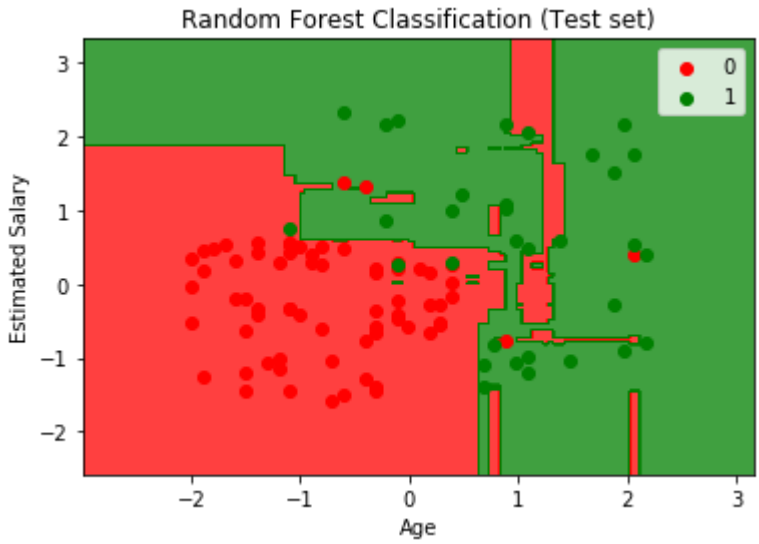


Visualising the Test set results

In [13]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.sh
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Random Forest Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.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 a 2-D array with a single row if you really want
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 a 2-D array with a single row if you really want
to specify the same RGB or RGBA value for all points.
```



Pandas Profiling in Python

- To Know More About - https://pandas-profiling.github.io/pandas-profiling/docs/master/rtd/index.html (https://pandas-profiling.github.io/pandas-profiling/docs/master/rtd/index.html)
- When importing a new data set for the very first time, the first thing to do is to get an understanding of the data. This includes steps like determining the range of specific predictors, identifying each predictor's data type, as well as computing the number or percentage of missing values for each predictor.
- The pandas_profiling library in Python include a method named as ProfileReport() which generate a basic report on the input DataFrame.

The report consist of the following:

1. Overview
2. Variables
3. Interactions
4. Co-relations
5. Missing Values
6. Sample

1. Overview Section :

- This section provides overall data set information. Dataset statistics and Variable types.
- Dataset statistics display columns, rows, missing values, etc.
- Variable Types shows data types of the attributes of the data set.
- It also shows "Warnings", where it shows which feature(s) are highly correlated to others.

2. Variables :

- This section provides information about every feature individually in detail.
- When we click on the Toggle details option as shown in the above image, the new section shows up.

3. Interactions :

- This section shows statistics, histograms, common values, and extreme values of features.
- here we intract various feature each other.

4. Co-relations :

- This Section shows how features are co-related with each other with the help of Seaborn's Heatmap.
- We can easily toggle between the different types of correlations like Pearson, Spearman, Kendall, and phik.

5. Missing Values :

- here we can see whether our dataset contain missing value or not.
- here it use 4 different function ex- Count,matrix,heatmap,dendrogram

6. Sample :

- This section displays the First 10 Rows and the Last 10 rows of the dataset.

Installation of Pandas Profiling:

Installation with the pip package

```
!pip install pandas-profiling
```

Installation with the conda package

```
conda install -c conda-forge pandas-profiling
```

Implement Using Python

In [2]:

```
import pandas as pd
df=pd.read_csv("https://raw.githubusercontent.com/agconti/kaggle-titanic/master/data/train.
df.head()
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	C
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	

In [7]:

```
import pandas_profiling as pp
# run the profile report
profile = df.profile_report(title='Pandas Profiling Report')
profile
```

```
Summarize dataset: 100%                    26/26 [00:11<00:00, 2.20it/s, Completed]

Generate report structure: 100%            1/1 [00:06<00:00, 6.68s/it]

Render HTML: 100%                          1/1 [02:32<00:00, 152.30s/it]
```

In [8]:

```
# save the report as html file
### It generate a sepaarte html file
profile.to_file(output_file="pandas_profiling1.html")
```

```
Export report to file: 100%                1/1 [01:32<00:00, 92.03s/it]
```

In [ ]:

```
# save the report as json file
profile.to_file(output_file="pandas_profiling2.json")
```

What is Data Preprocessing

- Data Contain text,image,video,time,audio.
- When we take raw data there will be some problem or error so to overcome these we perform data preprocess.
- It is a process that convert raw data into meaningful data using differnet techinque.
- Data In real world may consit of incomplete,noise,incosistant,duplicate it affect our model accuracy

Different Step For Data Preprocessing

```
 1. Data Cleaning - It means fill missing value,smooth out the noise while identif
y the outlier, and correct the data.
 2. Data Integration - Here we merge data from differnet source in to a coherent d
ata store such as data warehouse.
 3. Data Reduction -It is a technique in which we can reduce the data size by aggr
egating,elimanating redundant feature.
 4. Data Transformation - Data are transformed into appropriate forms for ml mode
l.here we use normalization technique.
 5. Data Discretization - in this techinque transforms numeric data by mapping val
ues to interval or concept labels.
```

1. Handling Missing Value

- We can classify the missing values in different types. Each type of missing value require slightly different handling. The main types are —

```
1. Missing completely at Random (MCAR) : As the name suggests missing completely a
t random means that there’s no relationship between whether a data point is missin
g and any values in the data set, missing or observed. The missing data is just a
 random subset of the data.


2. Missing at Random (MAR) : Missing at random means that the propensity of missin
g values has a systematic relationship with the observed data but not the missing
 data.


3. Missing Not at Random (MNAR) : Missing not at random means that there is a dist
inct relationship between the propensity of a value to be missing and its values.
```

Techniques of dealing with missing data

1. Drop missing values/columns/rows
2. Imputation

Dropping Missing data

- The simplest way to drop the columns/rows for which the data is not available.
- In the first two types of missing data, MCAR and MAR, in general, it is safe to remove the data with missing values depending upon their occurrences.
- but in the third case (MNAR) removing observations with missing values can produce a bias in the model.

- Lets take a dataset and see how to handle missing values in it.

In [1]:

```
import pandas as pd
import numpy as np
data = pd.read_csv('Melbourne_housing_FULL.csv')
data.head(5)
```

Out[1]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postc
0	Abbotsford	68 Studley St	2	h	NaN	SS	Jellis	3/09/2016	2.5	30
1	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5	30
2	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5	30
3	Abbotsford	18/659 Victoria St	3	u	NaN	VB	Rounds	4/02/2016	2.5	30
4	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5	30

5 rows × 21 columns

Now 1st step to check wheather this data set contain missing value or not.

In [2]:

```
data_temp = data.copy()
```

In [3]:

```
data_temp.isnull().sum()
```

Out[3]:

```
Suburb              0
Address             0
Rooms               0
Type                0
Price            7610
Method              0
SellerG             0
Date                0
Distance            1
Postcode            1
Bedroom2         8217
Bathroom         8226
Car              8728
Landsize        11810
BuildingArea    21115
YearBuilt       19306
CouncilArea         3
Lattitude        7976
Longtitude       7976
Regionname          3
Propertycount       3
dtype: int64
```

Now you can see there are huge number of missing value now apply 1st method that drop the missing value

In [4]:

```
#dropna function in Pandas removes all the rows with missing values
data_temp.dropna(inplace=True)
#Putting axis=1 removes the columns with missing values
data_temp.dropna(inplace=True, axis=0)
```

In [5]:

```
data_temp.isnull().sum()
```

Out[5]:

```
Suburb           0
Address          0
Rooms            0
Type             0
Price            0
Method           0
SellerG          0
Date             0
Distance         0
Postcode         0
Bedroom2         0
Bathroom         0
Car              0
Landsize         0
BuildingArea     0
YearBuilt        0
CouncilArea      0
Lattitude        0
Longtitude       0
Regionname       0
Propertycount    0
dtype: int64
```

In [6]:

```
print("Actual data :",data.shape)
print("after drop na value :",data_temp.shape)
```

```
Actual data : (34857, 21)
after drop na value : (8887, 21)
```

- If we drop the rows our total number of data points to train our model will go down which can reduce the model performance.
- Do this only if you have large number of training examples and the rows with missing data are not very high in number. Dropping the column altogether will remove a feature from our model i.e the model predictions will be independent of the building area.
- So dropping the misssing value is not good for any model as the number of data reduce so Lets look at a better approach for dealing with missing data.

Imputation

- Imputation means to replace or fill the missing data with some value.
- There are lot of ways to impute the data.

    1. A constant value that belongs to the set of possible values of that variable, such as 0, distinct from all other values
    2. A mean, median or mode value for the column

Lets go back to our dataset and see how each of these methods work

1st approach we can just put zero value in place of missing value.

impute with 0

In [7]:

```
data_temp2 = data.copy()
```

In [8]:

```
data_temp2.isnull().sum()
```

Out[8]:

```
Suburb              0
Address             0
Rooms               0
Type                0
Price            7610
Method              0
SellerG             0
Date                0
Distance            1
Postcode            1
Bedroom2         8217
Bathroom         8226
Car              8728
Landsize        11810
BuildingArea    21115
YearBuilt       19306
CouncilArea         3
Lattitude        7976
Longtitude       7976
Regionname          3
Propertycount       3
dtype: int64
```

In [9]:

```
data_temp2.fillna(0, inplace=True)
```

see here all missing value replaced by zero. but it also not a good approach for better model so now we see how we impute mean ,median and mode

In [10]:

```
data_temp2.isnull().sum()
```

Out[10]:

```
Suburb           0
Address          0
Rooms            0
Type             0
Price            0
Method           0
SellerG          0
Date             0
Distance         0
Postcode         0
Bedroom2         0
Bathroom         0
Car              0
Landsize         0
BuildingArea     0
YearBuilt        0
CouncilArea      0
Lattitude        0
Longtitude       0
Regionname       0
Propertycount    0
dtype: int64
```

Use Mean,Median,Mode

In [11]:

```
data_temp3 = data.copy()
```

In [12]:

```
data_temp3.isnull().sum()
```

Out[12]:

```
Suburb              0
Address             0
Rooms               0
Type                0
Price            7610
Method              0
SellerG             0
Date                0
Distance            1
Postcode            1
Bedroom2         8217
Bathroom         8226
Car              8728
Landsize        11810
BuildingArea    21115
YearBuilt       19306
CouncilArea         3
Lattitude        7976
Longtitude       7976
Regionname          3
Propertycount       3
dtype: int64
```

Imp Point to remember

- we all know basically three type of data int,float and object.
- so when we impute missing value in (int and float) we can use all mean,median and mode, but you can always use mean and median, but if the data set contain outlier then you definitly use median as mean affect by outlier. so better in int and float you must use meadin.
- Now in object you need to use mode only other wise it throws error. when you use mode keep in mind you should use [0] after mode so that the exact data can be extract.

Summerize

- int,float - use Mean(), Median()
- object - mode()[0]

Here you impute one by one means in int,float you impute mean,median and in object mode.

In [13]:

```
# object data i use mode()[0]
print("Missing Value before use imputation : ",data_temp3.CouncilArea.isnull().sum().sum())

data_temp3['CouncilArea'] = data_temp3['CouncilArea'].fillna(data_temp3['CouncilArea'].mode

print("Missing Value after use imputation : ",data_temp3.CouncilArea.isnull().sum().sum())
```

```
Missing Value before use imputation :  3
Missing Value after use imputation :  0
```

In [14]:

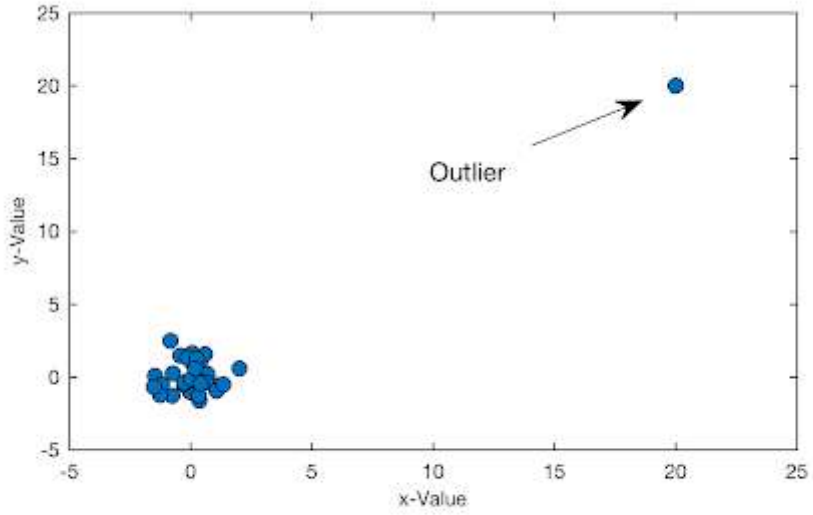
```
# float data i use median()
print("Missing Value before use imputation : ",data_temp3.Longtitude.isnull().sum().sum())

data_temp3['Longtitude'] = data_temp3['Longtitude'].fillna(data_temp3['Longtitude'].median(

print("Missing Value before use imputation : ",data_temp3.Longtitude.isnull().sum().sum())
```

```
Missing Value before use imputation :  7976
Missing Value before use imputation :  0
```

in this way you can fixed missing value in your data. its just basic technique there are other also available

Outlier

- Outliers are extreme values that deviate from other observations on data , they may indicate a variability in a measurement, experimental errors or a novelty. In other words, an outlier is an observation that diverges from an overall pattern on a sample.



Nature of Outliers :

1. Genuine extreme high and low values in the dataset
2. Introduced due to human or mechanical error
3. Introduced by replacing missing values

Types of outliers :

Outliers can be of two kinds:

1. Univariate - can be found when looking at a distribution of values in a single feature space.
2. Multivariate - can be found in a n-dimensional space (of n-features).

Most common causes of outliers on a data set :

- Data entry errors (human errors)
- Measurement errors (instrument errors)
- Experimental errors (data extraction or experiment planning/executing errors)
- Intentional (dummy outliers made to test detection methods)
- Data processing errors (data manipulation or data set unintended mutations)
- Sampling errors (extracting or mixing data from wrong or various sources)
- Natural (not an error, novelties in data)

Outliers can also come in different variate :

```
- Point Outlier -  single data points that lay far from the rest of the distributi
on
- collective outliers - can be noise in data, such as punctuation symbols when rea
lizing text analysis or background noise signal when doing speech recognition.
- contextual outliers -  subsets of novelties in data such as a signal that may in
dicate the discovery of new phenomena
```

Detection And Prevention

- There are many method for do this function but here i show widly use method.

Outlier Detection

- Boxplot
- Scatterplot
- Z_score

Outlier Prevention

- Z-score method
- IQR method
- Log transform

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.simplefilter('ignore')
```

In [2]:

```
from sklearn.datasets import load_boston
boston = load_boston()
df = pd.DataFrame(boston.data,columns=boston.feature_names)
df
```

Out[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90
...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90

506 rows × 13 columns

Outlier Detection

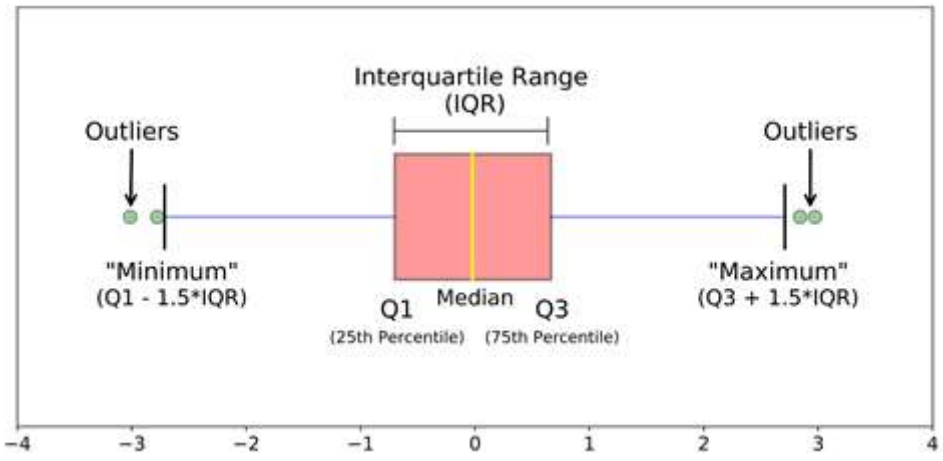
1. Boxplot

2. Scatter Plot

3. Z_score

1. BoxPlot

- The very purpose of box plots is to identify outliers in the data series before making any further analysis so that the conclusion made from the study gives more accurate results not influenced by any extremes or abnormal values.



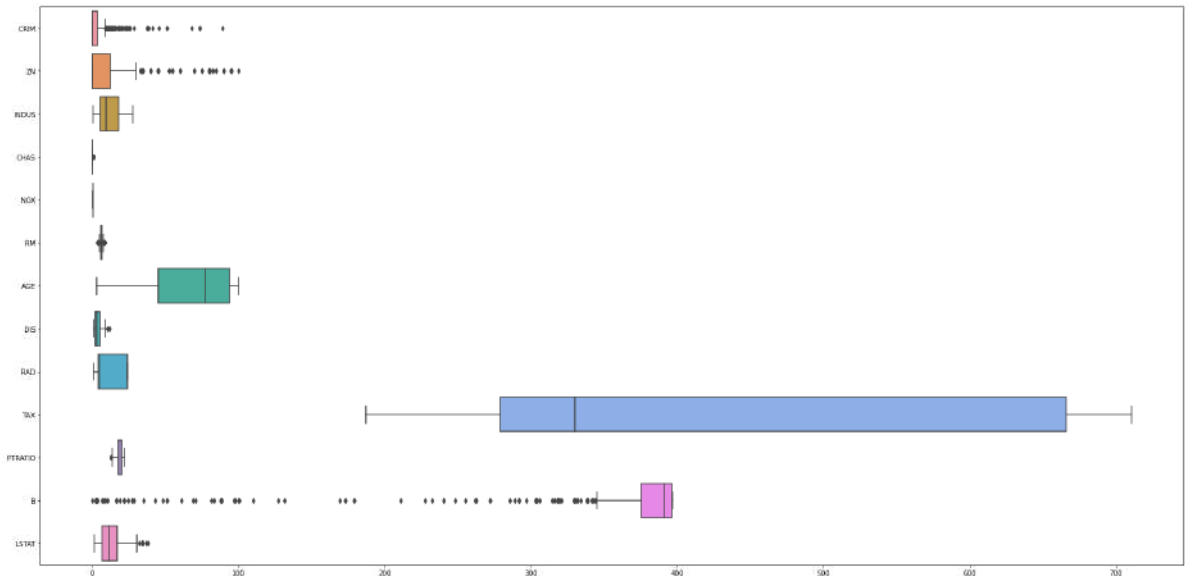
Implementation Boxplot

In [3]:

```
plt.figure(figsize=(30,15))
sns.boxplot(data=df, orient="h")
```

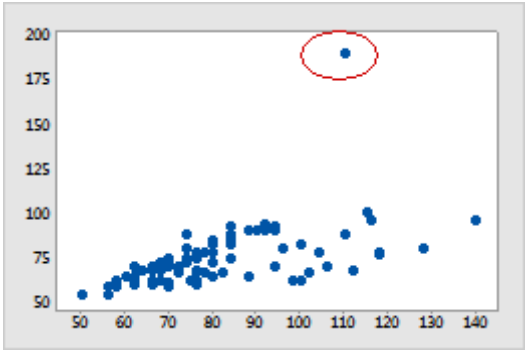
Out[3]:

<AxesSubplot:>



2. Scatter Plot

- Here we must put 2 variable as it is bivarient plot

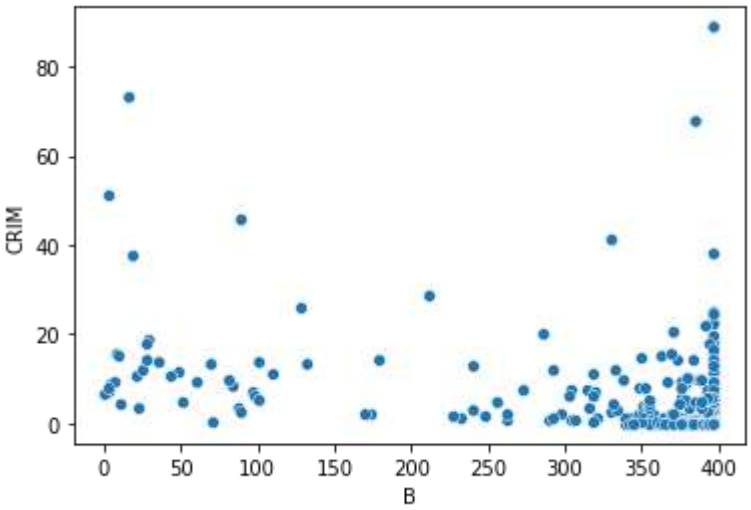


In [4]:

```
sns.scatterplot(df['B'],df['CRIM'])
```

Out[4]:

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



3. Z_Score

- Z score is an important concept in statistics. Z score is also called standard score. This score helps to understand if a data value is greater or smaller than mean and how far away it is from the mean. More specifically, Z score tells how many standard deviations away a data point is from the mean.

$$Z = \frac{X - \mu}{\sigma}$$

Z → Standard (Normal) or Z score

X → member element of group

μ → mean of expectation

σ → standard deviation

Implement

In [5]:

```
from scipy import stats
zscore = np.abs(stats.zscore(df))
print(zscore)
```

```
[[0.41978194 0.28482986 1.2879095  ... 1.45900038 0.44105193 1.0755623 ]
 [0.41733926 0.48772236 0.59338101 ... 0.30309415 0.44105193 0.49243937]
 [0.41734159 0.48772236 0.59338101 ... 0.30309415 0.39642699 1.2087274 ]
 ...
 [0.41344658 0.48772236 0.11573841 ... 1.17646583 0.44105193 0.98304761]
 [0.40776407 0.48772236 0.11573841 ... 1.17646583 0.4032249  0.86530163]
 [0.41500016 0.48772236 0.11573841 ... 1.17646583 0.44105193 0.66905833]]
```

- By seeing these value we can't say where outlier exist so here we need to define a threshload.
- Let take Threshold > 3

In [6]:

```
# here we fliter out all the zscore value which are greater than 3 so these are outlier
threshold = 3
print(np.where(zscore > 3))
```

```
(array([ 55,  56,  57, 102, 141, 142, 152, 154, 155, 160, 162, 163, 199,
       200, 201, 202, 203, 204, 208, 209, 210, 211, 212, 216, 218, 219,
       220, 221, 222, 225, 234, 236, 256, 257, 262, 269, 273, 274, 276,
       277, 282, 283, 283, 284, 347, 351, 352, 353, 353, 354, 355, 356,
       357, 358, 363, 364, 364, 365, 367, 369, 370, 372, 373, 374, 374,
       380, 398, 404, 405, 406, 410, 410, 411, 412, 412, 414, 414, 415,
       416, 418, 418, 419, 423, 424, 425, 426, 427, 427, 429, 431, 436,
       437, 438, 445, 450, 454, 455, 456, 457, 466], dtype=int64), array([
1,  1,  1, 11, 12,  3,  3,  3,  3,  3,  3,  3,  1,  1,  1,  1,  1,
        1,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  5,  3,  3,  1,  5,
        5,  3,  3,  3,  3,  3,  3,  1,  3,  1,  1,  7,  7,  1,  7,  7,  7,
        3,  3,  3,  3,  3,  5,  5,  5,  3,  3,  3, 12,  5, 12,  0,  0,  0,
        0,  5,  0, 11, 11, 11, 12,  0, 12, 11, 11,  0, 11, 11, 11, 11, 11,
       11,  0, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11],
      dtype=int64))
```

- you see there are two array , the first array contains the list of row numbers and second array respective column numbers.
- These all are outlier

Outlier Prevention :

1. Drop Using Z_score Threshold

2. Using IQR

3. Log Transform

Method 1 : Drop

1st Drop the value which are greater than the threshlod of z_score

In [7]:

```
df_clean=df
df_clean = df_clean[(zscore<3).all(axis=1)]
```

In [8]:

```
df.shape,df_clean.shape
```

Out[8]:

```
((506, 13), (415, 13))
```

See here i drop the outlier which are greater than 3 so the data value reduce 506 to 415

Method 2 : Use IQR

- The interquartile range IQR tells us the range .where the bulk of the values lie. The interquartile range is calculated by subtracting the first quartile from the third quartile. IQR = Q3 - Q1

In [9]:

```
df_iqr = df
Q1 = df_iqr.quantile(0.25)
Q3 = df_iqr.quantile(0.75)
IQR = Q3-Q1
```

In [10]:

```
df_iqr = df_iqr[~(df_iqr < (Q1-1.5*IQR)) | (df_iqr > (Q3+1.5*IQR))]
```

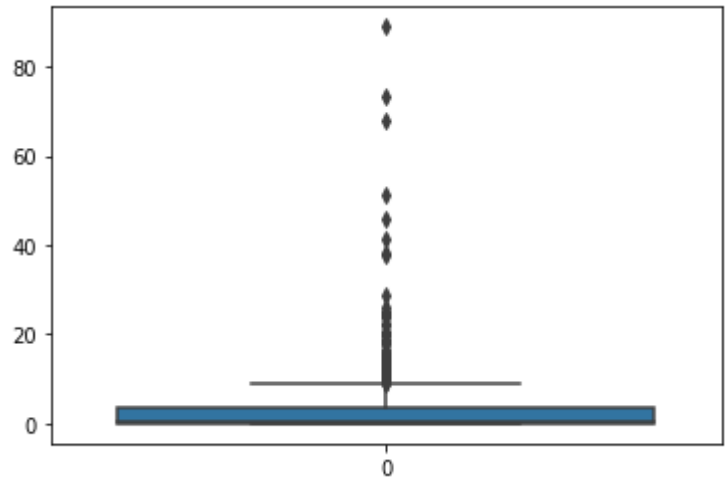
Method 3 :Log Transform

In [11]:

```
## before log transform
sns.boxplot(data=df['CRIM'])
```

Out[11]:

<AxesSubplot:>

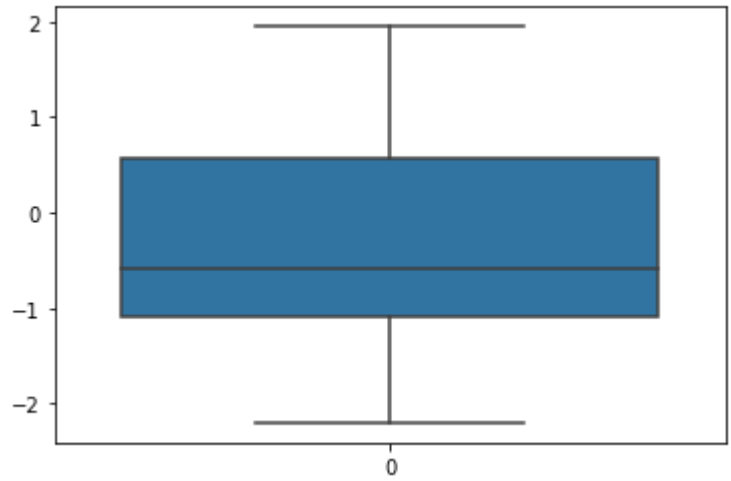


In [12]:

```
# after log transform
sns.boxplot(data=np.log10(df['CRIM']))
```

Out[12]:

<AxesSubplot:>



see after log transform all outlier gone.

Categorical Variable

- We all know our machine learning algorithm not properly work on categorical data (object), so we need to convert categorical to numerical variable.
- Before converting we 1st know types of categorical variable.

There are two types of Categorical Variable

```
1. Ordinal - These  variable are meaningfully orderd
2. Nominal - Here no intrensic order of the label
```

- Example of Ordinal variables: (All are in orders)

```
- High, Medium, Low
- Strongly agree, Agree, Neutral, Disagree, Strongly Disagree.
- Excellent, Okay, Bad
```

- Few examples as below for Nominal variable: (here no order)

```
- Red, Yellow, Pink, Blue
- Singapore, Japan, USA, India, Korea
- Cow, Dog, Cat, Snake
```

There are various technique to Handle categorical variable

- here i provided top 5 technique that use widly

1) One Hot Encoding (used both ordinal and nominal)

2) get_dummy (used both ordinal and nominal)

3) Label Encoding (used only ordinal)

4) mapping by replace function (used both ordinal and nominal)

Now lets implement All encoding method

In [1]:

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

In [2]:

```
data = {"Temp":['hot','cold','very hot','warm','hot','warm','warm','hot','hot','cold'], #or
        "Color":['red','yellow','black','black','red','white','black','yellow','black','wh
df = pd.DataFrame(data)
df.head(5)
```

Out[2]:

	Temp	Color
0	hot	red
1	cold	yellow
2	very hot	black
3	warm	black
4	hot	red

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Temp    10 non-null     object
 1   Color   10 non-null     object
dtypes: object(2)
memory usage: 288.0+ bytes
```

1. One Hot Encoding

- Scikit-learn has OneHotEncoder for this purpose, but it does not create an additional feature column.

In [4]:

```
df_ohc = df.copy()
df_ohc.head(5)
```

Out[4]:

	Temp	Color
0	hot	red
1	cold	yellow
2	very hot	black
3	warm	black
4	hot	red

In [5]:

```
from sklearn.preprocessing import OneHotEncoder
ohc = OneHotEncoder()
ohe = ohc.fit_transform(df_ohc.Temp.values.reshape(-1,1)).toarray()
dfOnehot = pd.DataFrame(ohe,columns=["Temp_"+str(ohc.categories_[0][i])
                                     for i in range (len(ohc.categories_[0]))
                                     ])
df_ohc = pd.concat([df_ohc,dfOnehot],axis=1)
df_ohc.head(5)
```

Out[5]:

	Temp	Color	Temp_cold	Temp_hot	Temp_very hot	Temp_warm
0	hot	red	0.0	1.0	0.0	0.0
1	cold	yellow	1.0	0.0	0.0	0.0
2	very hot	black	0.0	0.0	1.0	0.0
3	warm	black	0.0	0.0	0.0	1.0
4	hot	red	0.0	1.0	0.0	0.0

here i apply one hot encoding i only "Temp" , you can apply this any categorical variable irrespective of ordinal or nominal

2. get_dummies

- In this method, we map each category to a vector that contains 1 and 0 denoting the presence or absence of the feature. The number of vectors depends on the number of categories for features. This method produces a lot of columns that slows down the learning significantly if the number of the category is very high for the feature.
- this is very easy than sklearn onehot encoding. it just one line code.
- Pandas has get_dummies function, which is quite easy to use. For the sample data-frame code would be as below:

In [6]:

```
df_dummy = df.copy()
```

In [7]:

```
dfdummy = pd.get_dummies(df['Temp'],prefix="Temp_")
df_dummy = pd.concat([df_dummy,dfdummy],axis=1)
df_dummy.head(5)
```

Out[7]:

	Temp	Color	Temp__cold	Temp__hot	Temp__very hot	Temp__warm
0	hot	red	0	1	0	0
1	cold	yellow	1	0	0	0
2	very hot	black	0	0	1	0
3	warm	black	0	0	0	1
4	hot	red	0	1	0	0

- you compare sklearn onehotencoding and pandas getdummy, which is easy for you use.
- in this way you can apply onehot and getdummy to color variable

3. Label Encoding

- In this encoding, each category is assigned a value from 1 through N (here N is the number of categories for the feature. One major issue with this approach is there is no relation or order between these classes, but the algorithm might consider them as some order, or there is some relationship. In below example it may look like (Cold<Hot<Very Hot<Warm….0 < 1 < 2 < 3)
- Point to remember it only apply in ordinal variable.
- Scikit-learn code for the data-frame as follows:

In [8]:

```
df_label = df.copy()
from sklearn.preprocessing import LabelEncoder
lbe = LabelEncoder()
df_label['temp_label_encode'] = lbe.fit_transform(df_label.Temp)
df_label.head(5)
```

Out[8]:

	Temp	Color	temp_label_encode
0	hot	red	1
1	cold	yellow	0
2	very hot	black	2
3	warm	black	3
4	hot	red	1

Here see all our temp variable are convert according order.

4. Using Mapping by replace function

- here we replace the categorical variable with some numeric

In [9]:

```
# here i encode color column
df_mapp = df.copy()
df_mapp['Color'].value_counts()
```

Out[9]:

```
black     4
red       2
yellow    2
white     2
Name: Color, dtype: int64
```

In [10]:

```
df_mapp['Color_encode_map'] = df_mapp['Color'].replace(("black","white","red","yellow"),(0,
df_mapp.head(5)
```

Out[10]:

	Temp	Color	Color_encode_map
0	hot	red	2
1	cold	yellow	3
2	very hot	black	0
3	warm	black	0
4	hot	red	2

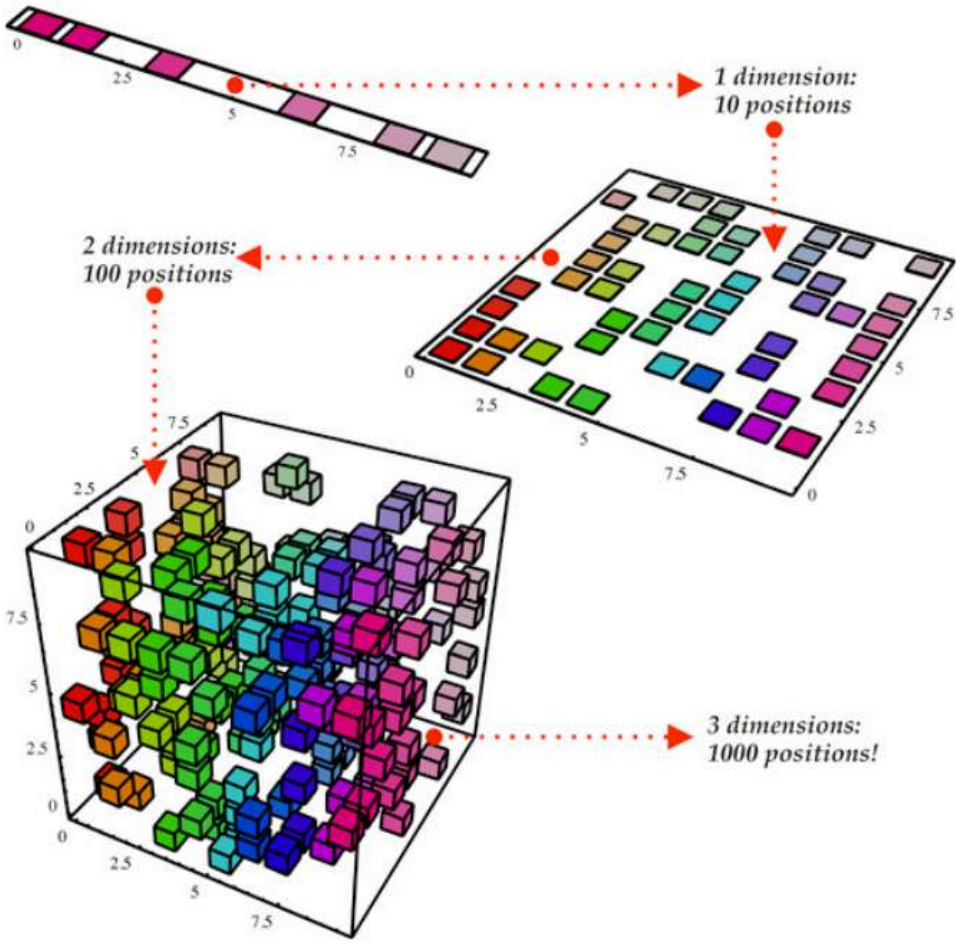
see here i encode black:0, white:1,red:2 and yellow:3.

Dimensionality Reduction for Machine Learning

- Have you ever worked on a dataset with more than a Hundred features? How about over 50,000 features?
- We have, and let us tell you it's a very challenging task, especially if you don't know where to start! Having a high number of variables is both a boon and a curse. It's great that we have loads of data for analysis, but it is challenging due to size.
- Using dimensionality reduction techniques, of course. You can use this concept to reduce the number of features in your dataset without having to lose much information and keep (or improve) the model's performance.

What is Dimensionality Reduction?

- Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables.



Why is Dimensionality Reduction required?

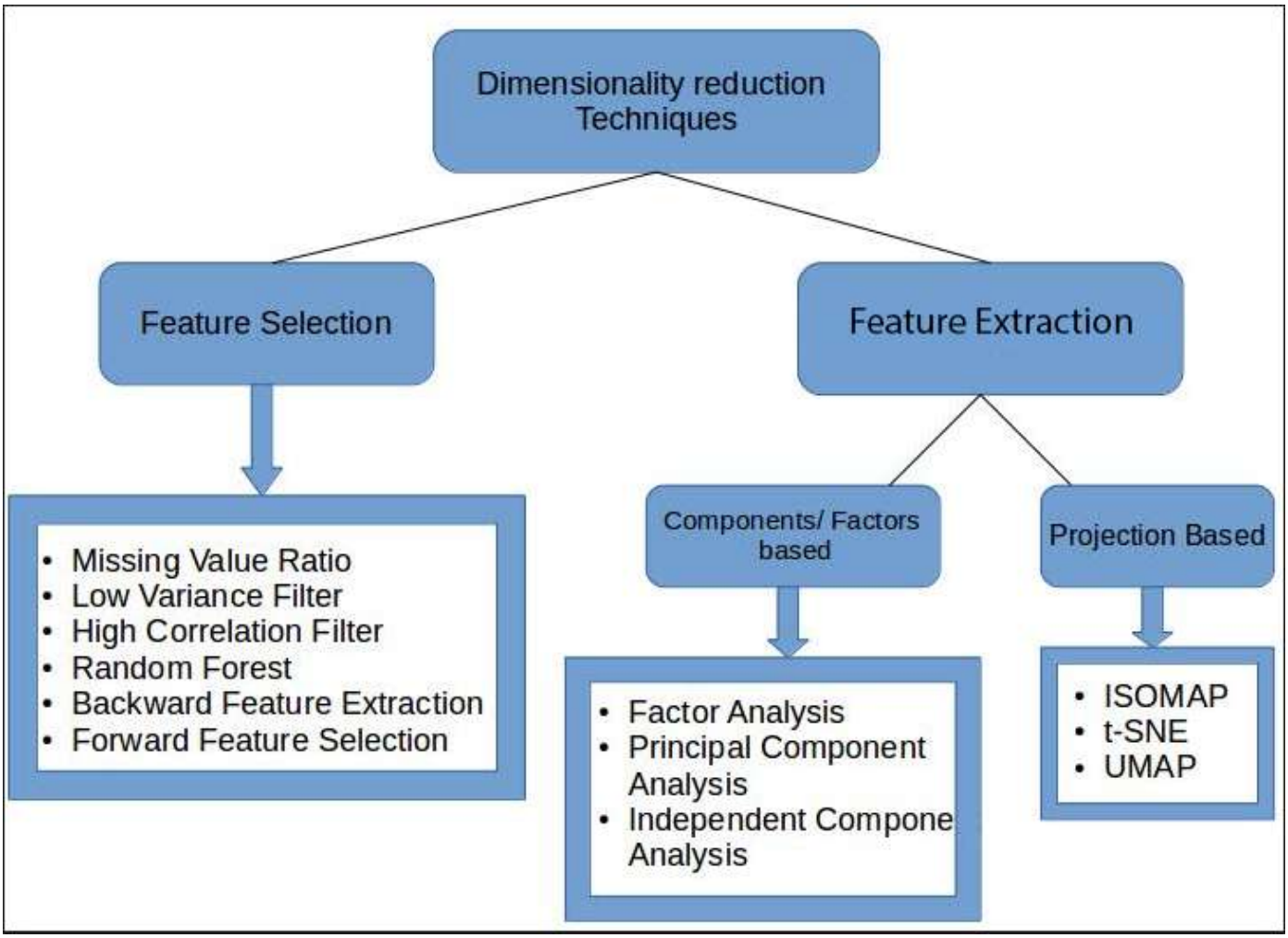
- Higher Dimensionality causes Curse of Dimensionality. so to reduce Curse of Dimensionality we can use Dimensionality Reduction.
- Fewer dimensions lead to less computation/training time.
- Some algorithms do not perform well when we have large dimensions. So reducing these dimensions needs to happen for the algorithm to be useful.

- It takes care of multicollinearity by removing redundant features. For example, you have two variables – ‘time spent on a treadmill in minutes’ and ‘calories burnt’. These variables are highly correlated as the more time you spend running on a treadmill, the more calories you will burn. Hence, there is no point in storing both as just one of them does what you require.
- It helps in visualizing data. As discussed earlier, it is very difficult to visualize data in higher dimensions so reducing our space to 2D or 3D may allow us to plot and observe patterns more clearly.
- If we have more features than observations than we run the risk of massively overfitting our model — this would generally result in terrible out of sample performance.

Curse of Dimensionality

- The curse of dimensionality basically means that the error increases with the increase in the number of features.
- It refers to the fact that algorithms are harder to design in high dimensions and often have a running time exponential in the dimensions

Dimensionality Reduction Techniques



Dimensionality Reduction Techniques

- Here I Covered 4 Method.

    1. Missing Value Ratio
    2. High Correlation Fillter
    3. Low_variance_filter
    4. By Random Forest Feature Selection

1. Missing Value Ratio (Method- 1)

In [1]:
```
#importing the libraries
import pandas as pd
#reading the file
data = pd.read_csv('missing_value_ratio.csv')
# first 5 rows of the data
data.head()
```

Out[1]:

	ID	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	count
0	AB101	1.0	0.0	0.0	1.0	9.84	14.395	81.0	NaN	16
1	AB102	1.0	NaN	0.0	NaN	9.02	13.635	80.0	NaN	40
2	AB103	1.0	0.0	NaN	1.0	9.02	13.635	80.0	NaN	32
3	AB104	NaN	0.0	NaN	1.0	9.84	14.395	75.0	NaN	13
4	AB105	1.0	NaN	0.0	NaN	9.84	14.395	NaN	16.9979	1

See here lot of missing value present in this dataset so here we put missing value ratio.

In [2]:
```
# percentage of missing values in each variable
data.isnull().sum()/len(data)*100
```

Out[2]:
```
ID             0.000000
season         0.069337
holiday       48.497689
workingday     0.069337
weather        0.030817
temp           0.000000
atemp          0.000000
humidity       0.038521
windspeed     41.016949
count          0.000000
dtype: float64
```

In [3]:
```
# saving missing values in a variable
a = data.isnull().sum()/len(data)*100
```

In [4]:
```
# saving column names in a variable
variables = data.columns
```

In [5]:
```
# new variable to store variables having missing values less than a threshold
variable = [ ]
for i in range(data.columns.shape[0]):
    if a[i]<=40:   #setting the threshold as 40%
        variable.append(variables[i])
```

- here i remove the columns which has missing value higher than 40%. 40% is just a threshold.
- You set a threshold which has Higher than 40-45 % .

In [6]:
```
print("--------------------Before remove feature------------------")
print("Before remove feature: ",variables)
print("--------------------After remove feature------------------")
print("After remove feature: ",variable)
```

```
--------------------Before remove feature------------------
Before remove feature:  Index(['ID', 'season', 'holiday', 'workingday', 'weathe
r', 'temp', 'atemp',
       'humidity', 'windspeed', 'count'],
      dtype='object')
--------------------After remove feature------------------
After remove feature:  ['ID', 'season', 'workingday', 'weather', 'temp', 'atem
p', 'humidity', 'count']
```

Here i remove 'holiday' and 'windspeed' as it has higher missing value ratio

In [7]:
```
# creating a new dataframe using the above variables
new_data = data[variable]
# first five rows of the new data
new_data.head()
```

Out[7]:

	ID	season	workingday	weather	temp	atemp	humidity	count
0	AB101	1.0	0.0	1.0	9.84	14.395	81.0	16
1	AB102	1.0	0.0	NaN	9.02	13.635	80.0	40
2	AB103	1.0	NaN	1.0	9.02	13.635	80.0	32
3	AB104	NaN	NaN	1.0	9.84	14.395	75.0	13
4	AB105	1.0	0.0	NaN	9.84	14.395	NaN	1

In [8]:
```
# percentage of missing values in each variable of new data
new_data.isnull().sum()/len(new_data)*100
```

Out[8]:
```
ID            0.000000
season        0.069337
workingday    0.069337
weather       0.030817
temp          0.000000
atemp         0.000000
humidity      0.038521
count         0.000000
dtype: float64
```

In [9]:
```
# shape of new and original data
print("Before Remove Mising value RAtio: ",data.shape)
print("Before Remove Mising value RAtio: ",new_data.shape)
```

```
Before Remove Mising value RAtio:  (12980, 10)
Before Remove Mising value RAtio:  (12980, 8)
```

In this way we reduce dimensionality of a data using missing value ratio.

2. High Correlation Fillter (Method 2)

- Here I just Demonstrate how you remove high correlation feature.

In [10]:
```
data_me2 = data.copy()
```

In [11]:
```
data_me2 = data_me2.drop(['ID'],axis=1)
```

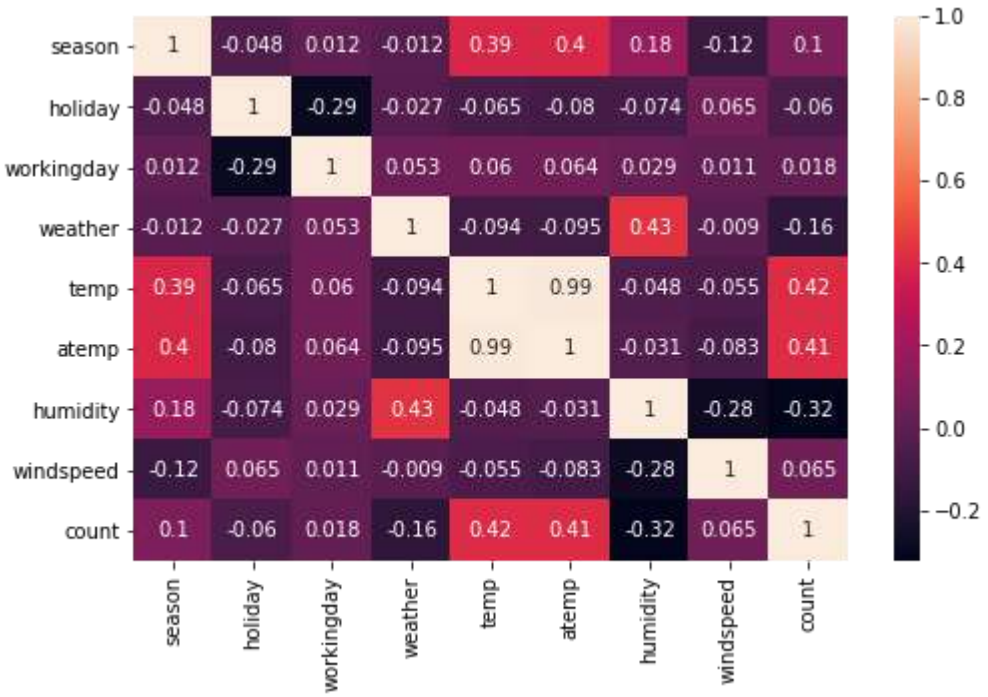
In [12]:
```
correlation = data_me2.corr()
numeric_columns = data_me2.columns

high_corr = [ ]

for c1 in numeric_columns:
  for c2 in numeric_columns:
    if c1 != c2 and c2 not in high_corr and correlation[c1][c2] > 0.9:
      high_corr.append(c1)
```

In [13]:
```
high_corr
```

Out[13]:
```
['temp']
```

In [14]:
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(8,5))
sns.heatmap(correlation,annot=True)
```

Out[14]: <AxesSubplot:>



- See here in heat map atemp and temp have high correlation so my method remove one higher correlation Value.

3. Low_variance_filter (Method 3)

In [15]:
```
#importing the libraries
import pandas as pd
from sklearn.preprocessing import normalize
#reading the file
data = pd.read_csv('low_variance_filter.csv')
# first 5 rows of the data
data.head()
```

Out[15]:

	ID	temp	atemp	humidity	windspeed	count
0	AB101	9.84	14.395	81	0.0	16
1	AB102	9.02	13.635	80	0.0	40
2	AB103	9.02	13.635	80	0.0	32
3	AB104	9.84	14.395	75	0.0	13
4	AB105	9.84	14.395	75	0.0	1

In [16]:
```
#percentage of missing values in each variable
data.isnull().sum()/len(data)*100
```

Out[16]:
```
ID           0.0
temp         0.0
atemp        0.0
humidity     0.0
windspeed    0.0
count        0.0
dtype: float64
```

- Here see missing value zero so here we consider variance value to reduce dimensionality.

In [17]:
```
data.var()
```

Out[17]:
```
temp            61.291712
atemp           73.137484
humidity       398.549141
windspeed       69.322053
count        25843.419864
dtype: float64
```

- See here all the variance value shows.
- Always do normalize of your dataset before doing This method.

In [18]:
```
#creating dummy variables of categorical variables
data = data.drop('ID', axis=1)
```

In [19]:
```
normalize = normalize(data)
```

In [20]:
```
data_scaled = pd.DataFrame(normalize)
data_scaled.var()
```

Out[20]:
```
0    0.005877
1    0.007977
2    0.093491
3    0.008756
4    0.111977
dtype: float64
```

- Now use a threshold value to elminate all low var feature.

In [21]:
```
#storing the variance and name of variables
variance = data_scaled.var()
columns = data.columns
```

In [22]:
```
#saving the names of variables having variance more than a threshold value
variable = [ ]
for i in range(0,len(variance)):
    if variance[i]>=0.006:   #setting the threshold as 1%
        variable.append(columns[i])
```

In [23]:
```
print("--------------------Before remove feature------------------")
print("Before remove feature: ",data.columns)
print("--------------------After remove feature------------------")
print("After remove feature: ",variable)
```

```
--------------------Before remove feature------------------
Before remove feature:  Index(['temp', 'atemp', 'humidity', 'windspeed', 'coun
t'], dtype='object')
--------------------After remove feature------------------
After remove feature:  ['atemp', 'humidity', 'windspeed', 'count']
```

- Here it eliminate temp as it var value is 0.005877.

In [24]:
```
# creating a new dataframe using the above variables
new_data = data[variable]
# first five rows of the new data
new_data.head()
```

Out[24]:

	atemp	humidity	windspeed	count
0	14.395	81	0.0	16
1	13.635	80	0.0	40
2	13.635	80	0.0	32
3	14.395	75	0.0	13
4	14.395	75	0.0	1

In [25]:
```
# shape of new and original data
print("Before Remove Mising value RAtio: ",data.shape)
print("Before Remove Mising value RAtio: ",new_data.shape)
```

```
Before Remove Mising value RAtio:  (12980, 5)
Before Remove Mising value RAtio:  (12980, 4)
```

- In This way we do Low Variance Fillter

4. By Random Forest Feature Selection (Method 4)

In [26]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [27]:
```
from sklearn.datasets import load_iris
import pandas as pd
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = data.target
df.head()
```

Out[27]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

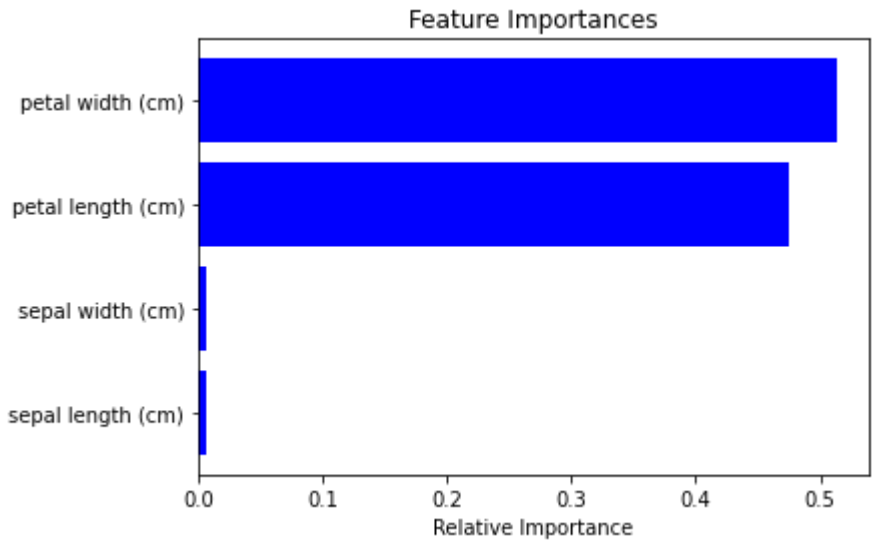
In [28]:
```
X = df.drop("Target",axis=1)
y = df['Target']
```

In [29]:
```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(random_state=1, max_depth=10)
model.fit(X,y)
```

Out[29]:
```
RandomForestRegressor(max_depth=10, random_state=1)
```

After fitting the model, plot the feature importance graph:

In [30]:
```
features = df.columns
importances = model.feature_importances_
indices = np.argsort(importances)[-9:]  # top 10 features
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



- By this you can determine which variable are important which are not . according to this you easily eliminate all less important feature.
- Based on the above graph, we can handpick the top-most features to reduce the dimensionality in our dataset.
- Alternatively, we can use the SelectFromModel of sklearn to do so. It selects the features based on the importance of their weights.

SelectFromModel of sklearn

In [31]:
```
from sklearn.feature_selection import SelectFromModel
feature = SelectFromModel(model)
X_select = feature.fit_transform(X,y)
```

In [32]:
```
print("--------------------Before remove feature------------------")
print("Before Select: ",X.shape)
print("--------------------After remove feature------------------")
print("After Select: ",X_select.shape)
```

```
--------------------Before remove feature------------------
Before Select:  (150, 4)
--------------------After remove feature------------------
After Select:  (150, 2)
```

In [33]:
```
model.fit(X_select,y)
```

Out[33]:
```
RandomForestRegressor(max_depth=10, random_state=1)
```

- In This way you can do number of feature and easily emliminate this.

Feature Selection :

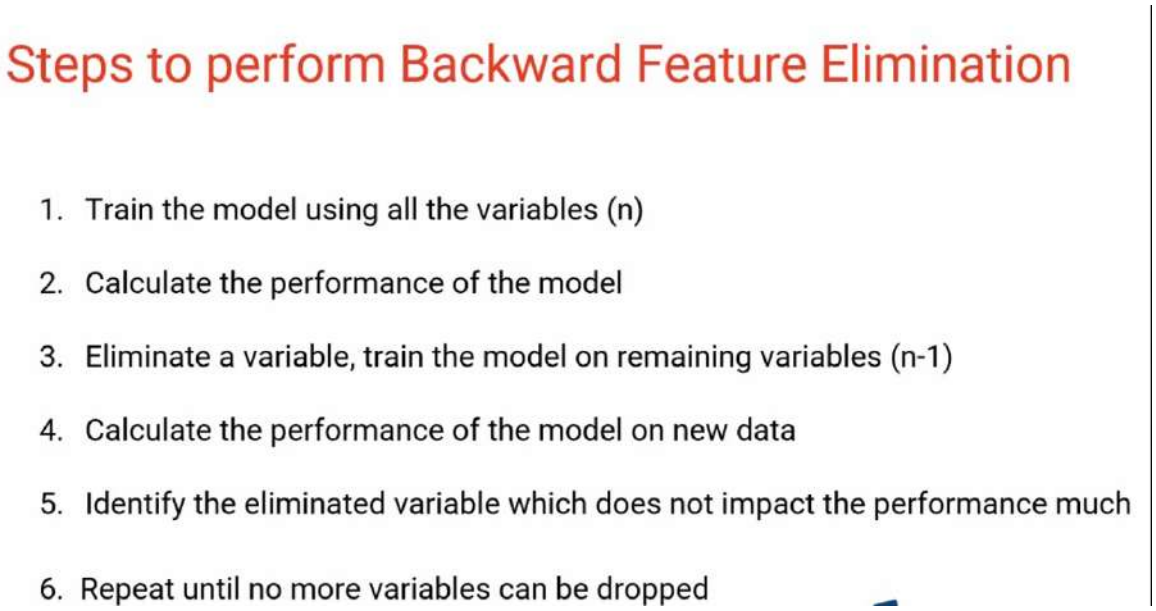
Here I covered two Feature slection Techniques That are

1. Backward Feature Elimination
2. Forward Feature Selection

When It Use ??

- When No missing Values in Dataset.
- Variance of all the variable are high.
- Low Correlation between independents variable.

These are the assumption , when it occure then we go for Backward Feature elimination or Forward Feature Selection.



You Nedd to install mlxtend library to perform these two opeartion

- **!pip install mlxtend**

1. Backward Feature Elimination Implementation

- In backward elimination, we start with all the features and removes the least significant feature at each iteration which improves the performance of the model. We repeat this until no improvement is observed on removal of features.

In [1]:
```
import warnings
warnings.simplefilter('ignore')
#importing the libraries
import pandas as pd
#reading the file
data = pd.read_csv('backward_feature_elimination.csv')
#shape of the data
print(data.shape)
# first 5 rows of the data
data.head()
```

```
(12980, 9)
```

Out[1]:

	ID	season	holiday	workingday	weather	temp	humidity	windspeed	count
0	AB101	1	0	0	1	9.84	81	0.0	16
1	AB102	1	0	0	1	9.02	80	0.0	40
2	AB103	1	0	0	1	9.02	80	0.0	32
3	AB104	1	0	0	1	9.84	75	0.0	13
4	AB105	1	0	0	1	9.84	75	0.0	1

In [2]:
```
# checking missing values in the data
data.isnull().sum()
```

Out[2]:
```
ID            0
season        0
holiday       0
workingday    0
weather       0
temp          0
humidity      0
windspeed     0
count         0
dtype: int64
```

- See here No Missing Value

In [3]:
```
# creating the training data
X = data.drop(['ID','count'], axis=1)
y = data['count']
```

In [4]:
```
X.shape, y.shape
```

Out[4]:
```
((12980, 7), (12980,))
```

In [5]:
```
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
from sklearn.linear_model import LinearRegression
```

In [6]:
```
lreg = LinearRegression()
sfs1 = sfs(lreg, k_features=5, forward=False, verbose=1, scoring='neg_mean_square
```

- in sfs

  1st params = model_name

  2nd params= how many feature you want to select

  3rd params = forward false means it use backward feature eliminition

  4th params= verbose is for print score with each itration

  5th params = scoring is the score.

In [7]:
```
sfs1 = sfs1.fit(X, y)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.1s finished
Features: 6/5[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concu
rrent workers.
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.1s finished
Features: 5/5
```

In [8]:
```
feat_names = list(sfs1.k_feature_names_)
print(feat_names) #it return that it select
```

```
['holiday', 'workingday', 'weather', 'temp', 'humidity']
```

In [9]:
```
new_data = data[feat_names]
new_data['count'] = data['count']
```

In [10]:
```
# first five rows of the new data
new_data.head()
```

Out[10]:

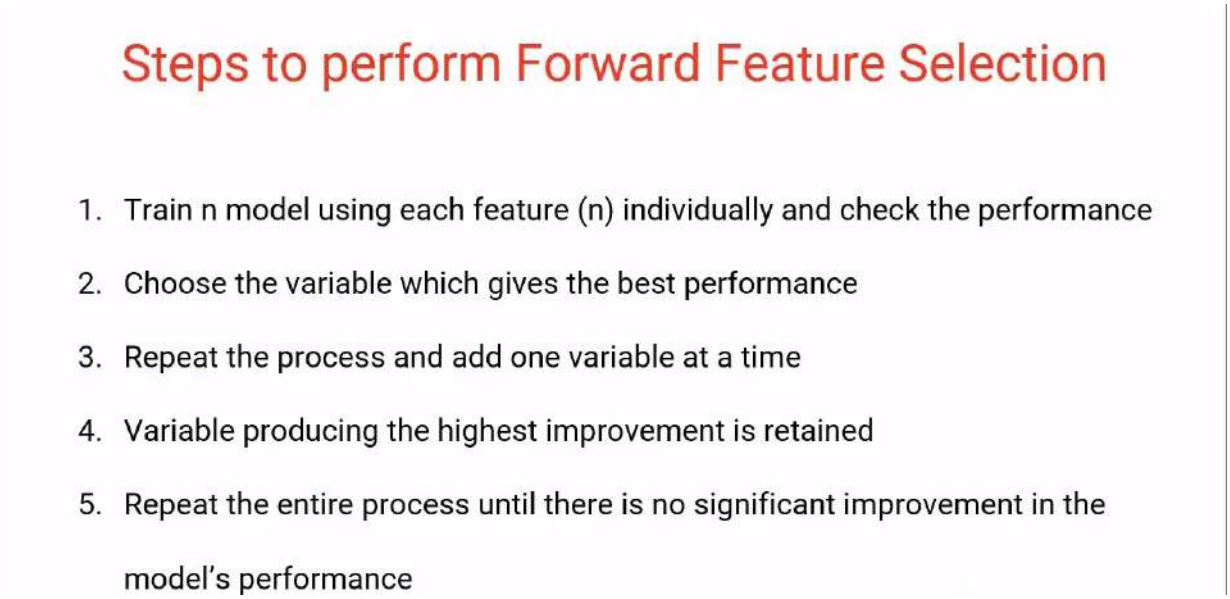
	holiday	workingday	weather	temp	humidity	count
0	0	0	1	9.84	81	16
1	0	0	1	9.02	80	40
2	0	0	1	9.02	80	32
3	0	0	1	9.84	75	13
4	0	0	1	9.84	75	1

In [11]:
```
# shape of new and original data
new_data.shape, data.shape
```

Out[11]:
```
((12980, 6), (12980, 9))
```

- See here new_data column is 6 as backward feature elimination eliminate 3 feature.

2. Forward_feature_selection



In [12]:
```
#importing the libraries
import pandas as pd
#reading the file
data = pd.read_csv('forward_feature_selection.csv')
#shape of the data
print(data.shape)
# first 5 rows of the data
data.head()
```

```
(12980, 9)
```

Out[12]:

	ID	season	holiday	workingday	weather	temp	humidity	windspeed	count
0	AB101	1	0	0	1	9.84	81	0.0	16
1	AB102	1	0	0	1	9.02	80	0.0	40
2	AB103	1	0	0	1	9.02	80	0.0	32
3	AB104	1	0	0	1	9.84	75	0.0	13
4	AB105	1	0	0	1	9.84	75	0.0	1

In [13]:
```
# checking missing values in the data
data.isnull().sum()
```

Out[13]:
```
ID            0
season        0
holiday       0
workingday    0
weather       0
temp          0
humidity      0
windspeed     0
count         0
dtype: int64
```

In [14]:
```
# creating the training data
X = data.drop(['ID', 'count'], axis=1)
y = data['count']
```

In [15]:
```
X.shape, y.shape
```

Out[15]:
```
((12980, 7), (12980,))
```

In [16]:
```
# importing the models
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
from sklearn.linear_model import LinearRegression
```

In [17]:
```
# calling the linear regression model
lreg = LinearRegression()
sfs1 = sfs(lreg, k_features=4, forward=True, verbose=2, scoring='neg_mean_squared
```

- Here all are same as backward feature elimination but in forward we write True here so it use forward feature selection
- in sfs

  1st params = model_name

  2nd params= how many feature you want to select

  3rd params = forward true as it use forward feature selection

  4th params= verbose is for print score with each itration

  5th params = scoring is the score.

In [18]:
```
sfs1 = sfs1.fit(X, y)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.1s finished

[2020-10-23 10:33:57] Features: 1/4 -- score: -23364.955503181[Parallel(n_jobs=
1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.0s finished

[2020-10-23 10:33:57] Features: 2/4 -- score: -21454.899339219788[Parallel(n_jo
bs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s finished

[2020-10-23 10:33:57] Features: 3/4 -- score: -21458.278788564392[Parallel(n_jo
bs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s finished

[2020-10-23 10:33:57] Features: 4/4 -- score: -21477.98857350279
```

In [19]:
```
feat_names = list(sfs1.k_feature_names_)
print(feat_names)
```

```
['holiday', 'workingday', 'temp', 'humidity']
```

In [20]:
```
# creating a new dataframe using the above variables and adding the target variab
new_data = data[feat_names]
new_data['count'] = data['count']
# first five rows of the new data
new_data.head()
```

Out[20]:

	holiday	workingday	temp	humidity	count
0	0	0	9.84	81	16
1	0	0	9.02	80	40
2	0	0	9.02	80	32
3	0	0	9.84	75	13
4	0	0	9.84	75	1

In [21]:
```
# shape of new and original data
new_data.shape, data.shape
```

Out[21]:
```
((12980, 5), (12980, 9))
```

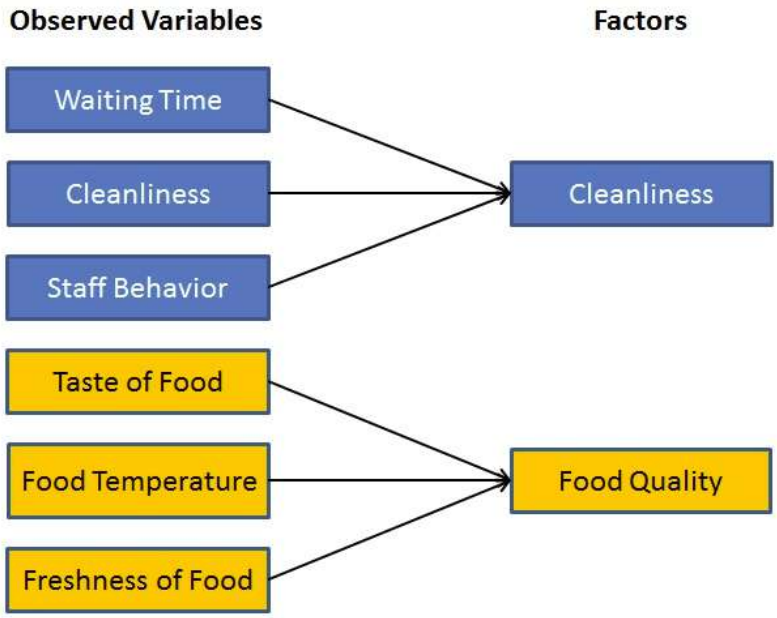
- By this we can reduce our dimensionality
- Till now we study all feature selection method, in my next notebook i am going to describe feature extraction method where most famous feature extraction are PCA,t-sne are used.

Feature Extraction

- Feature Extraction is the technique where you extract new features using the existing one.
- In this module, we will look at some prominent Factor Based feature extraction techniques based on along with its implementation in python.

1. Factor Analysis

- Factor Analysis (FA) is an exploratory data analysis method used to search influential underlying factors or latent variables from a set of observed variables.
- It helps in data interpretations by reducing the number of variables. It extracts maximum common variance from all variables and puts them into a common score.
- It is used basically in market research, advertising, psychology, finance, and operation research.



When We use this ??

- There are no outliers in data.
- The sample size should be greater than the factor.
- There should not be perfect multicollinearity.
- There should not be homoscedasticity between the variables.

Types of factor analysis:

1. Exploratory factor analysis (EFA)
2. Confirmatory factor analysis (CFA)
3. Multiple Factor Analysis
4. Generalized Procrustes Analysis (GPA)

To study more follow this https://www.geeksforgeeks.org/introduction-to-factor-analytics/ (https://www.geeksforgeeks.org/introduction-to-factor-analytics/)

Implementation

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.decomposition import FactorAnalysis
```

In [2]:
```
iris = load_iris()
X=iris.data
variable = iris.feature_names
```

In [3]:
```
factor = FactorAnalysis(random_state=0)
factor.fit_transform(X)
pd.DataFrame(factor.components_,columns = variable)
```

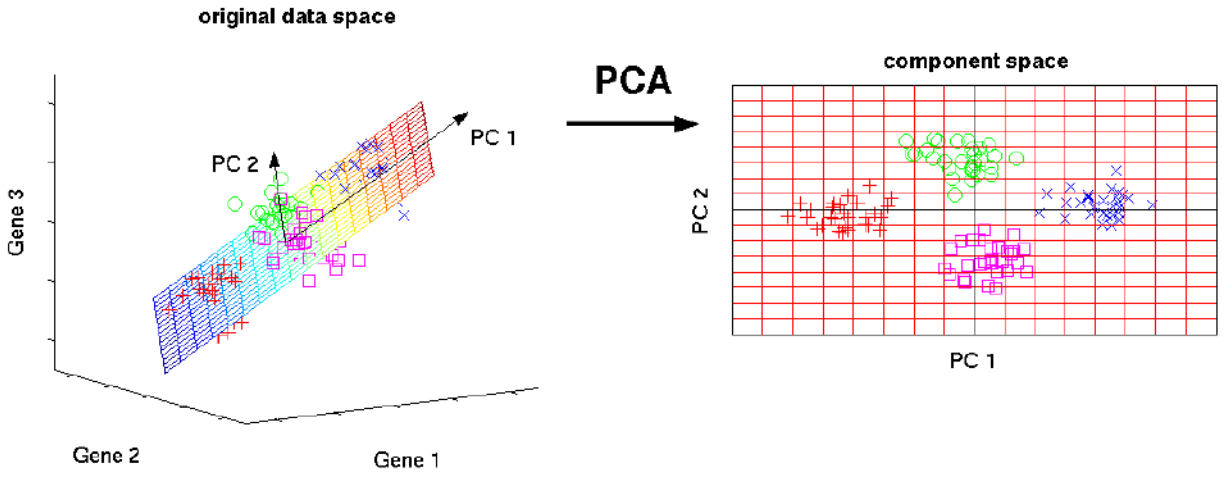
Out[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	0.706989	-0.158005	1.654236	0.70085
1	0.115161	0.159635	-0.044321	-0.01403
2	-0.000000	0.000000	0.000000	0.00000
3	-0.000000	0.000000	0.000000	-0.00000

- In this way factor analysis occured

2. PCA (Principal Component Analysis)

- The Principal Component Analysis(PCA) is a way of reducing the dimensions of a given dataset by extracting new features from the original features present in the dataset. So it combines our input variables(or features) in a specific way and gives “new” features by retaining the most valuable information of all the original features. The “new” variables after PCA are all independent of one another.



Why it is used :

- It is used when we need to tackle the curse of dimensionality among data with linear relationships, i.e. we're having too many dimensions (features) in your data causes noise and difficulties (it can be sound, picture or context). This specifically get worst when features have different scales (e.g. weight, length, area, speed, power, temperature, volume, time, cell number, etc.)

When should I use PCA:

1. Do you want to reduce the number of variables, but aren't able to identify variables to completely remove from consideration?
2. Do you want to ensure your variables are independent of one another?
3. Are you comfortable making your independent variables less interpretable?

If you answered "yes" to all three questions, then PCA is a good method to use. If you answered "no" to question 3, you should not use PCA.

Point to remember while doing PCA:

- The first step we need to do while performing PCA is data standardization. i.e. scaling the data to mean as 0 and variance as 1.

Implement

In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

In [5]:
```
#loads breast cancer dataset into variable by name cancer.
cancer = load_breast_cancer()
# creating dataframe
df = pd.DataFrame(cancer['data'], columns = cancer['feature_names'])
# checking head of dataframe
df.head()
```

Out[5]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	d
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	

5 rows × 30 columns

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

Out[6]:
```
(569, 30)
```

- So clearly, we can see that there are 30 columns and each of them are numerical values. So we can directly apply PCA.

- before applying PCA we need to scale or feature

In [7]:
```
scalar = StandardScaler()
  # Standardizing
scalar.fit(df)
scaled_data = scalar.transform(df)
```

In [8]:
```
# applying PCA
pca = PCA(n_components = 3)
pca.fit(scaled_data)
x_pca = pca.transform(scaled_data)
```

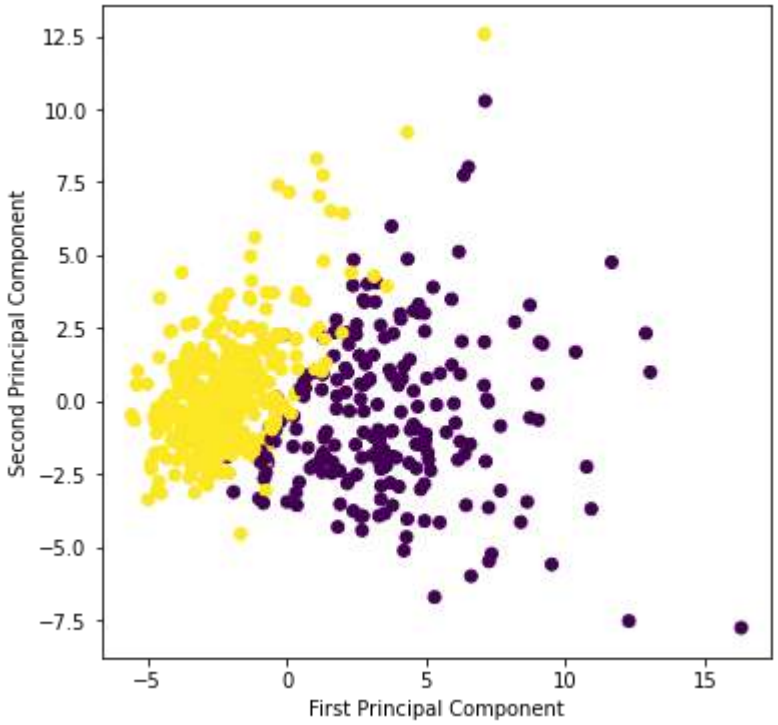
- In this, what we are doing is standardizing the data(i.e. df) and applying PCA on it.
- There, n-components represents the number of principal components(i.e. new features) that we want to.

In [9]:
```
print("Befor PCA shape: ",df.shape)
print("After PCA shape: ",x_pca.shape)
```

```
Befor PCA shape:  (569, 30)
After PCA shape:  (569, 3)
```

In [10]:
```
plt.figure(figsize =(6,6))

plt.scatter(x_pca[:, 0], x_pca[:, 1], c = cancer['target'])

# labeling x and y axes
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
```

Out[10]:
```
Text(0, 0.5, 'Second Principal Component')
```

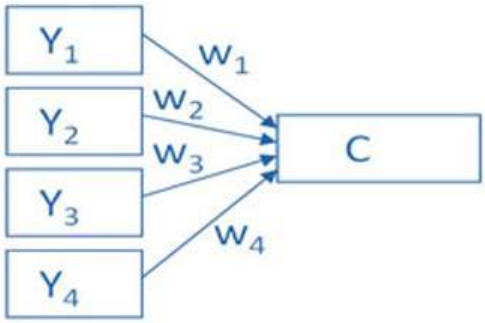


- In this step, I'm just displaying the 2-Dimensional plot of the data and it looks something like this.

Factor Analysis Vs. Principle Component Analysis

- PCA components explain the maximum amount of variance while factor analysis explains the covariance in data.
- PCA components are fully orthogonal to each other whereas factor analysis does not require factors to be orthogonal.
- PCA components are uninterpretable. In FA, underlying factors are labelable and interpretable.
- PCA is a kind of dimensionality reduction method whereas factor analysis is the latent variable method.
- PCA is a type of factor analysis. PCA is observational whereas FA is a modeling technique.

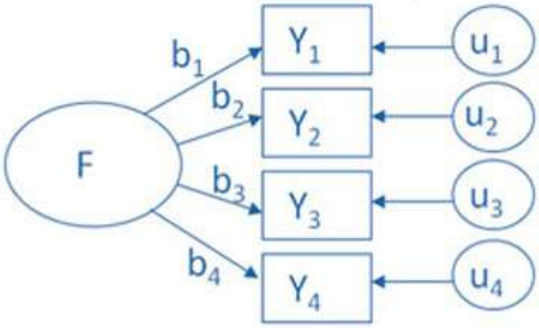
PCA



This model can be set up as a simple equation:

C = w1(Y1) + w2(Y2) + w3(Y3) + w4(Y4)

Factor Analysis



You can literally interpret this model as a set of regression equations:

Y1 = b1*F + u1
Y2 = b2*F + u2
Y3 = b3*F + u3
Y4 = b4*F + u4

PCA—how to choose the number of components?

- In this article, I am going to show you how to choose the number of principal components when using principal component analysis for dimensionality reduction.
- Principal Component Analysis (PCA) is an unsupervised technique for dimensionality reduction.

How does PCA work exactly? ¶

- PCA is the eigenvalue decomposition of the covariance matrix obtained after centering the features, to find the directions of maximum variation. The eigenvalues represent the variance explained by each principal component.
- The purpose of PCA is to obtain an easier and faster way to both manipulate data set (reducing its dimensions) and retain most of the original information through the explained variance.

The question now is How many components should I use for dimensionality reduction? What is the "right" number?

- In this post, we will discuss some tips for selecting the optimal number of principal components by providing practical examples in Python, by:

```
  1. Observing the cumulative ratio of explained variance. (always t
ake variance greater than 90%)
  2. Apply Cross Validation
  3. Using the covariance matrix With Elbow Curve
```

- Here I discuss 3 Method To Choose n_copmponent in PCA. You can choose any of One among this three

Hands On Example

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
%matplotlib inline
```

In [2]:
```
iris = load_iris()
iris_df = pd.DataFrame(iris.data,columns=[iris.feature_names])
iris_df.head()
```

Out[2]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [3]:
```
iris_df['target'] = iris.target
```

In [4]:
```
X = iris.data
y = iris.target
```

In [5]:
```
X.shape,y.shape
```

Out[5]:
```
((150, 4), (150,))
```

Before Doing PCA You must scalling the Dataset

In [6]:
```
from sklearn.preprocessing import StandardScaler
X_std = StandardScaler().fit_transform(X)
```

In [7]:
```
X_std.shape
```

Out[7]:
```
(150, 4)
```

Method:1 Observing the cumulative ratio of explained variance.

- here 1st we have to Compute the co-variance matrix
- In Second Compute the eigen values and vectors.
- In last step Project the data along the top eigen vectors based on the n_components.

In [8]:
```
## Compute the eigen values and vectors And Arrange the eigen values in the desce
X_covariance_matrix = np.cov(X_std.T)
eig_vals, eig_vecs = np.linalg.eig(X_covariance_matrix)
print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)
```

```
Eigenvectors
[[ 0.52106591 -0.37741762 -0.71956635  0.26128628]
 [-0.26934744 -0.92329566  0.24438178 -0.12350962]
 [ 0.5804131  -0.02449161  0.14212637 -0.80144925]
 [ 0.56485654 -0.06694199  0.63427274  0.52359713]]

Eigenvalues
[2.93808505 0.9201649  0.14774182 0.02085386]
```

In [9]:
```
tot = sum(eig_vals)
var_exp = [(i / tot)*100 for i in sorted(eig_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
print ("Variance captured by each component is \n",var_exp)
print(40 * '-')
print ("Cumulative variance captured as we travel each component \n",cum_var_exp)
```

```
Variance captured by each component is
 [72.96244541329985, 22.850761786701778, 3.668921889282875, 0.5178709107154802]
----------------------------------------
Cumulative variance captured as we travel each component
 [ 72.96244541  95.8132072   99.48212909 100.        ]
```

- Now you add all your Variance captured by each component and when it will be greater than 90 then you stop.
- Here Clearly the variance captured by the 1st 2 features when arranged in descending order contribute to almost 72+22 = 94% of the total variance by the features. Thus we can eliminate the remaining 2 features as they don't contribute much to the overall variance.
- Thus, the mystery of How to choose n_component in PCA is Solved
- n_components should be equal to the features which contribute a large number to the overall variance! The number depends on the business logic.

- So in this problem you take n_components = 2 as two feature have high value of Variance. we want the variance to be between 95–99%.

Method 2: Applying a cross-validation procedure

In [10]:
```
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.pipeline import Pipeline
```

In [11]:
```
pca = PCA(len(iris.feature_names)-1)
log_reg = LogisticRegression(max_iter=1000)

pipe = Pipeline(steps=[('pca', pca), ('log_reg', log_reg)])

params = {
    'pca__n_components': list(range(1, len(iris.feature_names))),
    'log_reg__C': np.logspace(0.1, 1, 10)
}

random_search = RandomizedSearchCV (pipe, params)
random_search.fit(X, y)

print('Best parameters obtained from Grid Search:\n',random_search.best_params_,
      random_search.best_score_)
```

```
Best parameters obtained from Grid Search:
 {'pca__n_components': 3, 'log_reg__C': 3.9810717055349722} 0.9800000000000001
```

In [12]:
```
results = pd.DataFrame(random_search.cv_results_)
results.head()
```

Out[12]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_pca__n_components	param
0	0.016112	0.004202	0.000300	0.000400	3	
1	0.010509	0.001265	0.000801	0.000400	3	
2	0.014011	0.002760	0.000700	0.000400	3	
3	0.010308	0.000401	0.000601	0.000491	2	
4	0.009807	0.001503	0.000601	0.000375	1	

Here you see in 3 component model give 96 and also 2 component model gives 96 so always take lowest feature because our aim to reduce dimensionality in dataset so here also you take 2 as n_component

Method 3: Using the covariance matrix With Elbow Curve

In [13]:
```
pca = PCA(n_components = len(iris.feature_names)-1)
principal_components = pca.fit_transform(X)

# calculating the covariance matrix
covMatrix = np.cov(principal_components.transpose())
```

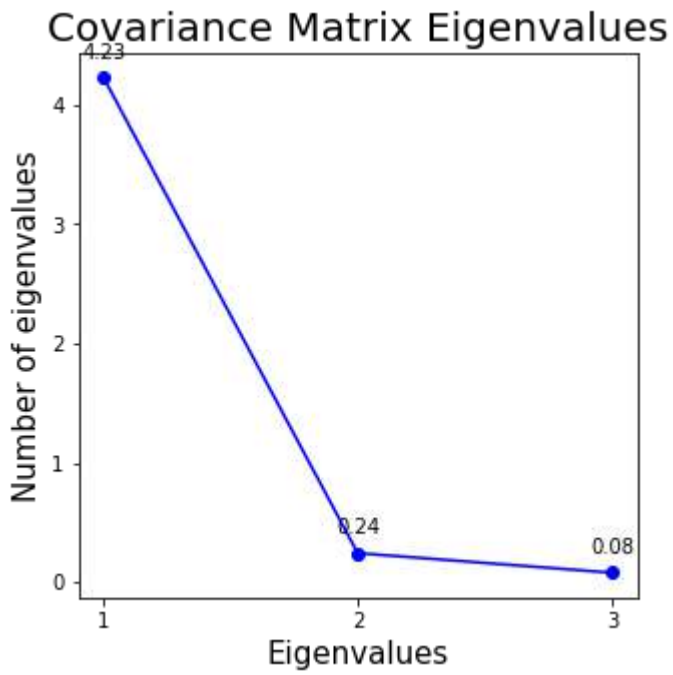
In [14]:
```
np.set_printoptions(suppress=True)
covMatrix
```

Out[14]:
```
array([[ 4.22824171, -0.        ,  0.        ],
       [-0.        ,  0.24267075, -0.        ],
       [ 0.        , -0.        ,  0.0782095 ]])
```

In [15]:
```
np.testing.assert_almost_equal(covMatrix - np.diag
                               (np.diagonal(covMatrix)), 0, decimal=10)
```

In [17]:
```
xp = np.arange(1, len(iris.feature_names), 1)
w = np.diag(covMatrix)

plt.clf()
plt.figure(figsize=(5,5))
plt.plot(xp, w, 'bo-')
plt.title("Covariance Matrix Eigenvalues", fontsize=20)
plt.xlabel('Eigenvalues', fontsize=15)
plt.ylabel('Number of eigenvalues',fontsize=15)
for x_plot,y_plot in zip(xp,w):
    label = "{:.2f}".format(y_plot)
    plt.annotate(label,
                 (x_plot,y_plot),
                 textcoords="offset points",
                 xytext=(0,10),
                 ha='center')
plt.xticks(np.arange(1,len(iris.feature_names),1))
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



This Looks like your elbow method. By this also we calculate the n_component. Here n_copmonent also 2

Conclude :

- This tutorial is meant to provide a few tips on the selection of the number of components to be used for the dimensionality reduction in the PCA, showing practical demonstrations in Python.
- Finally, it is also explained how to perform the projection onto the reduced subspace of a new sample, information which is rarely found on tutorials on the subject.
- So Now i hope the mystry behind How to choose n_component in PCA is now Solved

What Is Bias ?

- It is difference between Predicted Value and Actual Value.
- Bias refers to the error introduced due to assumptions made to model a relatively complex real-life problem by an approximate simple model.

- Low Bias: Predicting less assumption about Target Function(Predict Value >> Actual Value)

    - `difference is Very Samll Between predcit and Actual Value`

- High Bias: Predicting more assumption about Target Function (Predict Value >>>>>> Actual Value)

    - `difference is very much large between predcit and Actual Value`

Examples of low-bias machine learning algorithms include Decision Trees, k-Nearest Neighbors and Support Vector Machines.

Examples of high-bias machine learning algorithms include Linear Regression, Linear Discriminant Analysis, and Logistic Regression.

What Is Variance ?

- Variance is the amount that the estimate of the target function will change if different training data was used.
- It determine how spread of predcit value each other

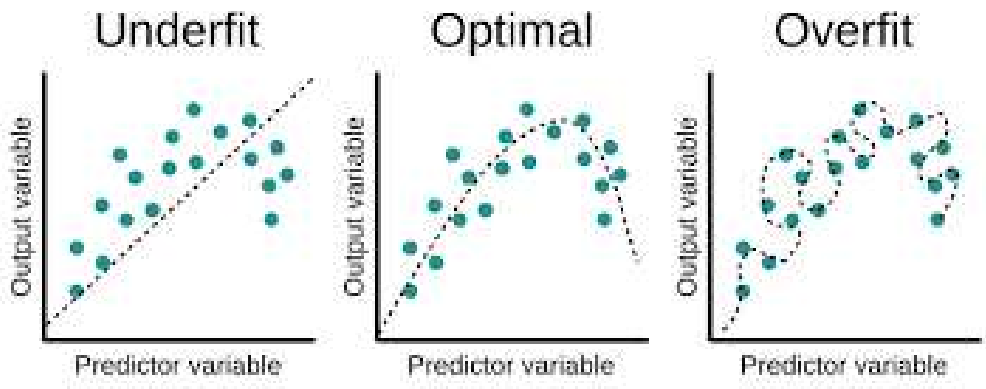
    - `Low Variance: Predicting small changes to the estimate of the target function with changes to the training dataset.`

    - `High Variance: Predicting large changes to the estimate of the target function with changes to the training dataset.`

- Examples of low-variance machine learning algorithms include Linear Regression, Linear Discriminant Analysis, and Logistic Regression.
- Examples of high-variance machine learning algorithms include Decision Trees, k-Nearest Neighbors and Support Vector Machines.

Till now we have some basic knowledge about Bias and Variance, now let's know some other important terminology used in Bias and Variance that are Underfitting and Overfitting.

Overfit And Underfit :



Underfit :

- These models usually have high bias and low variance.
- underfitting happens when a model unable to capture the underlying pattern of the data.
- It happens when we have very less amount of data to build an accurate model or when we try to build a linear model with a nonlinear data. Also, these kind of models are very simple to capture the complex patterns in data like Linear and logistic regression.
- Here Training Accuracy is 50% But Testing Accuracy is around 49%. See there is less accuracy this tends to underfit.
- Underfitting is when the model's error on both the training and test sets (i.e. during training and testing) is very high.

How to Oercome Underfitting :

- In that case, there are 2 gold standard approaches:

```
1. Try another model
2. Increase the complexity of the current model
```

Solution 1 is trivial. Concerning solution 2, an example an be the following: if someone is fitting a linear regression to some data, then increasing the complexity would mean to fit a polynomial model.

Overfit :

- These models have low bias and high variance.
- overfitting happens when our model captures the noise along with the underlying pattern in data.
- It happens when we train our model a lot over the noisy dataset.
- These models are very complex like Decision trees which are prone to overfitting.

- - Here Training Accuracy is 99% But Testing Accuracy is around 50%. there is huge difference between training and testing score this tends to overfit
- Overfitting is when the model's error on the training set (i.e. during training) is very low but then, the model's error on the test set (i.e. unseen samples) is large!

How to overcome Overfitting :

- The most common problem in the ML learning filed is overfitting.
- Action that could (potentially) limit overfitting:

```
1. We can use a Cross-validation (CV) scheme.

2. Reduce the complexity of the model (make the model less complex).
```

- When it comes to solution 1 i.e. the use of cross-validation, the most famous CV scheme is the KFolds cross-validation. Using a KFolds scheme, we train and test your model k-times on different subsets of the training data and estimate a performance metric using the test (unseen) data.
- When it comes to solution 2 i.e. reducing the complexity of the model can help reduce the overfitting. For example, if someone is using an SVM model with RBF kernel then reducing the complexity would mean to use a linear kernel. In another case, if someone is fitting a polynomial to some data, then reducing the complexity would mean to fit a linear model instead (linear regression).

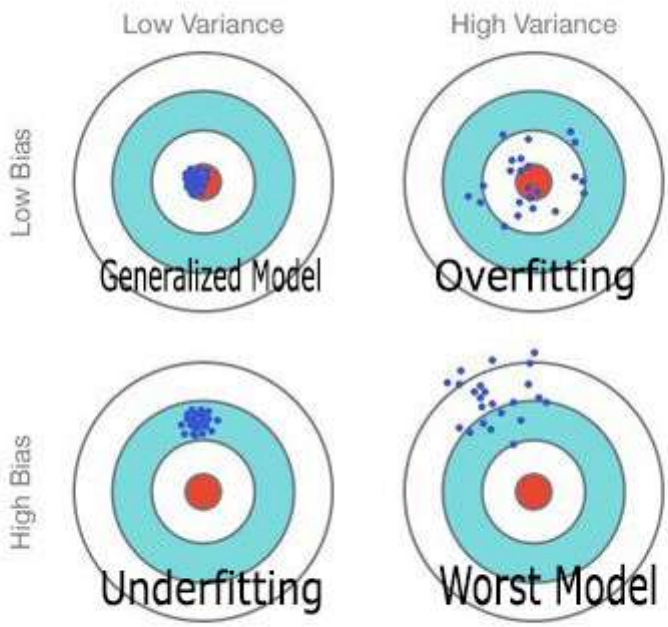


Fig. 1: Graphical Illustration of bias-variance trade-off , Source: Scott Fortmann-Roe., Understanding Bias-Variance Trade-off

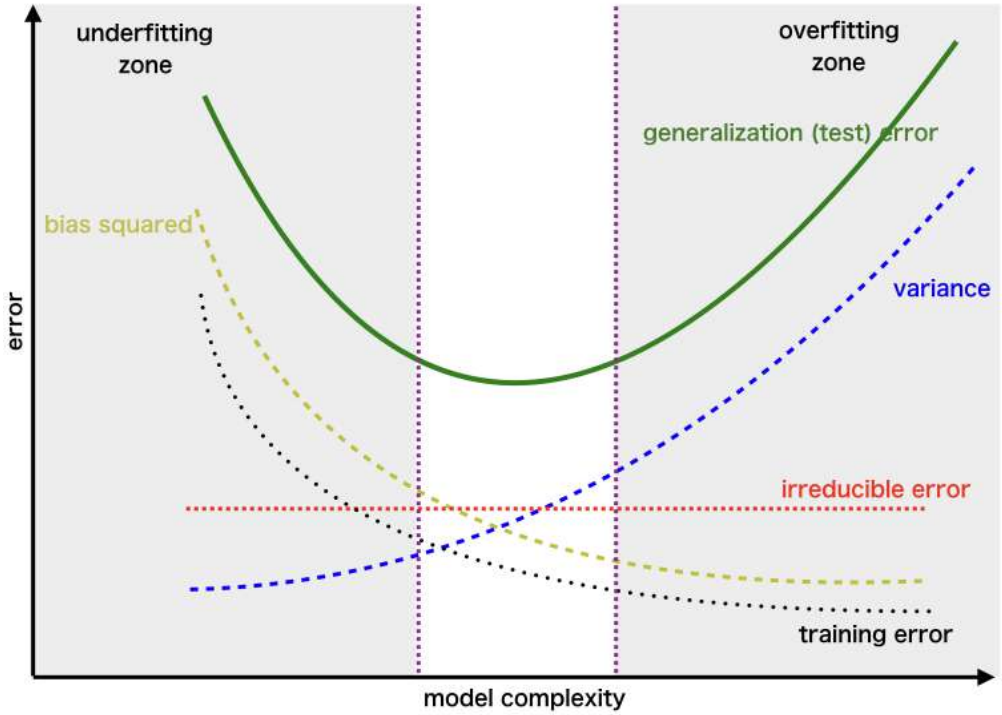
What is Bias-Variance Tradeoff ?

- If our model is too simple and has very few parameters then it may have high bias and low variance.
- On the other hand, if our model has a large number of parameters then it’s going to have high variance and low bias.
- So we need to find the right/good balance without overfitting and underfitting the data.

- There is no escaping the relationship between bias and variance in machine learning.

```
* Increasing the bias will decrease the variance.
* Increasing the variance will decrease the bias.
```

- If the algorithm is too simple then it may be on high bias and low variance condition and thus is error-prone. If algorithms fit too complex then it may be on high variance and low bias. In the latter condition, the new entries will not perform well. Well, there is something between both of these conditions, known as Trade-off or Bias Variance Trade-off.
- The error to complexity graph to show trade-off is given as –



.

The above picture shows that :

- Bias initially decreases faster than variance.
- After a point, variance increases significantly with an increase in flexibility with little impact on bias.

This is referred to as the best point chosen for the training of the algorithm which gives low error in training as well as testing data.

Hyperparameter Tuning :

- It is the main part when you build your machine learning model. As all you know in ml model there are overfiting concept so to reduce this overfit we use hyperparameter tuning.
- If we train a linear regression with SGD,parameters of a model are the slope and the bias and hyperparameter is learning rate.
- Some examples of model hyperparameters include:

```
- The C and sigma hyperparameters for support vector machines.
- The k in k-nearest neighbors.
- Max_Depth in Decision tree
```

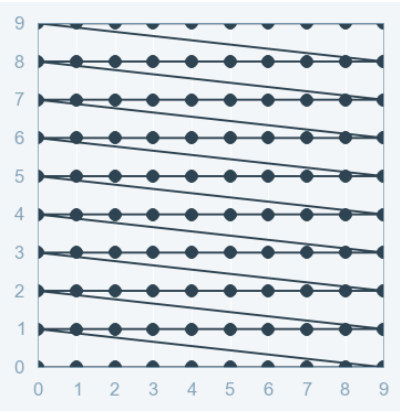
To perform this we have two strategy

```
1. GridSearchCV
2. RandomizedSearchCV
```

- Here i use CV(Cross validation) term in end of every hyperparameter approach because here cross validation also performed.

1. GridSearchCV

- In GridSearchCV approach, machine learning model is evaluated for a range of hyperparameter values. This approach is called GridSearchCV, because it searches for best set of hyperparameters from a grid of hyperparameters values.
- In Grid Search, we try every combination of a preset list of values of the hyper-parameters and evaluate the model for each combination.



Implementation

It Include 4 steps:

1. **initialize all parameter**
2. **Instantiating Ml Model**
3. **Instantiating the GridSearchCV object**
4. **Print the tuned parameters and score**

Here I am taking only Logistic Regression "C" value for hyperparameter tuning. you can take any model.

```
# Necessary imports
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
import numpy as np
```

```
# initialize all parameter
param_grid = {'C': np.logspace(-5, 8, 15) }
```

```
# Instantiating logistic regression classifier
logreg = LogisticRegression()
```

```
# Instantiating the GridSearchCV object
logreg_cv = GridSearchCV(logreg, param_grid, cv = 5)
logreg_cv.fit(X, y) #here always put all data not train data
```

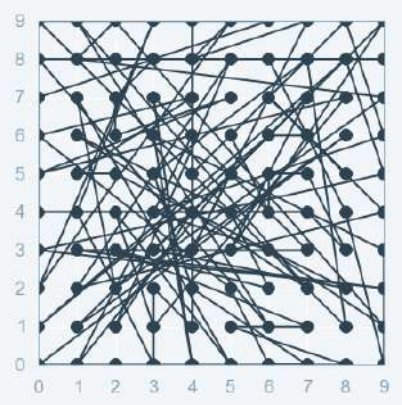
```
# Print the tuned parameters and score
print("Tuned Logistic Regression Parameters: {}".format(logreg_cv.best_params_))
print("Best score is {}".format(logreg_cv.best_score_))
```

Drawbacks Of Gridsearch Cv

- GridSearchCV will go through all the intermediate combinations of hyperparameters which makes grid search computationally very expensive.
- Now i dicuss about Random Search Cv which is faster than grid search cv

2. Random Search Cv

- RandomizedSearchCV solves the drawbacks of GridSearchCV, as it goes through only a fixed number of hyperparameter settings. It moves within the grid in random fashion to find the best set hyperparameters.
- Random search is a technique where random combinations of the hyperparameters are used to find the best solution for the built model.



- Code is same as GridSearch Cv , here instaed of Gridsearch we use Random Search

Implementation

It Include 4 steps:

1. **initialize all parameter**
2. **Instantiating Ml Model**
3. **Instantiating the GridSearchCV object**
4. **Print the tuned parameters and score**

Here I use decision tree parameter for tuning

```
# Necessary imports
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV
```

```
# initialize all parameter
param_grid = {
    "max_depth" : [50,100,150,200],
    "min_samples_split" : [1,2,3,4,5,6,7,8,9],
    "min_samples_leaf" : [1,2,3,4,5,6,7,8,9],
    "criterion": ["gini", "entropy"],
    "max_leaf_nodes":[1,5,10,15,20]
        }
```

```
# Instantiating logistic regression classifier
classifier = DecisionTreeClassifier()
```

```
# Instantiating the GridSearchCV object
classifier_cv =  RandomizedSearchCV(classifier, param_grid, cv = 5)
classifier_cv.fit(X, y) #here always put all data not train data
```

```
# Print the tuned parameters and score
print("Tuned Decision Tree Parameters: {}".format(classifier_cv.best_params_))
print("Best score is {}".format(classifier_cv.best_score_))
```

Cross Validation

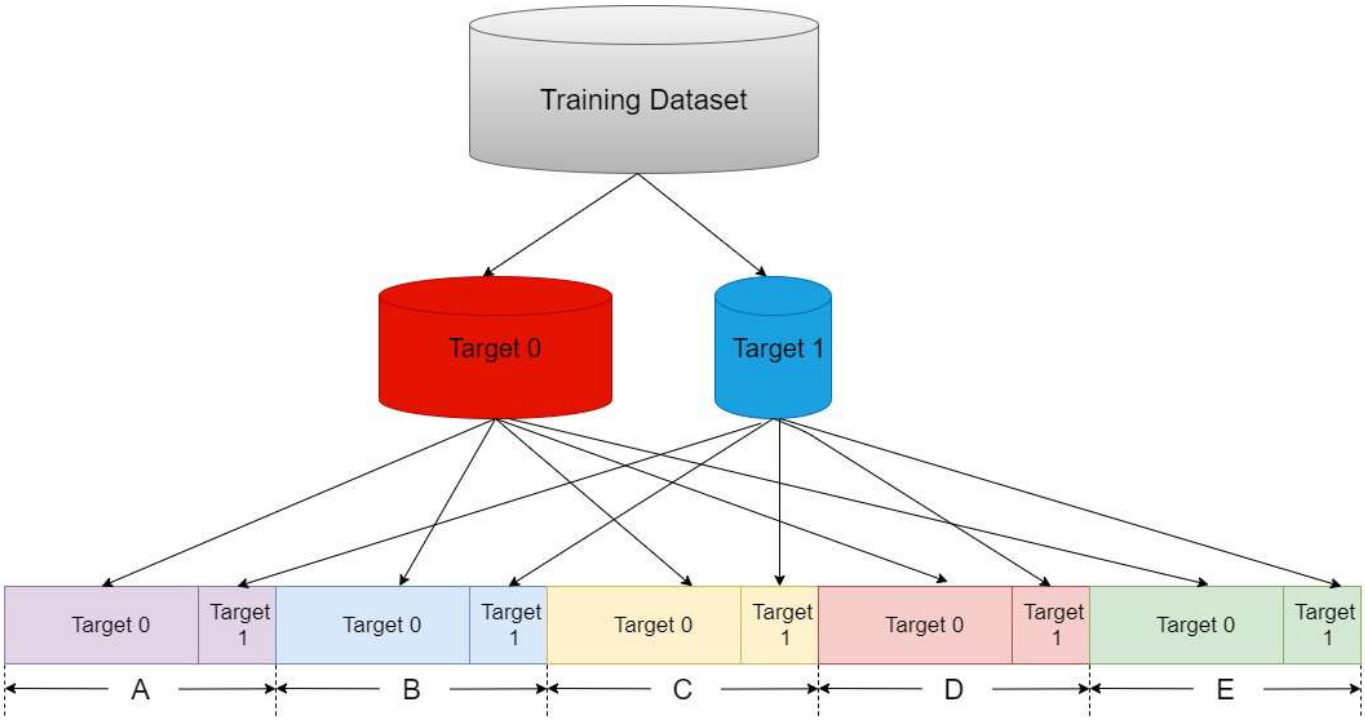
- Cross-validation is a technique for evaluating ML models by training several ML models on subsets of the available input data and evaluating them on the complementary subset of the data. Use cross-validation to detect overfitting, ie, failing to generalize a pattern.
- The three steps involved in cross-validation are as follows :

```
1. Reserve some portion of sample data-set.
2. Using the rest data-set train the model.
3. Test the model using the reserve portion of the data-set.
```

Here I Only Dicusss about Stratified k-fold cross validation

Stratified k-fold cross validation

- Stratified k-fold cross-validation is same as just k-fold cross-validation, But in Stratified k-fold cross-validation, it does stratified sampling instead of random sampling.
- One obvious problem with normal KFold, is that each in each fold the distribution of classes in the validation set, will be not be same. This is a big problem with imbalanced datasets.
- To overcome this problem we will use Stratified-KFold Validation. StratifiedKFold ensures that each of the splits have same proportion of examples of each class.
- StratifiedKFold is a variation of KFold. First, StratifiedKFold shuffles your data, after that splits the data into n_splits parts and Done. Now, it will use each part as a test set. Note that it only and always shuffles data one time before splitting.



Demonstration

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import confusion_matrix, roc_auc_score ,roc_curve,auc
from sklearn.model_selection import StratifiedKFold
import warnings
warnings.simplefilter('ignore')
```

In [2]:

```
cancer = load_breast_cancer()
df = pd.DataFrame(cancer.data, columns=cancer.feature_names)
df['target'] = pd.Series(cancer.target)
df.head()
```

Out[2]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mear symmetry
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.241
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.181
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.206
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.259
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.180

5 rows × 31 columns

In [3]:

```
X = df.drop('target',axis=1)
y = df['target'].astype('category')
```

Use manual train_test_split

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=0.2)
```

In [5]:

```
lr_manual = LogisticRegression()
lr_manual.fit(X_train,y_train)
```

Out[5]:

```
LogisticRegression()
```

In [6]:

```
confusion_matrix(y_test,lr_manual.predict(X_test))
```

Out[6]:

```
array([[39,  5],
       [ 0, 70]], dtype=int64)
```

Use StratifiedKFold

In [7]:

```
kf = StratifiedKFold(n_splits=5,shuffle=True,random_state=45)
pred_test_full =0
cv_score =[]
i=1
for train_index,test_index in kf.split(X,y):
    print('{} of KFold {}'.format(i,kf.n_splits))

    ### Training Set
    xtr,xvl = X.iloc[train_index],X.iloc[test_index]

    ### Validation Set
    ytr,yvl = y.iloc[train_index],y.iloc[test_index]

    #model
    lr = LogisticRegression(C=2)
    lr.fit(xtr,ytr)
    score = roc_auc_score(yvl,lr.predict(xvl))
    print('ROC AUC score:',score)
    cv_score.append(score)

#     pred_test = lr.predict_proba(x_test)[:,1]
#     pred_test_full +=pred_test
    i+=1
```

```
1 of KFold 5
ROC AUC score: 0.9415329184408779
2 of KFold 5
ROC AUC score: 0.932361611529643
3 of KFold 5
ROC AUC score: 0.9523809523809523
4 of KFold 5
ROC AUC score: 0.9692460317460316
5 of KFold 5
ROC AUC score: 0.9312541918175722
```

In [8]:

```
print('Confusion matrix\n',confusion_matrix(yvl,lr.predict(xvl)))
print('Cv',cv_score,'\nMean cv Score',np.mean(cv_score))
```

```
Confusion matrix
 [[38  4]
 [ 3 68]]
Cv [0.9415329184408779, 0.932361611529643, 0.9523809523809523, 0.96924603174
60316, 0.9312541918175722]
Mean cv Score 0.9453551411830154
```

here I use logistic regression for demonstrate the k-fold. you can use any algorithm.

Metrics for Machine learning model

- Evaluation is always good in any field right! In the case of machine learning, it is best the practice. In this post, I will almost cover all the popular as well as common metrics used for machine learning.
- Different performance metrics are used to evaluate different Machine Learning Algorithms. For now, we will be focusing on the ones used for Classification problems.

The Classification Metric Are:

1. Confusion Matrix
2. Accuracy
3. Precision
4. Recall
5. Specificity
6. F1-Score
7. Roc And Auc

1. Confusion Matrix :

- The Confusion matrix is one of the most intuitive and easiest metrics used for finding the correctness and accuracy of the model. It is only used for Classification problem where the output can be of two or more types of classes.

There are 4 terms you should keep in mind:

```
    1. True Positives (TP): It is the case where we predicted Yes and the real out
put was also yes.
    2. True Negatives (TN): It is the case where we predicted No and the real outp
ut was also No.
    3. False Positives (FP): It is the case where we predicted Yes but it was actu
ally No.
    4. False Negatives (FN): It is the case where we predicted No but it was actua
lly Yes.
```

Lets Take an example to know better this 4 term :

Let’s give a label of to our target variable:

1: When a person is having Corona

0: When a person is NOT having Corona.

Now lets apply this corona example in to this 4 term:

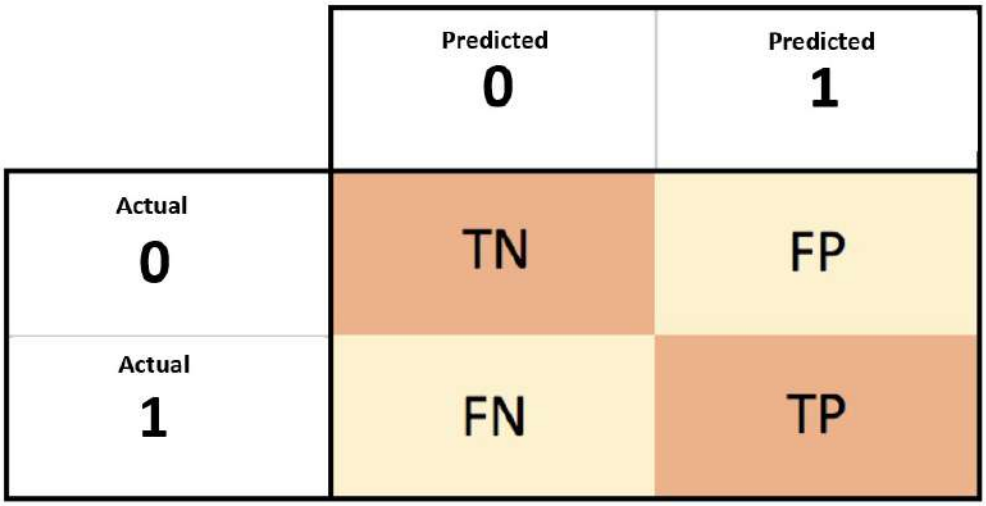
1. True Positives (TP) :- Ex: The case where a person is actually having Corona(1) and the model also predict person haing Corona(1) Then it comes under True positive(TP).

2. True Negatives (TN) :- Ex: The case where a person NOT having Corona(0) and the model also prdict person Not Having Corona(0) Then it comes under True Negatives(TN).

3. False Positives (FP) :- Ex: A person NOT having Corona(0) But the model is predict Person haing Corona(1) Then It comes under False Positives (FP).

4. False Negatives (FN) :- Ex: A person having Corona (1) But the model predict Person not Haing Corona (0) Then it comes under False Negatives(FN).

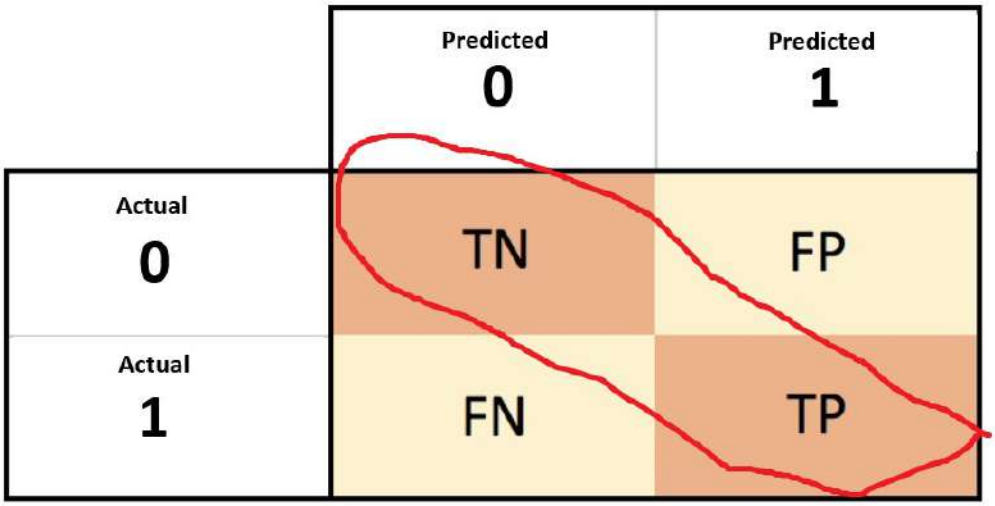
SO in the above example you see these 4 type of term associate with Confusion Matrix.



- we all want is that the model should give 0 False Positives and 0 False Negatives.
- But that's not the case in real life as any model will NOT be 100% accurate most of the times.
- So we might want to minimise either False Positives or False negatives.

2. Accuracy :

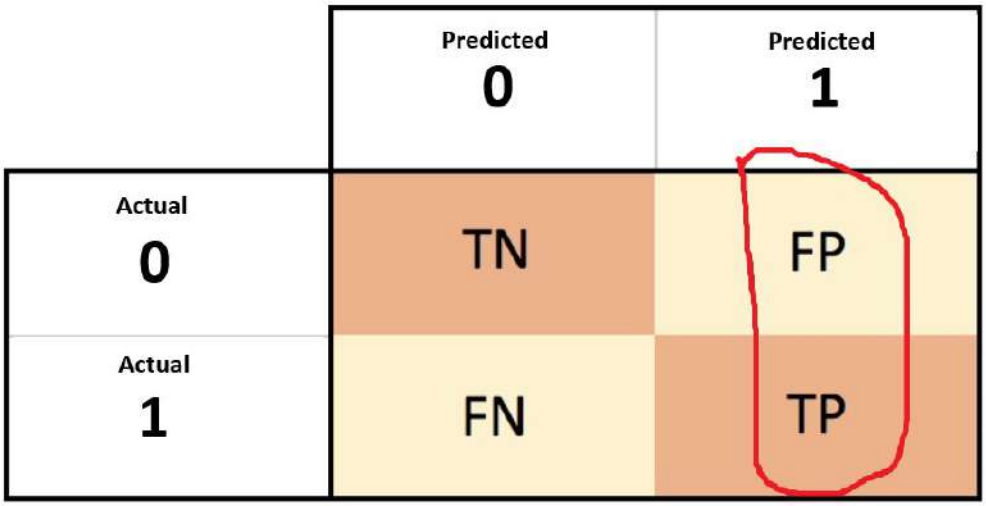
- Accuracy in classification problems is the number of correct predictions made by the model over all kinds predictions made.



$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

- Accuracy is a good measure when the target variable classes in the data are nearly balanced.
- Accuracy should NEVER be used as a measure when the target variable classes in the data are a majority of one class. (imbalanced dataset)

3. Precision :

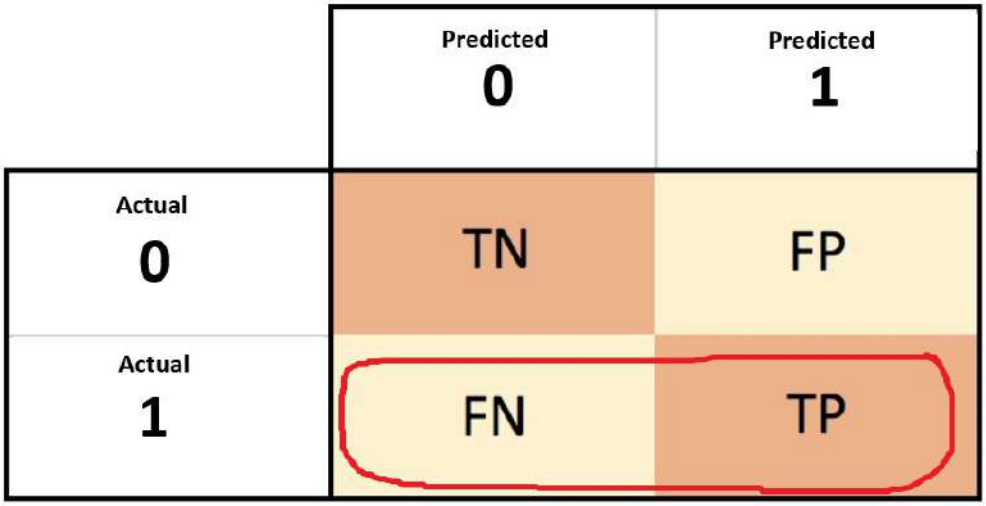


$$\text{Precision} = \frac{TP}{TP + FP}$$

- Precision is a measure that tells us what proportion of patients that we diagnosed as having Corona, actually had Corona. The predicted positives (Total People predicted as Corona are TP and FP) and the people actually having a Corona are TP.

- precision gives us information about its performance with respect to false positives(how many did we caught).
- Precision is about being precise. So even if we managed to capture only one Corona case, and we captured it correctly, then we are 100% precise.

4. Recall or Sensitivity:

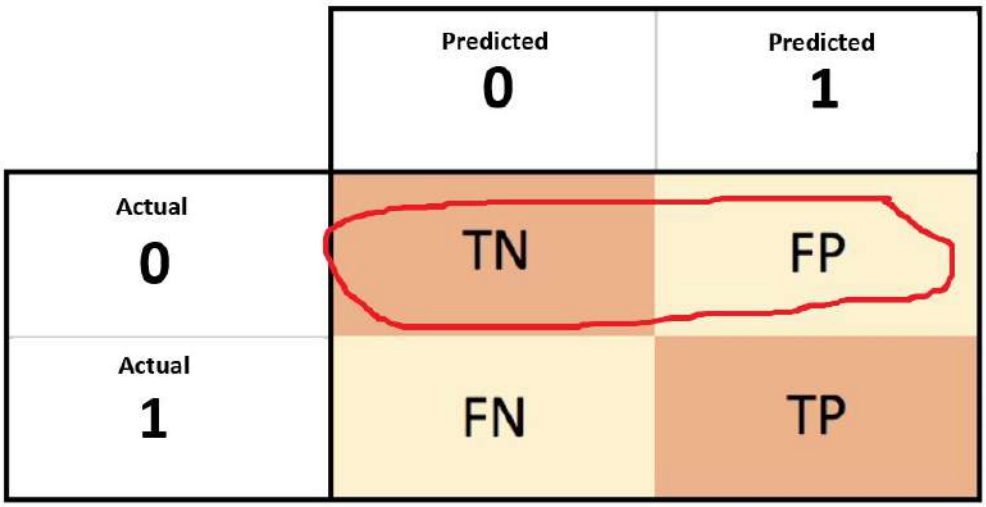


$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

- Recall is a measure that tells us what proportion of patients that actually had Corona was diagnosed by the algorithm as having Corona. The actual positives (People having Corona are TP and FN) and the people diagnosed by the model having a Corona are TP. (Note: FN is included because the Person actually had a Corona even though the model predicted otherwise).

- It is clear that recall gives us information about a classifier's performance with respect to false negatives (how many did we miss).
- Recall is not so much about capturing cases correctly but more about capturing all cases that have "Corona" with the answer as "Corona". So if we simply always say every case as "Corona", we have 100% recall.

5. Specificity :

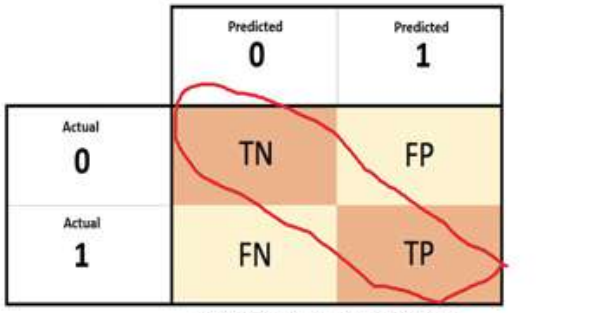


$$\text{Specificity} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}$$

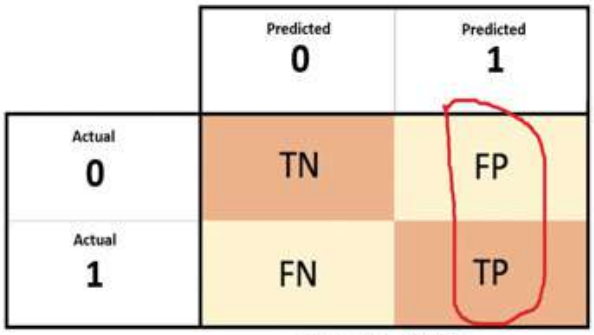
- Specificity is the exact opposite of Recall.
- Specificity is a measure that tells us what proportion of patients that did NOT have Corona, were predicted by the model as not Corona . The actual negatives (People actually NOT having Corona are FP and TN) and the people diagnosed by us not having Corona are TN. (Note: FP is included because the Person did NOT actually have Corona even though the model predicted otherwie).

All 4 Metric :

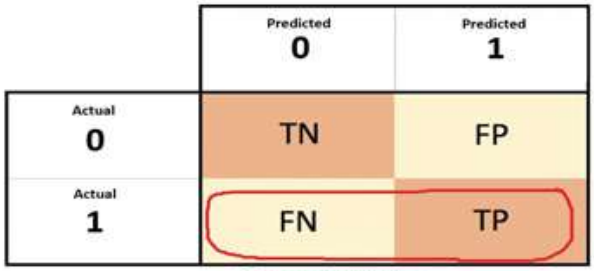
1. Accuracy 2. Precision 3. Recall or Sensitivity 4.Specificity



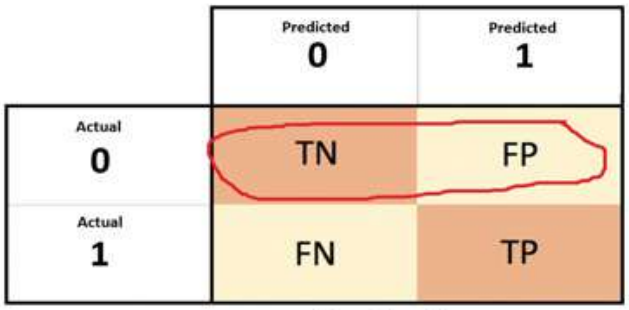
$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$



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



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



$$Specificity = \frac{True\ Negatives}{True\ Negatives + False\ Positives}$$

f1-score and Roc Auc i am explain in next notebook.

F1-Score

- F1-score is a metric which takes into account both ,precision and recall as we can't always evaluate both and then take the higher one for our model. It is the harmonic mean of precision and recall. It tells us about the balance that exists between precision and recall.
- F1-score: This is the harmonic mean of Precision and Recall and gives a better measure of the incorrectly classified cases than the Accuracy Metric.

$$\text{F1-score} = \left(\frac{\text{Recall}^{-1} + \text{Precision}^{-1}}{2}\right)^{-1} = 2 * \frac{(\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}$$

We use the Harmonic Mean since it penalizes the extreme values.

When It used:

- F1-score is used when the False Negatives and False Positives are crucial.
- F1-score is a better metric when there are imbalanced classes.
- In most real-life classification problems, imbalanced class distribution exists and thus F1-score is a better metric to evaluate our model on.

Some advantages of F1-score:

- Very small precision or recall will result in lower overall score. Thus it helps balance the two metrics.
- If you choose your positive class as the one with fewer samples, F1-score can help balance the metric across positive/negative samples.
- As illustrated by the first figure in this article, it combines many of the other metrics into a single one, capturing many aspects at once.

F-β score:

- The F-beta score is the weighted harmonic mean of precision and recall, reaching its optimal value at 1 and its worst value at 0. The beta parameter determines the weight of recall in the combined score.

The general formula for non-negative real β is:

$$F_\beta = \frac{(1 + \beta^2) \cdot (\text{precision} \cdot \text{recall})}{(\beta^2 \cdot \text{precision} + \text{recall})}$$

- Here Three Condition ariese

1. β=1 (F1 score) - When FP And FN are equally important.
2. β=0.5 (F0.5 score) - when importance of FP is greater than FN.

3. β=2 (F2 score) - When FN imapct is high we increase β value 1 to 2.

ROC And AUC Curve

- Befor Going to ROC and AUC i discuss about main 4 term

1. TPR (Recall)
2. TNR (Specificity)
3. FPR (1-specificity)
4. FNR

$$TPR = \frac{TP}{Actual\ Positive} = \frac{TP}{TP + FN}$$

$$FNR = \frac{FN}{Actual\ Positive} = \frac{FN}{TP + FN}$$

$$TNR = \frac{TN}{Actual\ Negative} = \frac{TN}{TN + FP}$$

$$FPR = \frac{FP}{Actual\ Negative} = \frac{FP}{TN + FP}$$

- TPR (True Positive Rate): In machine learning, the true positive rate, also referred to sensitivity or recall, is used to measure the percentage of actual positives which are correctly identified.
- TNR (True Negative Rate) : The Specificity of a test, also referred to as the true negative rate (TNR), is the proportion of samples that test negative using the test in question that are genuinely negative.
- FPR (False Positive Rate) : In statistics, when performing multiple comparisons, a false positive ratio (also known as fall-out or false alarm ratio) is the probability of falsely rejecting the null hypothesis for a particular test. The false positive rate is calculated as the ratio between the number of negative events wrongly categorized as positive (false positives) and the total number of actual negative events (regardless of classification).
- FNR (False Negative Rate) : The rate of occurrence of negative test results in those who have the attribute or disease for which they are being tested.

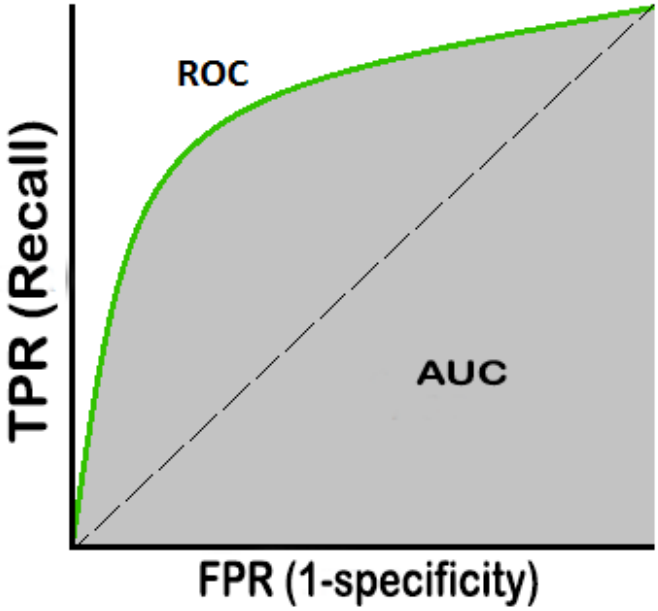
Now i am going to discuss about ROC And AUC

What is ROC curve?

- ROC curve is one the important evaluating metrics that should be used to check the performance of an classification model.It is also called relative operating characteristic curve, because it is a comparison of two main characteristics (TPR and FPR).
- It is plotted between sensitivity(recall) and False Positive Rate (FPR = 1-specificity).

What is AUC?

- AUC also called as AREA UNDER CURVE.It is used in classification analysis in order to determine which of the used models predicts the classes best. An example of its application are ROC curves.
- AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0 and if the predictions are 100% correct has an AUC of 1.



implementation Of ROC And AUC

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:

```
url="https://raw.githubusercontent.com/Suji04/Diabetes-Detection/master/diabetes.csv"
data = pd.read_csv(url)
data.head(5)
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28

In [3]:

```
X = data.iloc[:,0:-1].values
y = data.iloc[:,-1].values
```

Train-Test Split

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 =
```

Feature Scling

In [5]:

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

Classifier Model

A. Logistic Regrssion

In [6]:

```
from sklearn.linear_model import LogisticRegression
model_logistic = LogisticRegression(C=2)
model_logistic.fit(X_train, y_train)

y_pred_logistic = model_logistic.decision_function(X_test)
```

B. DecisionTreeClassifier

In [7]:

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=50)
dt.fit(X_train,y_train)

y_pred_dt = dt.predict_proba(X_test)
```

B. RandomForestClassifier

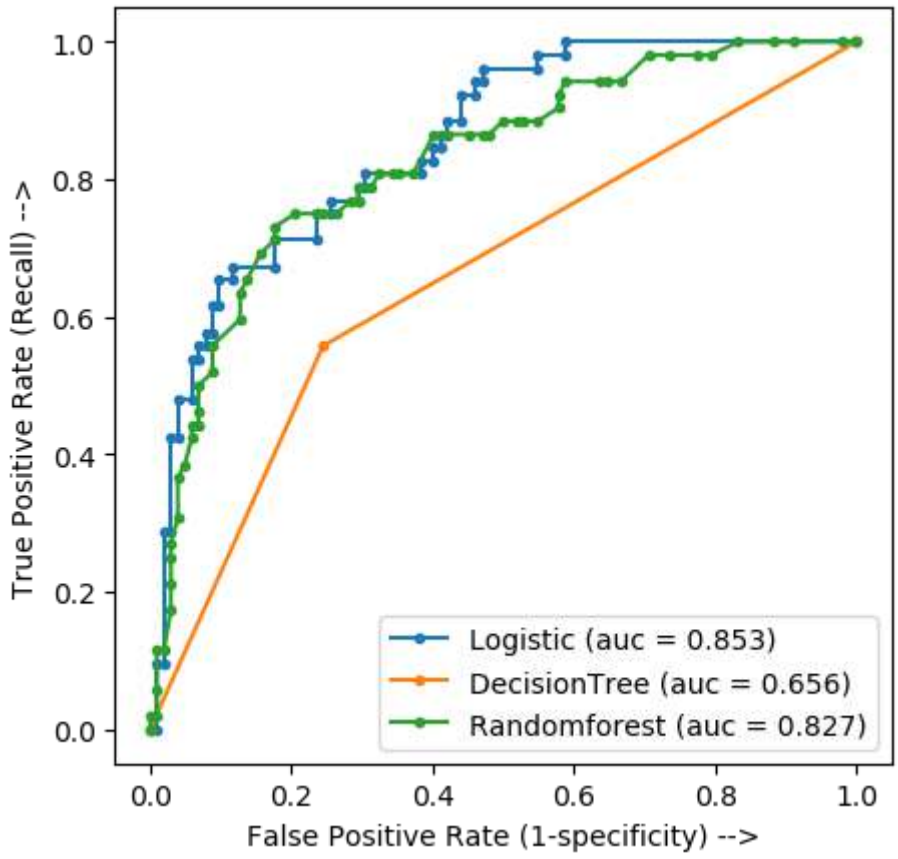
In [8]:

```
from sklearn.ensemble import RandomForestClassifier
rfr = RandomForestClassifier(n_estimators=100)
rfr.fit(X_train,y_train)

y_pred_rfr = rfr.predict_proba(X_test)
```

Find Roc And Auc Score Of These three model

In [9]:

```
from sklearn.metrics import roc_curve, auc

logistic_fpr, logistic_tpr, threshold = roc_curve(y_test, y_pred_logistic)
auc_logistic = auc(logistic_fpr, logistic_tpr)

dt_fpr, dt_tpr, threshold = roc_curve(y_test, y_pred_dt[:,1:2])
auc_dt = auc(dt_fpr, dt_tpr)

rfr_fpr, rfr_tpr, threshold = roc_curve(y_test, y_pred_rfr[:,1:2])
auc_rfr = auc(rfr_fpr, rfr_tpr)

# svm_fpr, svm_tpr, threshold = roc_curve(y_test, y_pred_svm)
# auc_svm = auc(svm_fpr, svm_tpr)

plt.figure(figsize=(5, 5), dpi=100)
# plt.plot(svm_fpr, svm_tpr, linestyle='-', label='SVM (auc = %0.3f)' % auc_svm)
plt.plot(logistic_fpr, logistic_tpr, marker='.', label='Logistic (auc = %0.3f)' % auc_logis
plt.plot(dt_fpr, dt_tpr, marker='.', label='DecisionTree (auc = %0.3f)' % auc_dt)
plt.plot(rfr_fpr, rfr_tpr, marker='.', label='Randomforest (auc = %0.3f)' % auc_rfr)

plt.xlabel('False Positive Rate (1-specificity) -->')
plt.ylabel('True Positive Rate (Recall) -->')

plt.legend()

plt.show()
```



- Here you see that 3 model has different Auc score. so here we consider which has higher auc score

Regression Model Metric

- In my Previous notebook i am focussed on classification metrics like precision, recall, AUC etc.
- For a change, I wanted to explore all kinds of metrics including those used in regression as well. MAE and RMSE are the two most popular metrics for continuous variables. Let's start with the more popular one.
- All Types Of Metric Used In Regression

```
1. Mean Squared Error(MSE)
2. Root-Mean-Squared-Error(RMSE).
3. Mean-Absolute-Error(MAE).
4. Root Mean Squared Logarithmic Error (RMSLE).
5. R² or Coefficient of Determination.
6. Adjusted R²
```

1. Mean Squared Error(MSE) :

- MSE or Mean Squared Error is one of the most preferred metrics for regression tasks.
- It is simply the average of the squared difference between the target value and the value predicted by the regression model.
- As it squares the differences, it penalizes even a small error which leads to over-estimation of how bad the model is. It is preferred more than other metrics because it is differentiable and hence can be optimized better.

$$\underbrace{\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}}_{\text{test set}} \underbrace{(y_i}_{\text{predicted vaue}} - \underbrace{\hat{y}_i)^2}_{\text{actual value}}$$

2. Root Mean Squared Error (RMSE) :

- It is simple root of MSE.
- RMSE is the most widely used metric for regression tasks and is the square root of the averaged squared difference between the target value and the value predicted by the model.
- It is preferred more in some cases because the errors are first squared before averaging which poses a high penalty on large errors. This implies that RMSE is useful when large errors are undesired.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

3. Mean-Absolute-Error(MAE) :

- MAE is the absolute difference between the target value and the value predicted by the model.
- The MAE is more robust to outliers and does not penalize the errors as extremely as mse. MAE is a linear score which means all the individual differences are weighted equally. It is not suitable for applications where you want to pay more attention to the outliers.

$$\text{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

4. Root Mean Squared Logarithmic Error (RMSLE) :

- In case of Root mean squared logarithmic error, we take the log of the predictions and actual values. So basically, what changes are the variance that we are measuring.
- RMSLE is usually used when we don't want to penalize huge differences in the predicted and the actual values when both predicted and true values are huge numbers.

$$\text{RMSLE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\log(y_i+1) - \log(\hat{y}_i+1))^2}$$

All Four Metric

$$\text{MSE} = \underbrace{\frac{1}{n}\sum_{i=1}^{n}}_{\text{test set}}\underbrace{(y_i}_{\text{predicted vaue}} - \underbrace{\hat{y}_i)^2}_{\text{actual value}}$$

$$\text{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

$$\text{RMSLE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\log(y_i+1) - \log(\hat{y}_i+1))^2}$$

5. R^2 or Coefficient of Determination:

- Coefficient of Determination or R^2 is another metric used for evaluating the performance of a regression model.
- The metric helps us to compare our current model with a constant baseline and tells us how much our model is better.
- The constant baseline is chosen by taking the mean of the data and drawing a line at the mean. R^2 is a scale-free score that implies it doesn't matter whether the values are too large or too small, the R^2 will always be less than or equal to 1.

Sum Squared Regression Error

$$R^2 = 1 - \frac{SS_{Regression}}{SS_{Total}}$$

Sum Squared Total Error

6. Adjusted R^2:

- it is suitable for multiple linear regrssion where more than one independent variable.
- Adjusted R^2 depicts the same meaning as R^2 but is an improvement of it.
- R^2 suffers from the problem that the scores improve on increasing terms even though the model is not improving which may misguide the researcher.
- Adjusted R^2 is always lower than R^2 as it adjusts for the increasing predictors and only shows improvement if there is a real improvement.

$$R^2_{adjusted} = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

where

R^2 = sample R-square

p = Number of predictors

N = Total sample size.

Q. Why is R^2 Negative ?

- There is a misconception among people that R^2 score ranges from 0 to 1 but actually it ranges from $-\infty$ to 1. Due to this misconception, they are sometimes scared why the R^2 is negative which is not a possibility according to them.

The main reasons for R^2 to be negative are the following:

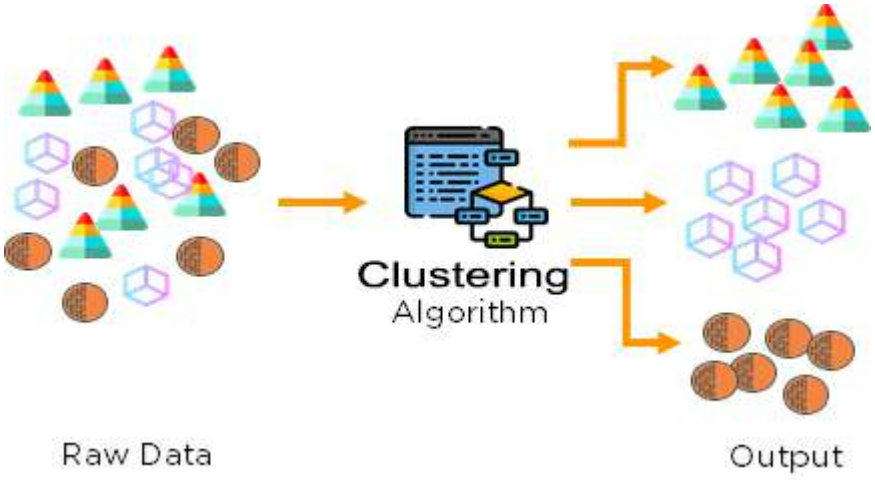
- R^2 compares the fit of the chosen model with that of a horizontal straight line (the null hypothesis). If the chosen model fits worse than a horizontal line, then R^2 is negative.

- Note that R^2 is not always the square of anything, so it can have a negative value without violating any rules of math.
- Mainly R^2 is negative only when the chosen model does not follow the trend of the data, so fits worse than a horizontal line.
- Maybe their area a large number of outliers in the data that causes the mse of the model to be more than mse of the baseline causing the R^2 to be negative(i.e the numerator is greater than the denominator).

In this Notebook, i discovered about the various metrics used in regression analysis and also tried to answer the question of why R^2 is negative?

Clustering

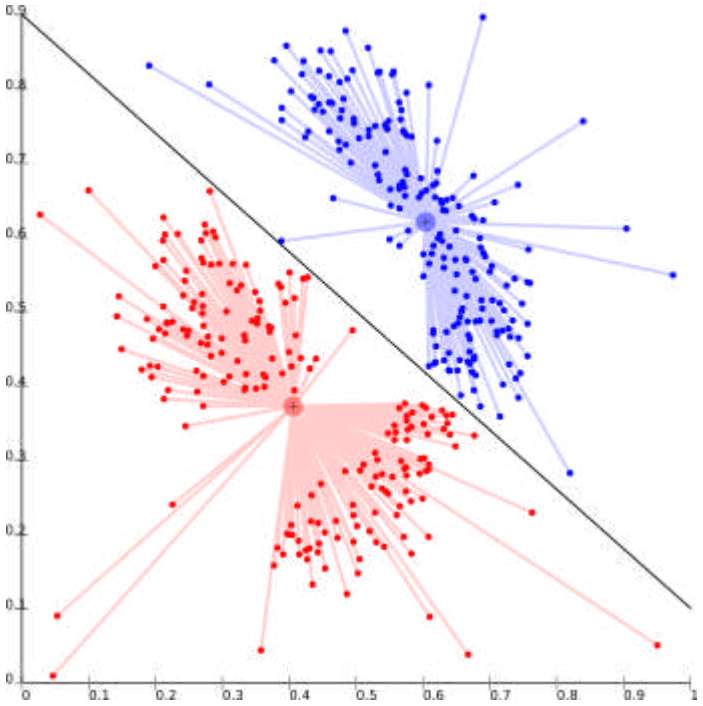
- It is basically a type of unsupervised learning method . An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labelled responses.
- Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.
- Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups.



Types Of Clustering :

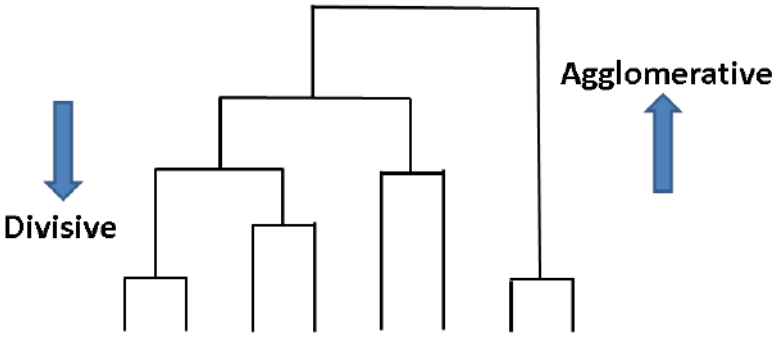
1. Centroid-based clustering:

- This is basically one of iterative clustering algorithm in which the clusters are formed by the closeness of data points to the centroid of clusters.
- Here , the cluster center i.e. centroid is formed such that the distance of data points is minimum with the center.
- Data is grouped into centroids based on how close they are to the center of the centroids.
- For Ex- K – means algorithm is one of popular example of this algorithm.



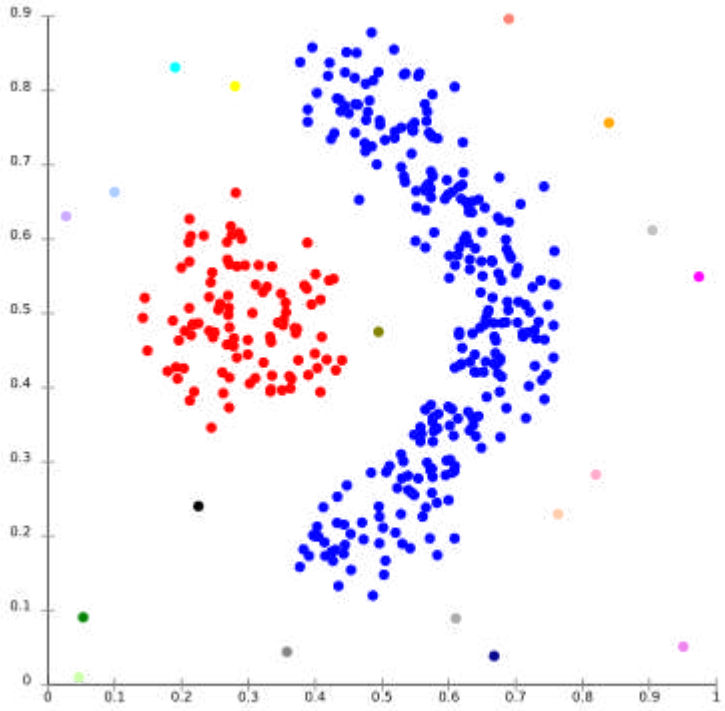
2. Hierarchical clustering:

- When you want machine to find the right number of clusters. Each data item is considered as a cluster and then data items are grouped together based on their distance recursively until optimum clusters of data are calculated.
- Algorithms include Agglomerative clustering/ hierarchical clustering.



3. Density Models:

- In this clustering model there will be a searching of data space for areas of varied density of data points in the data space .
- For Ex- DBSCAN and OPTICS.



- There are also other clustering algorithm types such as distribution based clustering which uses underlying probability distribution to group data into clusters. refer this: https://www.geeksforgeeks.org/different-types-clustering-algorithm/?ref=rp (https://www.geeksforgeeks.org/different-types-clustering-algorithm/?ref=rp)

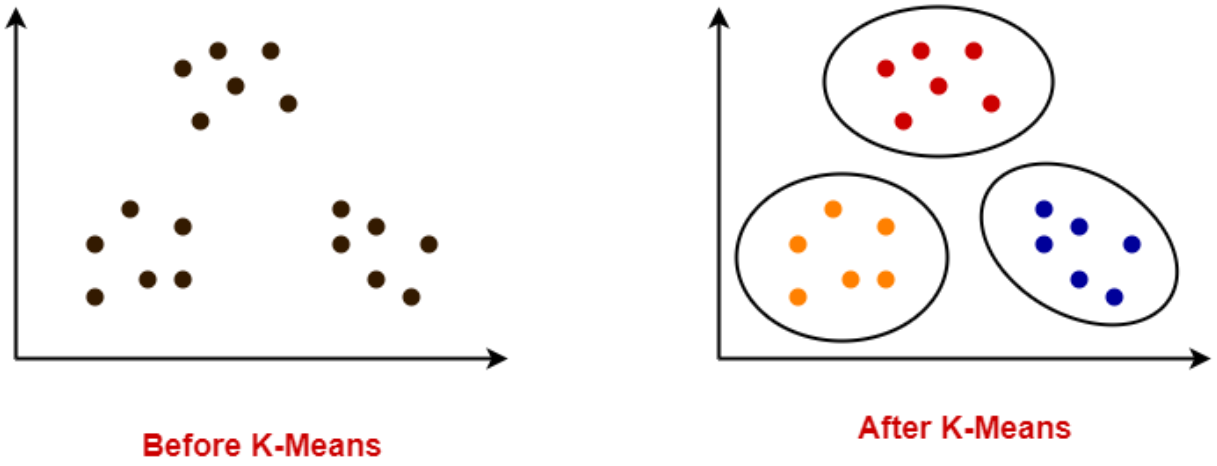
Applications of Clustering in different fields :

1. Identifying Fake News
2. Spam filter
3. Marketing and Sales
4. Classifying network traffic
5. Identifying fraudulent or criminal activity
6. Earthquake studies etc

K-Means Clustering

It was proposed by Stuart Lloyd at the Bell Labs in 1957 as a technique for pulse-code modulation, but it was only published outside of the company in 1982, in a paper titled _“Least square quantization in PCM”._By then, in 1965, Edward W. Forgy had published virtually the same algorithm, so K-Means is sometimes referred to as Lloyd-Forgy.

K-Means is a clustering approach in which the data is grouped into K distinct non-overlapping clusters based on their distances from the K centres. The value of **K** needs to be specified first and then the algorithm assigns the points to exactly one cluster.



Theory

The theory discussed above can be mathematically expressed as:

- Let C1, C2, Ck be the K clusters
- Then we can write: $C_1 U C_2 U C_3 U \dots U C_k$ ={1,2,3,…,n} i.e., each datapoint has been assigned to a cluster.
- Also,

$$C_k \cap C_{k'} = \emptyset \text{ for all } k \neq k'.$$

  This means that the clusters are non-overlapping.
- The idea behind the K-Means clustering approach is that the within-cluster variation amongst the point should be minimum. The within-cluster variance is denoted by: W(Ck).Hence, according to the statement above, we need to minimize this variance for all the clusters. Mathematically it can be written as:

$$\underset{C_1,\dots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} W(C_k) \right\}.$$

- The next step is to define the criterion for measuring the within-cluster variance. Generally, the criterion is the Euclidean distance between two data points.

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2,$$

- The above formula says that we are calculating the distances between all the point in a cluster, then we are repeating it for all the K clusters(That's why two summation signs) and then we are dividing it by the number of observation in the clusters (Ck is the number of observations in the Kth cluster) to calculate the average.

So, ultimately our goal is to minimize:

$$\underset{C_1,\ldots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \right\}.$$

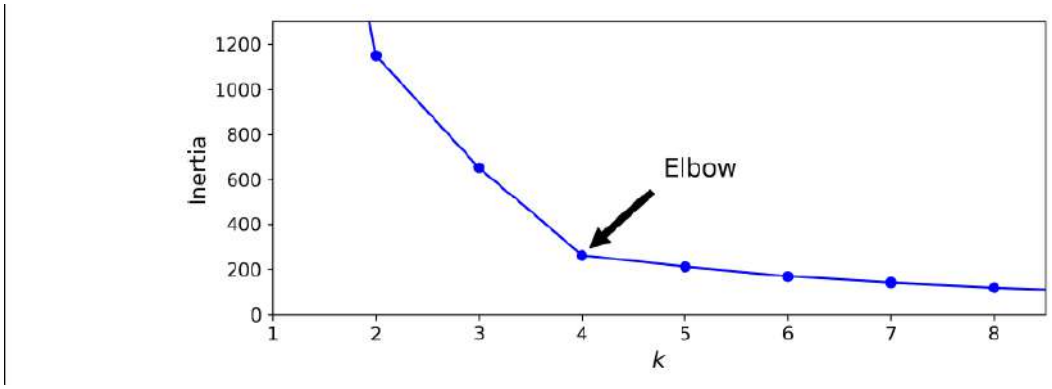
Algorithm:

1. Randomly assign K centres.
2. Calculate the distance of all the points from all the K centres and allocate the points to cluster based on the shortest distance. The model's *inertia* is the mean squared distance between each instance and its closest centroid. The goal is to have a model with the lowes intertia.
3. Once all the points are assigned to clusters, recompute the centroids.
4. Repeat the steps 2 and 3 until the locations of the centroids stop changing and the cluster allocation of the points becomes constant.

As we saw earlier, we need to provide the value of K beforehand. But the question is how to get a good value of K. An optimum value of K is obtained using the Elbow Method.

The Elbow-Method

This method is based on the relationship between the within-cluster sum of squared distances(WCSS Or Inertia) and the number of clusters. It is observed that first with an increase in the number of clusters WCSS decreases steeply and then after a certain number of clusters the drop in WCSS is not that prominent. The point after which the graph between WCSS and the number of clusters becomes comparatively smother is termed as the elbow and the number of cluster at that point are the optimum number of clusters as even after increasing the clusters after that point the variation is not decreasing by much i.e., we have accounted for almost all the dissimilarity in the data. An elbow-curve looks like:



Hands On Example

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

In [2]:
```
dataset=pd.read_csv('Mall_Customers.csv')
dataset.head()
```

Out[2]:

	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

- Here I only Consider Annual Income (k$) and Spending Score (1-100) you can use as many data you want

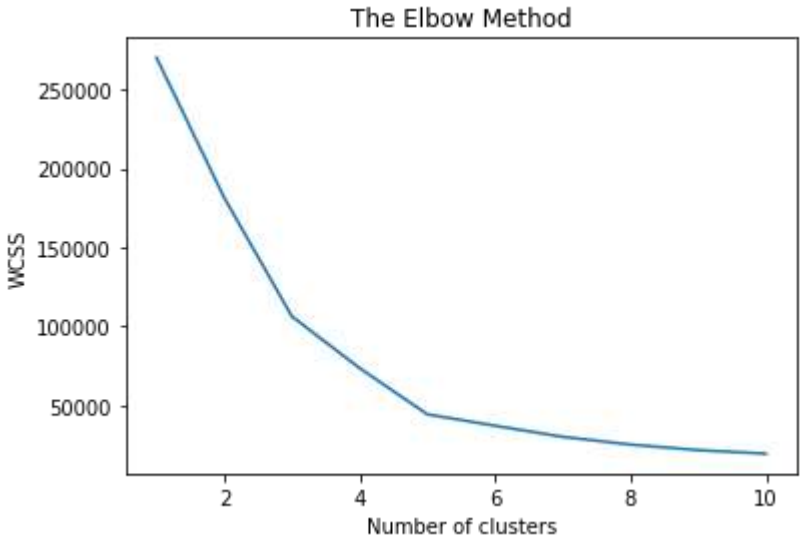
In [3]:
```
#dataset
X=dataset.iloc[:,3:]
X.head()
```

Out[3]:

	Annual Income (k$)	Spending Score (1-100)
0	15	39
1	15	81
2	16	6
3	16	77
4	17	40

Elbow Method to find Appropriate K-Value

In [4]:
```
#elbow method Find appropriate K-value
from sklearn.cluster import KMeans
wcss=[]
for i in range (1,11):
    kmeans=KMeans(n_clusters=i,init='k-means++',random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



- Here you see clearly at point 5 the curve is down so here i choose 5 is my K_value

Fitting K-Means to the dataset

In [5]:
```
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
```

In [6]:
```
y_kmeans
```

Out[6]:
```
array([3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0,
       3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 3, 1,
       3, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 4, 2, 1, 2, 4, 2, 4, 2,
       1, 2, 4, 2, 4, 2, 4, 2, 4, 2, 1, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2,
       4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2,
       4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2,
       4, 2])
```

In [7]:
```
# It predicts the cluster number to which the datapoint belongs to
test=kmeans.predict([[15, #Annual Income (k$)
                      39 #Spending Score (1-100)
                     ]])
print("it belongs to ",test[0],"cluster")
```

```
it belongs to  3 cluster
```

In [8]:
```
# Looking at the points which belong to Cluster0
X[y_kmeans==0].head() #it show in 0 number cluster which are include
```

Out[8]:

	Annual Income (k$)	Spending Score (1-100)
1	15	81
3	16	77
5	17	76
7	18	94
9	19	72

In [9]:
```
# Visualising the clusters
plt.scatter(X[y_kmeans == 0]['Annual Income (k$)'], X[y_kmeans == 0]['Spending Sc
plt.scatter(X[y_kmeans == 1]['Annual Income (k$)'], X[y_kmeans == 1]['Spending Sc
plt.scatter(X[y_kmeans == 2]['Annual Income (k$)'], X[y_kmeans == 2]['Spending Sc
plt.scatter(X[y_kmeans == 3]['Annual Income (k$)'], X[y_kmeans == 3]['Spending Sc
plt.scatter(X[y_kmeans == 4]['Annual Income (k$)'], X[y_kmeans == 4]['Spending Sc
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

K-Means Advantages :

1) If variables are huge, then K-Means most of the times computationally faster than hierarchical clustering, if we keep k smalls.

2) K-Means produce tighter clusters than hierarchical clustering, especially if the clusters are globular.

K-Means Disadvantages :

1) Difficult to predict K-Value.

2) With global cluster, it didn't work well.

3) Different initial partitions can result in different final clusters.

4) It does not work well with clusters (in the original data) of Different size and Different density

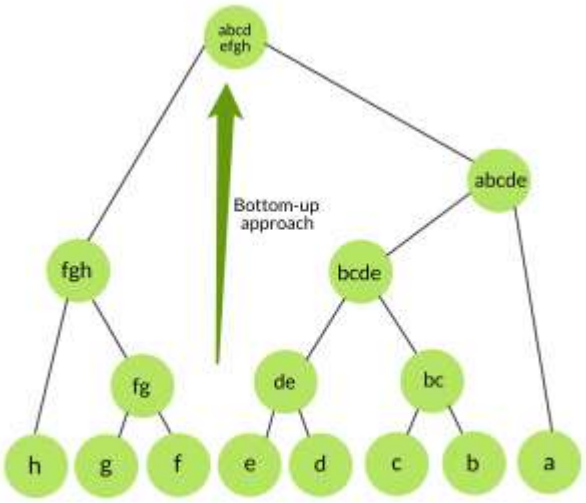
Hierarchical clustering

- One main disadvantage of K-Means is that it needs us to pre-enter the number of clusters (K). Hierarchical clustering is an alternative approach which does not need us to give the value of K beforehand and also, it creates a beautiful tree-based structure for visualization.
- There are two types of hierarchical clustering:

```
1. Agglomerative
2. Divisive
```

1. Agglomerative Clustering:

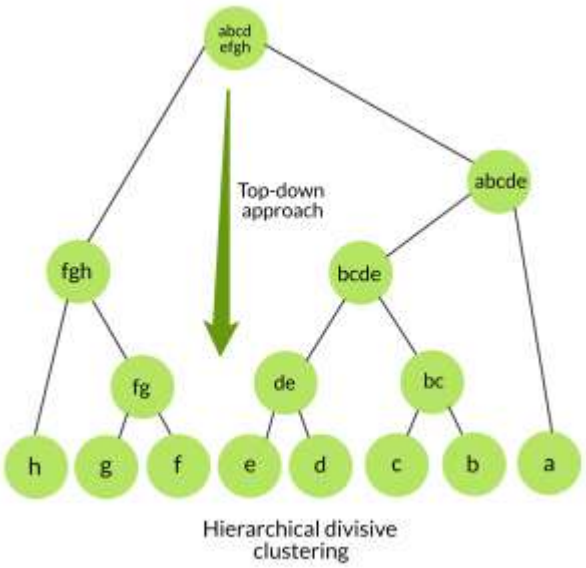
- Also known as bottom-up approach or hierarchical agglomerative clustering (HAC). A structure that is more informative than the unstructured set of clusters returned by flat clustering. This clustering algorithm does not require us to prespecify the number of clusters. Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively agglomerates pairs of clusters until all clusters have been merged into a single cluster that contains all data.



Hierarchical agglomerative clustering

2. Divisive clustering :

- Also known as top-down approach. This algorithm also does not require to prespecify the number of clusters. Top-down clustering requires a method for splitting a cluster that contains the whole data and proceeds by splitting clusters recursively until individual data have been splitted into singleton cluster.



Hierarchical divisive clustering

- Here, I am going to discuss the bottom-up (or Agglomerative) approach of cluster building. We start by defining any sort of similarity between the datapoints. Generally, we consider the Euclidean distance. The points which are closer to each are more similar than the points which re farther away. The Algorithms starts with considering all points as separate clusters and then grouping pints together to form clusters.

The Algorithm:

1. Begin with n observations and a measure (such as Euclidean distance) of all the n(n−1)/2 pairwise dissimilarities(or the Euclidean distances generally). Treat each observation as its own cluster. Initially, we have n clusters.
2. Compare all the distances and put the two closest points/clusters in the same cluster. The dissimilarity(or the Euclidean distances) between these two clusters indicates the height in the dendrogram at which the fusion line should be placed.
3. Compute the new pairwise inter-cluster dissimilarities(or the Euclidean distances) among the remaining clusters.
4. Repeat steps 2 and 3 till we have only one cluster left.

Hands On Example With Python

Execute the following script to import the desired libraries:

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
import numpy as np
```

Next, to import the dataset for this example, run the following code:

```
In [2]: customer_data=pd.read_csv('Mall_Customers.csv')
```

Let's explore our dataset a bit. To check the number of records and attributes, execute the following script:

```
In [3]: customer_data.shape
```

```
Out[3]: (200, 5)
```

The script above will return (200, 5) which means that the dataset contains 200 records and 5 attributes.

To eyeball the dataset, execute the head() function of the data frame. Take a look at the following script:

```
In [4]: customer_data.head()
```

Out[4]:

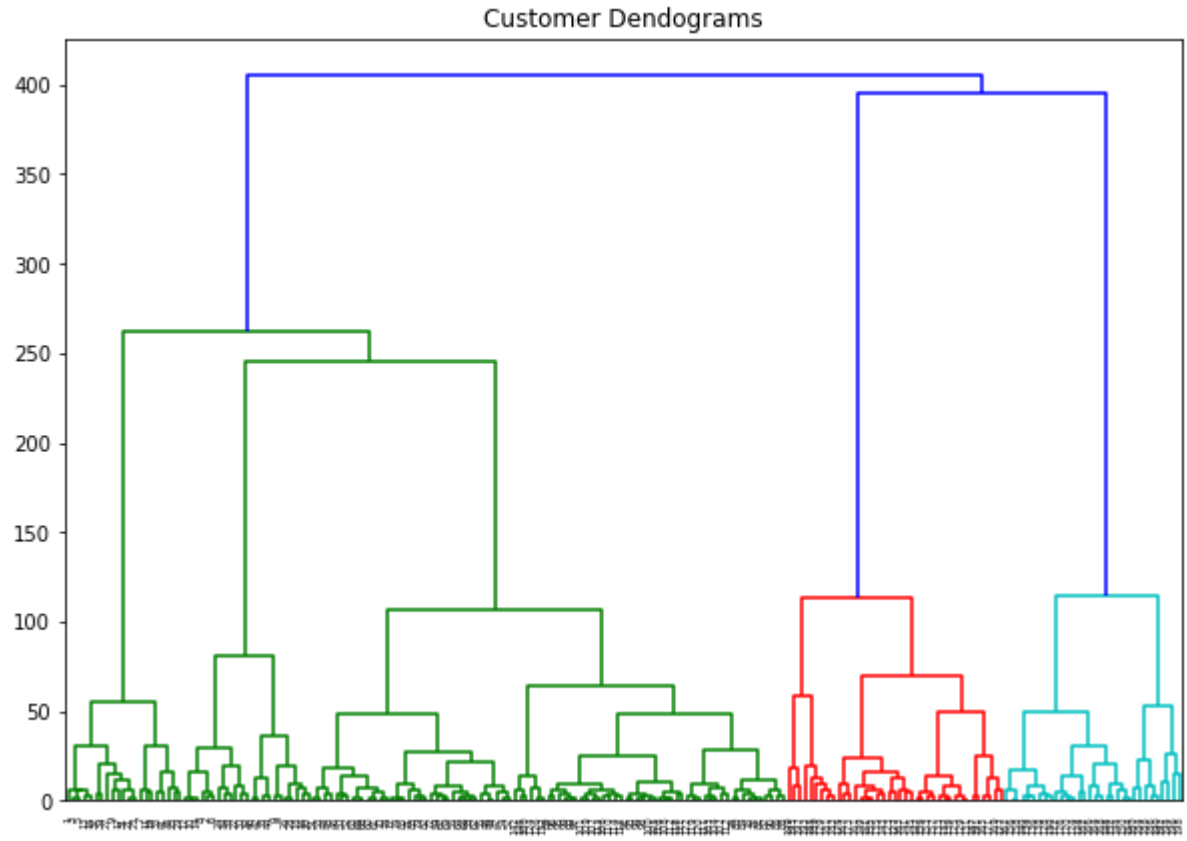
	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

Our dataset has five columns: CustomerID, Genre, Age, Annual Income, and Spending Score. To view the results in two-dimensional feature space, we will retain only two of these five columns. We can remove CustomerID column, Genre, and Age column. We will retain the Annual Income (in thousands of dollars) and Spending Score (1-100) columns. The Spending Score column signifies how often a person spends money in a mall on a scale of 1 to 100 with 100 being the highest spender. Execute the following script to filter the first three columns from our dataset:

```
In [5]: data = customer_data.iloc[:, 3:5].values
```

Next, we need to know the clusters that we want our data to be split to. We will again use the scipy library to create the dendrograms for our dataset. Execute the following script to do so:

In [6]:
```
import scipy.cluster.hierarchy as shc

plt.figure(figsize=(10, 7))
plt.title("Customer Dendograms")
dend = shc.dendrogram(shc.linkage(data, method='ward'))
```



In the script above we import the hierarchy class of the scipy.cluster library as shc. The hierarchy class has a dendrogram method which takes the value returned by the linkage method of the same class. The linkage method takes the dataset and the method to minimize distances as parameters. We use 'ward' as the method since it minimizes then variants of distances between the clusters.

Now we know the number of clusters for our dataset, the next step is to group the data points into these five clusters. To do so we will again use the AgglomerativeClustering class of the sklearn.cluster library. Take a look at the following script:

In [7]:
```
from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean',
                                  linkage='ward')
cluster.fit_predict(data)
```

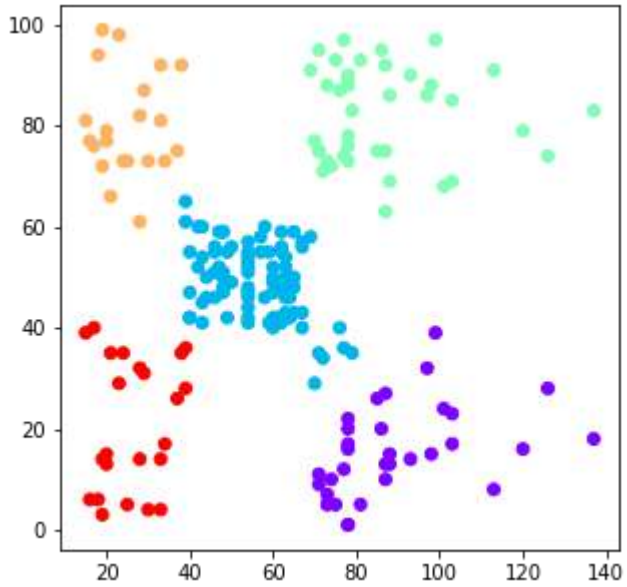
Out[7]:
```
array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
       4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,
       4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 0, 2, 0, 2,
       1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2], dtype=int64)
```

You can see the cluster labels from all of your data points. Since we had five clusters, we have five labels in the output i.e. 0 to 4.

As a final step, let's plot the clusters to see how actually our data has been clustered:

In [9]:
```
plt.figure(figsize=(5, 5))
plt.scatter(data[:,0], data[:,1], c=cluster.labels_, cmap='rainbow')
```

Out[9]:
```
<matplotlib.collections.PathCollection at 0x2655fd11948>
```

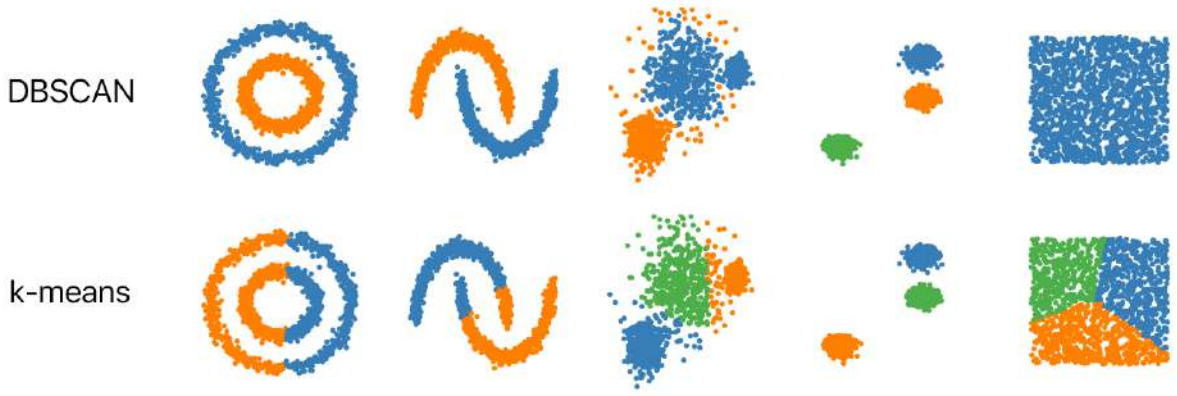


You can see the data points in the form of five clusters. The data points in the bottom right belong to the customers with high salaries but low spending. These are the customers that spend their money carefully. Similarly, the customers at top right (green data points), these are the customers with high salaries and high spending. These are the type of customers that companies target. The customers in the middle (blue data points) are the ones with average income and average salaries. The highest numbers of customers belong to this category. Companies can also target these customers given the fact that they are in huge numbers, etc.

Density-Based Spatial Clustering of applications with noise (DBSCAN)

Why do we need a Density-Based clustering algorithm like DBSCAN when we already have K-means clustering?

- K-Means clustering may cluster loosely related observations together. Every observation becomes a part of some cluster eventually, even if the observations are scattered far away in the vector space.
- Since clusters depend on the mean value of cluster elements, each data point plays a role in forming the clusters. A slight change in data points might affect the clustering outcome.
- This problem is greatly reduced in DBSCAN due to the way clusters are formed. This is usually not a big problem unless we come across some odd shape data.
- Another challenge with k-means is that you need to specify the number of clusters ("k") in order to use it. Much of the time, we won't know what a reasonable k value is a priori.
- nice about DBSCAN is that you don't have to specify the number of clusters to use it. All you need is a function to calculate the distance between values and some guidance for what amount of distance is considered "close".
- DBSCAN also produces more reasonable results than k-means across a variety of different distributions. Below figure illustrates the fact:



Some definitions first:

Epsilon: This is also called eps. This is the distance till which we look for the neighbouring points.

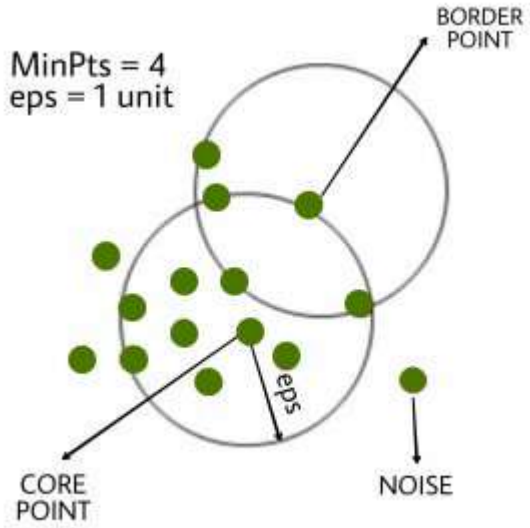
Min_points: The minimum number of points specified by the user.

Core Points: If the number of points inside the *eps radius* of a point is greater than or equal to the *min_points* then it's called a core point.

Border Points: If the number of points inside the *eps radius* of a point is less than the *min_points* and it lies within the *eps radius* region of a core point, it's called a border point.

Noise: A point which is neither a core nor a border point is a noise point.

Let's say if the eps=1 and min_points =4



Characteristics

- It identifies clusters of any shape in a data set, it means it can detect arbitrarily shaped clusters.
- It is based on intuitive notions of clusters and noise.
- It is very robust in detection of outliers in data set
- It requires only two points which are very insensitive to the order of occurrence of the points in data set

Algorithm Steps:

1. The algorithm starts with a random point in the dataset which has not been visited yet and its neighbouring points are identified based on the eps value.
2. If the point contains greater than or equal points than the min_pts, then the cluster formation starts and this point becomes a _core point_, else it's considered as noise. The thing to note here is that a point initially classified as noise can later become a border point if it's in the eps radius of a core point.
3. If the point is a core point, then all its neighbours become a part of the cluster. If the points in the neighbourhood turn out to be core points then their neighbours are also part of the cluster.
4. Repeat the steps above until all points are classified into different clusters or noises.

This algorithm works well if all the clusters are dense enough, and they are well separated by low-density regions.

Hands On Implementation And Compare All Three Algorithm

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

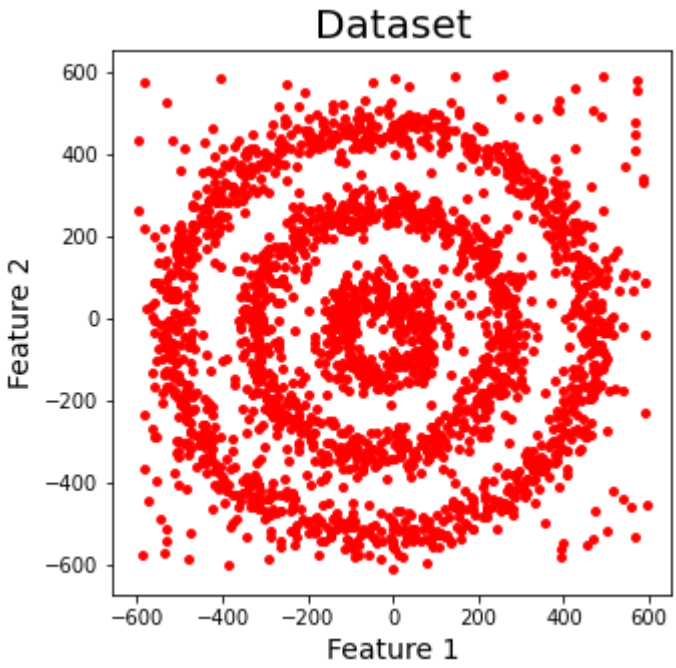
In [2]:
```
dataset = pd.read_csv('dbscan.csv')
df = dataset
df.head()
```

Out[2]:

	0	1
0	484.891555	-31.006357
1	489.391178	21.973916
2	462.886575	-27.599889
3	517.218479	5.588090
4	455.669049	1.982181

Let's plot these data points and see how they look in the feature space. Here, I use the scatter plot for plotting these data points. Use the following syntax:

In [3]:
```
plt.figure(figsize=(5,5))
plt.scatter(df['0'],df['1'],s=15,color='red')
plt.title('Dataset',fontsize=20)
plt.xlabel('Feature 1',fontsize=14)
plt.ylabel('Feature 2',fontsize=14)
plt.show()
```



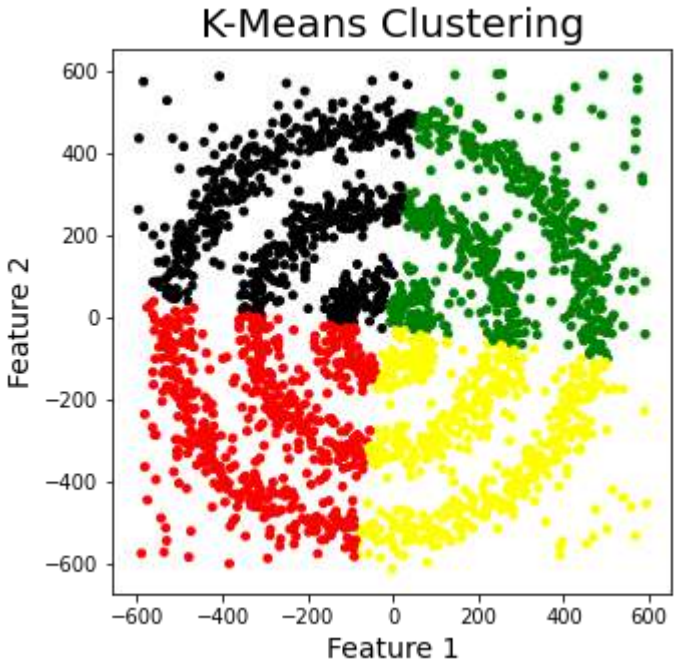
K-Means vs. Hierarchical vs. DBSCAN Clustering

K-Means

In [4]:
```
from sklearn.cluster import KMeans
k_means=KMeans(n_clusters=4,random_state=42)
k_means.fit(df[['0','1']])
```

Out[4]:
```
KMeans(n_clusters=4, random_state=42)
```

In [5]:
```
df['KMeans_labels']=k_means.labels_
colors=['red','green','yellow','black']
# Plotting resulting clusters
plt.figure(figsize=(5,5))
plt.scatter(df['0'],df['1'],c=df['KMeans_labels'],cmap=matplotlib.colors.ListedCo
plt.title('K-Means Clustering',fontsize=20)
plt.xlabel('Feature 1',fontsize=14)
plt.ylabel('Feature 2',fontsize=14)
plt.show()
```



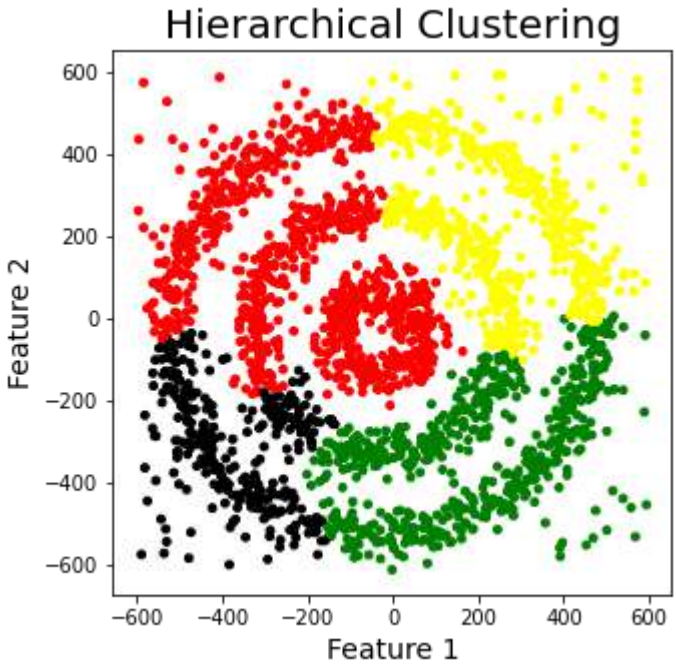
- Here, K-means failed to cluster the data points into four clusters. Also, it didn't work well with noise. Therefore, it is time to try another popular clustering algorithm, i.e., Hierarchical Clustering.

2. Hierarchical Clustering

In [6]:
```
from sklearn.cluster import AgglomerativeClustering
model = AgglomerativeClustering(n_clusters=4, affinity='euclidean')
model.fit(df[['0','1']])
```

Out[6]:
```
AgglomerativeClustering(n_clusters=4)
```

In [7]:
```
df['HR_labels']=model.labels_

# Plotting resulting clusters
plt.figure(figsize=(5,5))
plt.scatter(df['0'],df['1'],c=df['HR_labels'],cmap=matplotlib.colors.ListedColorm
plt.title('Hierarchical Clustering',fontsize=20)
plt.xlabel('Feature 1',fontsize=14)
plt.ylabel('Feature 2',fontsize=14)
plt.show()
```



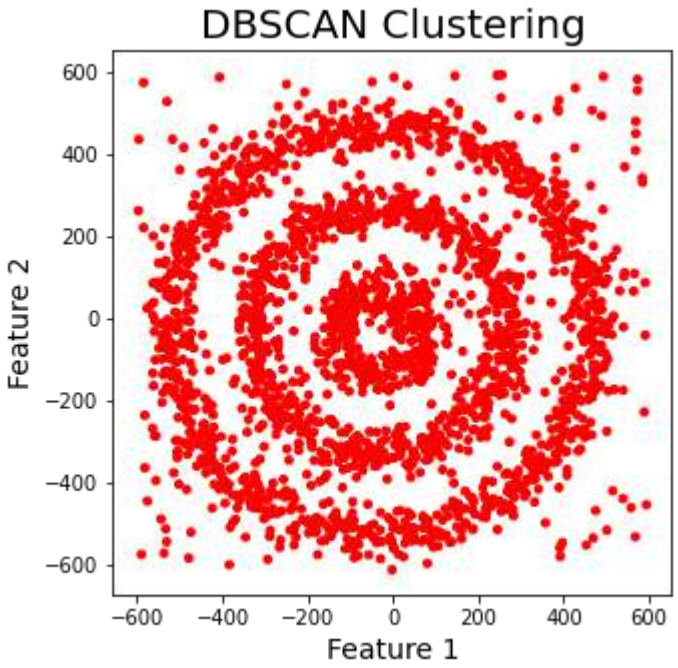
hierarchical clustering algorithm also failed to cluster the data points properly. Now got to DBSCAN clustering

3. DBSCAN Clustering

In [8]:
```
from sklearn.cluster import DBSCAN
dbscan=DBSCAN()
dbscan.fit(df[['0','1']])
```

Out[8]:
```
DBSCAN()
```

In [9]:
```
df['DBSCAN_labels']=dbscan.labels_

# Plotting resulting clusters
plt.figure(figsize=(5,5))
plt.scatter(df['0'],df['1'],c=df['DBSCAN_labels'],cmap=matplotlib.colors.ListedCo
plt.title('DBSCAN Clustering',fontsize=20)
plt.xlabel('Feature 1',fontsize=14)
plt.ylabel('Feature 2',fontsize=14)
plt.show()
```

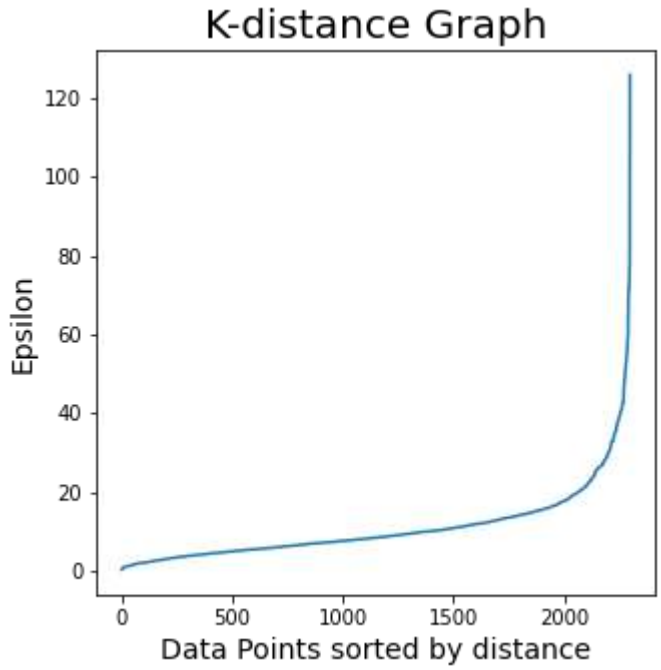


All the data points are now of red color which means they are treated as noise. It is because the value of epsilon is very small and we didn't optimize parameters. Therefore, we need to find the value of epsilon and minPoints and then train our model again.

For epsilon, I am using the K-distance graph. For plotting a K-distance Graph, we need the distance between a point and its nearest data point for all data points in the dataset. We obtain this using NearestNeighbors from sklearn.neighbors.

In [10]:
```
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(n_neighbors=2)
nbrs = neigh.fit(df[['0','1']])
distances, indices = nbrs.kneighbors(df[['0','1']])
```

In [11]:
```
# Plotting K-distance Graph
distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.figure(figsize=(5,5))
plt.plot(distances)
plt.title('K-distance Graph',fontsize=20)
plt.xlabel('Data Points sorted by distance',fontsize=14)
plt.ylabel('Epsilon',fontsize=14)
plt.show()
```

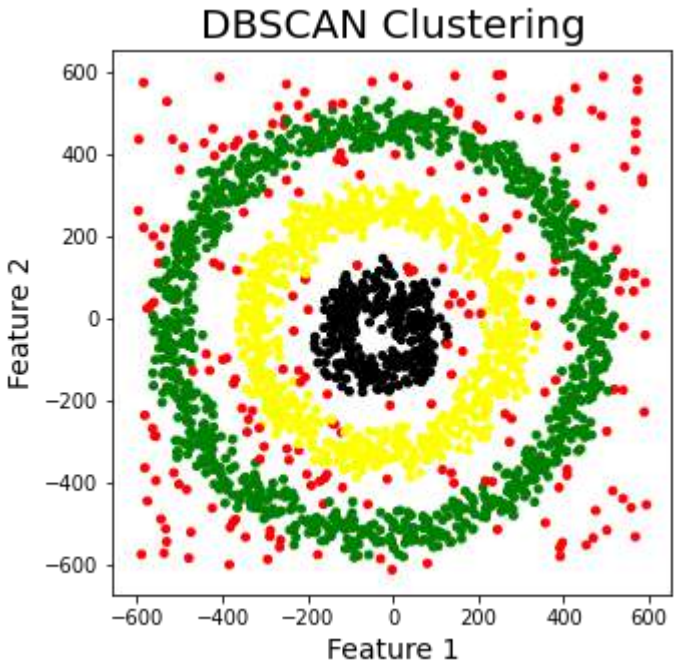


The optimum value of epsilon is at the point of maximum curvature in the K-Distance Graph, which is 30 in this case. Now, it's time to find the value of minPoints. The value of minPoints also depends on domain knowledge. This time I am taking minPoints as 6:

In [12]:
```
from sklearn.cluster import DBSCAN
dbscan_opt=DBSCAN(eps=30,min_samples=6)
dbscan_opt.fit(df[['0','1']])
```

Out[12]:
```
DBSCAN(eps=30, min_samples=6)
```

In [13]:
```
df['DBSCAN_opt_labels']=dbscan_opt.labels_

# Plotting the resulting clusters
plt.figure(figsize=(5,5))
plt.scatter(df['0'],df['1'],c=df['DBSCAN_opt_labels'],cmap=matplotlib.colors.List
plt.title('DBSCAN Clustering',fontsize=20)
plt.xlabel('Feature 1',fontsize=14)
plt.ylabel('Feature 2',fontsize=14)
plt.show()
```



DBSCAN amazingly clustered the data points into three clusters, and it also detected noise in the dataset represented by the red color.

One thing important to note here is that, though DBSCAN creates clusters based on varying densities, it struggles with clusters of similar densities. Also, as the dimension of data increases, it becomes difficult for DBSCAN to create clusters and it falls prey to the Curse of Dimensionality.

@@ Credit : https://www.analyticsvidhya.com/blog/2020/09/how-dbscan-clustering-works/ (https://www.analyticsvidhya.com/blog/2020/09/how-dbscan-clustering-works/)

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

Out[14]:

	0	1	KMeans_labels	HR_labels	DBSCAN_labels	DBSCAN_opt_labels
0	484.891555	-31.006357	1	1	-1	0
1	489.391178	21.973916	1	2	-1	0
2	462.886575	-27.599889	1	1	-1	0
3	517.218479	5.588090	1	1	-1	0
4	455.669049	1.982181	1	2	-1	0

Evaluation of Clustering

Cluster Validity

The validation of clusters created is a troublesome task. The problem here is: ”clusters are in the eyes of the beholder”

A good cluster will have:

- High inter-class similarity, and
- Low intraclass similarity

Aspects of cluster validation

- **External:** Compare your cluster to the ground truth.
- **Internal:** Evaluating the cluster without reference to external data.
- **Reliability:** The clusters are not formed by chance(randomly)- some statistical framework can be used.

External Measures:

1. Adjusted Rand index
2. Jaccard Score
3. Purity Score

1. Adjusted Rand index

N: Number of objects in the data P: $P_1, P_2, \dots, P_m$ the set of ground truth clusters C: $C_1, C_2, \dots C_n$ the set of clusters formed by the algorithm

The Incidence Matrix: N* N matrix

$P_i j = 1$ if the two points O_i and O_j belong to the same cluster in the ground truth else $P_i j = 0$

$C_i j = 1$ if the two points O_i and O_j belong to the same cluster in the ground truth else $C_i j = 0$

Now there can be the following scenarios:

1. $C_i j = P_i j = 1$ --> both the points belong to the same cluster for both our algorithm and ground truth(Agree)--- **SS**
2. $C_i j = P_i j = 0$ --> both the points don’t belong to the same cluster for both our algorithm and ground truth(Agree)--- **DD**
3. ${}_C i j = 1 but P_i j = 0$ --> The points belong in the same cluster for our algorithm but in different clusters for the ground truth (Disagree)---- **SD**
4. $C_i j = 0 but P_i j = 1$ --> The points don’t belong in the same cluster for our algorithm but in same clusters for the ground truth (Disagree)----**DS**

Rand Index= $\frac{Total Agree}{Total Disagree} = \frac{(SS+DD)}{(SS+DD+DS+SD)}$

Given the knowledge of the ground truth class assignments labels_true and our clustering algorithm assignments of the same samples labels_pred, the adjusted Rand index is a function that measures the similarity of the two assignments, ignoring permutations and with chance normalization:

In [1]:
```
from sklearn import metrics
y_true = [0, 0, 0, 1, 1, 1]
y_pred = [0, 0, 1, 1, 2, 2]
```

In [2]:
```
print("adjusted_rand_score : %0.3f" % metrics.adjusted_rand_score(y_true, y_pred
```

```
adjusted_rand_score : 0.242
```

The Adjusted Rand index is bounded between -1 and 1. Closer to 1 is good, while closer to -1 is bad.

2. Jaccard Score

Jaccard Coeeficient= $\frac{SS}{(SS+SD+DS)}$

In [3]:
```
print("jaccard_score: %0.3f" % metrics.jaccard_score(y_true, y_pred, average='mac
```

```
jaccard_score: 0.306
```

jaccard_score may be a poor metric if there are no positives for some samples or classes. Jaccard is undefined if there are no true or predicted labels, and our implementation will return a score of 0 with a warning.

A higher value of Rand Index and Jaccard's coefficient mean that the clusters generated by our algorithm mostly agree to the ground truth.

3. Purity Score:

Entropy of Cluster i, given by $e_i = -\sum p_{ij} log(p_{ij})$

For the entire clustering algorithm, the entropy can be given as: $e = \sum \frac{m_i}{n} e_i$

Purity is the total percentage of data points clustered correctly.

The purity of cluster i, given by $p_i = max(p_{ij})$

And for the entire cluster it is: $p(C) = \sum \frac{m_i}{n} p_i$

The Scikit-Learn Package hasen't yet implemented the Purity metrics. Hence, we'll write our custom code to implememt that.

In [4]:
```
import numpy as np
from sklearn import metrics

def purity_score(y_true, y_pred):
    # compute contingency matrix (also called confusion matrix)
    contingency_matrix = metrics.cluster.contingency_matrix(y_true, y_pred)
    # return purity
    return np.sum(np.amax(contingency_matrix, axis=0)) / np.sum(contingency_matri
```

In [5]:
```
print("purity_score is : ",purity_score(y_true, y_pred).round(2))
```

```
purity_score is :  83.33
```

A high value of purity score means that our clustering algorithm performs well against the ground truth.

So far we have discussed the External Measures. But as it is an unsupervised learning problem, most of the times there is no ground truth to compare to. Let's discuss the internal measures.

Internal Measure

1. silhouette_score
2. Davies-Bouldin Index (DB Index)
3. Dunn Index

1. silhouette_score

Note: BSS+WSS is always a constant.

The silhouette can be calculated as:

$$s(x) = \frac{b(x) - a(x)}{\max\{a(x), b(x)\}}$$

Where a(x) is the avarage distance of x from all the other points in the same cluster and b(x) is the avarage distance of x from all the other points in the other clusters.

And the Silhoeutte coefficient is given by:

$$SC = \frac{1}{N} \sum S(x)$$

In [6]:
```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Generating the sample data from make_blobs

X, Y = make_blobs()

cluster = KMeans(n_clusters = 5)
cluster_labels = cluster.fit_predict(X)

# The silhouette_score gives the
# average value for all the samples.
silhouette_avg = silhouette_score(X, cluster_labels)

print("The average silhouette_score is :", silhouette_avg)
```

```
The average silhouette_score is : 0.45794199236538047
```

2. Davies-Bouldin Index (DB Index) :

- The DB Index is calculated by the following formula:

$$DB = \frac{1}{n}\sum_{i=1}^{n} max_{j \neq i}\left(\frac{\sigma_i + \sigma_j}{d(c_i, c_j)}\right)$$

where:

n is the number of clusters

σi is the average distance of all points in cluster i from the cluster centroid ci.

- The score is defined as the average similarity measure of each cluster with its most similar cluster, where similarity is the ratio of within-cluster distances to between-cluster distances. Thus, clusters which are farther apart and less dispersed will result in a better score.

In [8]:
```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import davies_bouldin_score
import warnings
warnings.simplefilter('ignore')

# loading the dataset
X, Y = make_blobs()

# K-Means
cluster = KMeans(n_clusters = 4)
cluster_labels = cluster.fit_predict(X)


print("DB Index Score: ",davies_bouldin_score(X, cluster_labels))
```

```
DB Index Score:  0.5765098067760293
```

3. Dunn Index

- The aim of this Dunn index to identify sets of clusters that are compact, with a small variance between members of the cluster, and well separated, where the means of different clusters are sufficiently far apart, as compared to the within cluster variance.
- Higher the Dunn index value, better is the clustering.

$$\text{Dunn index}(U) = \min_{1 \le i \le c} \left\{ \min_{1 \le j \le c,\, j \ne i} \left\{ \frac{\delta(X_i, X_j)}{\max_{1 \le k \le c} [\Delta(X_k)]} \right\} \right\}$$

where:

$\delta(X_i, X_j)$ is the intercluster distance i.e. the distance between cluster X_i and X_j

$\Delta(X_k)$ is the intracluster distance of cluster X_k i.e. distance within the cluster X_k

- it gets better when clusters are well-spaced and dense. But the Dunn Index increases as performance improves.

```
import pandas as pd
from sklearn import datasets
from jqmcvi import base

# loading the dataset
X = datasets.load_iris()
df = pd.DataFrame(X.data)

# K-Means
from sklearn import cluster
k_means = cluster.KMeans(n_clusters=3)
```

```
k_means.fit(df) #K-means training
y_pred = k_means.predict(df)

# We store the K-means results in a dataframe
pred = pd.DataFrame(y_pred)
pred.columns = ['Type']

# we merge this dataframe with df
prediction = pd.concat([df, pred], axis = 1)

# We store the clusters
clus0 = prediction.loc[prediction.Species == 0]
clus1 = prediction.loc[prediction.Species == 1]
clus2 = prediction.loc[prediction.Species == 2]
cluster_list = [clus0.values, clus1.values, clus2.values]

print(base.dunn(cluster_list))
```