Python for Machine Learning and Data Science

Aqib Hameed

1 Why Machine Learning ??

Machine learning is a growing technology which enables computers to learn automatically from past data. Machine learning uses various algorithms for building mathematical models and making predictions using historical data or information. Currently, it is being used for various tasks such as image recognition, speech recognition, email filtering, Facebook auto-tagging, recommender system, and many more.

1.1 How does Machine Learning work

A Machine Learning system learns from historical data, builds the prediction models, and whenever it receives new data, predicts the output for it. The accuracy of predicted output depends upon the amount of data, as the huge amount of data helps to build a better model which predicts the output more accurately.
Suppose we have a complex problem, where we need to perform some predictions, so instead of writing a code for it, we just need to feed the data to generic algorithms, and with the help of these algorithms, machine builds the logic as per the data and predict the output.
The below block diagram explains the working of Machine Learning algorithm:

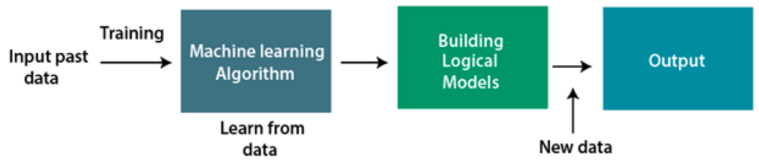

Figure 1: ML working.

2 Machine Learning Applications

Machine learning is growing very rapidly day by day. We are using machine learning in our daily life even without knowing it such as Google Maps, Google assistant, Alexa, etc. Below are some most trending real-world applications of Machine Learning.

2.1 Image Recognition

Image recognition is one of the most common applications of machine learning. It is used to identify objects, persons, places, digital images, etc. The popular use case of image recognition and face detection is, Automatic friend tagging suggestion

2.2 Traffic prediction

If we want to visit a new place, we take help of Google Maps, which shows us the correct path with the shortest route and predicts the traffic conditions.

2.3 Product recommendations

Machine learning is widely used by various e-commerce and entertainment companies such as Amazon, Netflix, etc., for product recommendation to the user.

2.4 Self-driving cars

Machine learning plays a significant role in self-driving cars. Tesla, the most popular car manufacturing company is working on self-driving car. It is using unsupervised learning method to train the car models to detect people and objects while driving.

2.5 Email Spam and Malware Filtering

Whenever we receive a new email, it is filtered automatically as important, normal, and spam. We always receive an important mail in our inbox with the important symbol and spam emails in our spam box, and the technology behind this is Machine learning.
Some machine learning algorithms such as Multi-Layer Perceptron, Decision tree, and Nave Bayes classifier are used for email spam filtering and malware detection.

2.6 Stock Market trading

Machine learning is widely used in stock market trading. In the stock market, there is always a risk of up and downs in shares, so for this machine learning's long short term memory neural network is used for the prediction of stock market trends.

2.7 Medical Diagnosis

In medical science, machine learning is used for diseases diagnoses. With this, medical technology is growing very fast and able to build 3D models that can predict the exact position of lesions in the brain.
It helps in finding brain tumors and other brain-related diseases easily.

3 Machine learning Life cycle

Machine learning has given the computer systems the abilities to automatically learn without being explicitly programmed. But how does a machine learning system work? So, it can be described using the life cycle of machine learning. The main purpose of the life cycle is to find a solution to the problem or project.
Machine learning life cycle involves seven major steps, which are given below.

3.1 Gathering Data

Data Gathering is the first step of the machine learning life cycle. The goal of this step is to identify and obtain all data-related problems.
In this step, we need to identify the different data sources, as data can be collected from various sources such as files, database, internet, or mobile devices. It is one of the most important steps of the life cycle. The quantity and quality of the collected data will determine the efficiency of the output. The more will be the data, the more accurate will be the prediction.
We get a coherent set of data, also called as a dataset. It will be used in further steps.

3.2 Data preparation

After collecting the data, we need to prepare it for further steps. Data preparation is a step where we put our data into a suitable place and prepare it to use in our machine learning training.

Data exploration is used to understand the nature of data that we have to work with. We need to understand the characteristics, format, and quality of data. A better understanding of data leads to an effective outcome. In this, we find Correlations, general trends, and outliers.
Now the next step is preprocessing of data for its analysis.

3.3 Data Wrangling

Data wrangling is the process of cleaning and converting raw data into a useable format. It is the process of cleaning the data, selecting the variable to use, and transforming the data in a proper format to make it more suitable for analysis in the next step.
In real-world applications, collected data may have various issues, including Missing Values,Duplicate data,Invalid data,Noise. We use various filtering techniques to clean the data.It is mandatory to detect and remove the issues because it can negatively affect the quality of the outcome.

3.4 Data Analysis

The aim of this step is to build a machine learning model to analyze the data using various analytical techniques and review the outcome. It starts with the determination of the type of the problems, where we select the machine learning techniques such as Classification, Regression, Cluster analysis, Association, etc. then build the model using prepared data, and evaluate the model.

3.5 Train Model

Now the next step is to train the model, in this step we train our model to improve its performance for better outcome of the problem.
We use datasets to train the model using various machine learning algorithms. Training a model is required so that it can understand the various patterns, rules, and, features.

3.6 Test Model

Once our machine learning model has been trained on a given dataset, then we test the model. In this step, we check for the accuracy of our model by providing a test dataset to it.
Testing the model determines the percentage accuracy of the model as per the requirement of project or problem.

3.7 Deployment

The last step of machine learning life cycle is deployment, where we deploy the model in the real-world system.
If the above-prepared model is producing an accurate result as per our requirement with acceptable speed, then we deploy the model in the real system.

4 Data Preprocessing

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.
It involves below steps:

4.1 Get the Dataset

To create a machine learning model, the first thing we required is a dataset as a machine learning model completely works on data. The collected data for a particular problem in a proper format is known as the dataset.
Dataset may be of different formats for different purposes. To use the dataset in our code, we usually put it into a CSV file. However, sometimes, we may also need to use an HTML or xlsx file. For data Preprocessing we will use dummy dataset in CSV file format.

4.2 Importing Libraries

In order to perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing which are the following

- **Numpy:**
  Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices.

- **Matplotlib:**
  It is a Python 2D plotting library, and with this library, we need to

import a sub-library pyplot. This library is used to plot any type of charts in Python for the code.

- **Pandas:**
  It is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library.

Below the piece of code how we import the above libraries.

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

4.3 Importing the Datasets

Before importing a dataset, we need to put the CSV file in the current directory.
Now to import the dataset, we will use read_csv() function of pandas library, which is used to read a csv file and performs various operations on it. Using this function, we can read a csv file locally as well as through an URL. We can use read_csv function as below

```
data_set = pd.read_csv('Dataset.csv')

Output:

   Country   Age   Salary Purchased
0   France  44.0  72000.0        No
1    Spain  27.0  48000.0       Yes
2  Germany  30.0  54000.0        No
3    Spain  38.0  61000.0        No
4  Germany  40.0      NaN       Yes
5   France  35.0  58000.0       Yes
6    Spain   NaN  52000.0        No
7   France  48.0  79000.0       Yes
8  Germany  50.0  83000.0        No
9   France  37.0  67000.0       Yes
```

The data_set is a name of the variable to store our dataset, and inside the function, we have passed the name of our dataset. Once we execute the above line of code, it will successfully import the dataset in our code.

4.3.1 Extracting dependent and independent variables

In machine learning, it is important to distinguish the matrix of features (independent variables) and dependent variables from dataset. In our dataset, there are three independent variables that are Country, Age, and Salary, and one is a dependent variable which is Purchased.

4.3.2 Extracting independent variable

To extract an independent variable, we will use iloc[ ] method of Pandas library. It is used to extract the required rows and columns from the dataset

```
x = data_set.iloc[:,:-1].values
```

In the above code, the first colon(:) is used to take all the rows, and the second colon(:) is for all the columns. Here we have used :-1, because we don't want to take the last column as it contains the dependent variable. So by doing this, we will get the matrix of features.

4.3.3 Extracting dependent variable

To extract dependent variables, again, we will use Pandas .iloc[] method.

```
y = data_set.iloc[:,3].values
```

Here we have taken all the rows with the last column only. It will give the array of dependent variables.

4.4 Handling Missing data

The next step of data preprocessing is to handle missing data in the datasets. If our dataset contains some missing data, then it may create a huge problem for our machine learning model. Hence it is necessary to handle missing values present in the dataset.
There are mainly two ways to handle missing data, which are

- **By deleting the particular row:**
  The first way is used to commonly deal with null values. In this way, we just delete the specific row or column which consists of null values. But this way is not so efficient and removing data may lead to loss of information which will not give the accurate output.

- **By calculating the mean:**
  In this way, we will calculate the mean of that column or row which

contains any missing value and will put it on the place of missing value. This strategy is useful for the features which have numeric data such as age, salary, year, etc.

To handle missing values, we will use Scikit-learn library in our code.we will use SimpleImputer class of sklearn.preprocessing library. Below is the code for it.

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
# Fitting imputer object to the independent variables x.
imputer_fit = imputer.fit(x[:, 1:3])
# Replacing missing data with the calculated mean value
x[:, 1:3] = imputer.transform(x[:, 1:3])

Output:

[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 63777.77777777778]
 ['France' 35.0 58000.0]
 ['Spain' 38.77777777777778 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

As we can see in the above output, the missing values have been replaced with the means of rest column values.

4.5 Encoding Categorical data

Categorical data is data which has some categories such as, in our dataset; there are two categorical variable, Country, and Purchased.
Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So it is necessary to encode these categorical variables into numbers.

- **For Country variable:**

Firstly, we will convert the country variables into categorical data. So to do this, we will use LabelEncoder() class from preprocessing library.

```
#Catgorical data
#for Country Variable
from sklearn.preprocessing import LabelEncoder
label_encoder_x= LabelEncoder()
x[:, 0]= label_encoder_x.fit_transform(x[:, 0])

Output:

[[0 44.0 72000.0]
 [2 27.0 48000.0]
 [1 30.0 54000.0]
 [2 38.0 61000.0]
 [1 40.0 63777.77777777778]
 [0 35.0 58000.0]
 [2 38.77777777777778 52000.0]
 [0 48.0 79000.0]
 [1 50.0 83000.0]
 [0 37.0 67000.0]]
```

In above code, we have imported LabelEncoder class of sklearn library. This class has successfully encoded the variables into digits.
But in our case, there are three country variables, and as we can see in the above output, these variables are encoded into 0, 1, and 2. By these values, the machine learning model may assume that there is some correlation between these variables which will produce the wrong output. So to remove this issue, we will use dummy encoding.
Dummy variables are those variables which have values 0 or 1. The 1 value gives the presence of that variable in a particular column, and rest variables become 0. With dummy encoding, we will have a number of columns equal to the number of categories.
In our dataset, we have 3 categories so it will produce three columns having 0 and 1 values. For Dummy Encoding, we will use OneHotEncoder class of preprocessing library. Here we have code of this.

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
columnTransformer = ColumnTransformer([('encoder', OneHotEncoder(),
  [0])], remainder='passthrough')
```

```
    x = columnTransformer.fit_transform(x)

Output:

[[1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [0.0 1.0 0.0 30.0 54000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 1.0 0.0 40.0 63777.77777777778]
 [1.0 0.0 0.0 35.0 58000.0]
 [0.0 0.0 1.0 38.77777777777778 52000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 1.0 0.0 50.0 83000.0]
 [1.0 0.0 0.0 37.0 67000.0]]
```

- **For Purchased Variable:**
  For the second categorical variable, we will only use labelencoder object of LableEncoder class. Here we are not using OneHotEncoder class because the purchased variable has only two categories yes or no, and which are automatically encoded into 0 and 1. Here we have code of this.

```
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)

Output:

[0 1 0 0 1 1 0 1 0 1]
```

4.6 Splitting the Dataset into the Training set and Test set

In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model. Suppose, if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models. If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So we always try to make a machine learning model which performs well

with the training set and also with the test dataset.
Training Set: A subset of dataset to train the machine learning model, and we already know the output.
Test set: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output. For splitting the dataset, we will use the below lines of code.

```
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)

Output:

x_train=  [[0.0 1.0 0.0 40.0 63777.77777777778]
 [1.0 0.0 0.0 37.0 67000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [0.0 0.0 1.0 38.77777777777778 52000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [1.0 0.0 0.0 44.0 72000.0]
 [1.0 0.0 0.0 35.0 58000.0]]
x_test=  [[0.0 1.0 0.0 30.0 54000.0]
 [0.0 1.0 0.0 50.0 83000.0]]
y_train=  [1 1 1 0 1 0 0 1]
y_test=  [0 0]
```

- In the above code, the first line is used for splitting arrays of the dataset into random train and test subsets.
- In the second line, we have used four variables for our output that are **x_train** features for the training data **x_test** features for testing data **y_train** Dependent variables for training data **y_test** Independent variable for testing data.
- In train_test_split() function, we have passed four parameters in which first two are for arrays of data, and test_size is for specifying the size of the test set. The test_size maybe .5, .3, or .2, which tells the dividing ratio of training and testing sets.
- The last parameter random_state is used to set a seed for a random generator so that you always get the same result, and the most used value for this is 42.

4.7 Feature Scaling

Feature scaling is the final step of data preprocessing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no any variable dominate the other variable.

```
Output:

    Country   Age    Salary  Purchased
0    France  44.0   72000.0         No
1     Spain  27.0   48000.0        Yes
2   Germany  30.0   54000.0         No
3     Spain  38.0   61000.0         No
4   Germany  40.0       NaN        Yes
5    France  35.0   58000.0        Yes
6     Spain   NaN   52000.0         No
7    France  48.0   79000.0        Yes
8   Germany  50.0   83000.0         No
9    France  37.0   67000.0        Yes
```

As we can see, the age and salary column values are not on the same scale. A machine learning model is based on Euclidean distance, and if we do not scale the variable, then it will cause some issue in our machine learning model.
we will use the standardization method for our dataset.
For feature scaling, we will import StandardScaler class of sklearn.preprocessing library as

```
from sklearn.preprocessing import StandardScaler
```

Now, we will create the object of StandardScaler class for independent variables or features. And then we will fit and transform the training dataset.

```
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
```

For test dataset, we will directly apply transform() function instead of fit_transform() because it is already done in training set.
Output:
By executing the above lines of code, we will get the scaled values for x_train and x_test as

```
x_train= [[-1.  2.64575131 -0.77459667  0.26306757  0.12381479]
          [ 1. -0.37796447 -0.77459667 -0.25350148  0.46175632]
          [-1. -0.37796447  1.29099445 -1.97539832 -1.53093341]
          [-1. -0.37796447  1.29099445  0.05261351 -1.11141978]
          [ 1. -0.37796447 -0.77459667  1.64058505  1.7202972 ]
          [-1. -0.37796447  1.29099445 -0.0813118  -0.16751412]
          [ 1. -0.37796447 -0.77459667  0.95182631  0.98614835]
          [ 1. -0.37796447 -0.77459667 -0.59788085 -0.48214934]]
x_test=  [[-1.  2.64575131 -0.77459667 -1.45882927 -0.90166297]
          [-1.  2.64575131 -0.77459667  1.98496442  2.13981082]]
```

As we can see in the above output, all the variables are scaled between values -1 to 1.
Here, we have not scaled the dependent variable because there are only two values 0 and 1. But if these variables will have more range of values, then we will also need to scale those variables.

5 Type of Machine learning

Machine Learning Algorithm can be broadly classified into two types.

1. **Supervised Learning Algorithms**
   Supervised learning is the types of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict the output.
   Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.

   Types of supervised Machine learning Algorithms
   Supervised learning can be further divided into two types of problems.

   - **Regression:**
     Regression algorithms are used if there is a relationship between the input variable and the output variable. It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc. Below are some popular Regression algorithms which come under supervised learning are Linear Regression Regression Trees, Non-Linear Regression, Bayesian Linear Regression, Polynomial Regression

- **Classification:**
  Classification algorithms are used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Female, True-false, etc. Spam Filtering, Random Forest, Decision Trees, Logistic Regression, Support vector Machines

Few of them models we will be discussed here

(a) **ML Polynomial Regression**

Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable(x) as nth degree polynomial. The Polynomial Regression equation is given below.

$y = b_0 + b_1x_1 + b_2x_1^2 + b_2x_1^3 + ...b_nx_1^n$
It is also called the special case of Multiple Linear Regression in ML. Because we add some polynomial terms to the Multiple Linear regression equation to convert it into Polynomial Regression

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

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

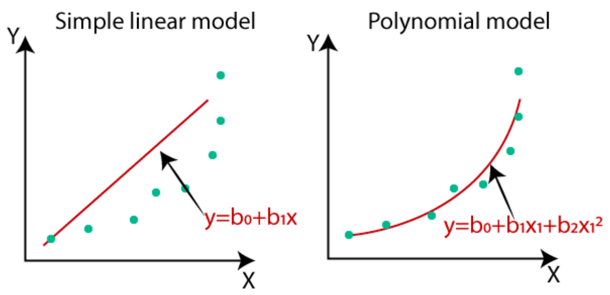

Figure 2: Linear vs Polynomial.

Implementation of Polynomial Regression using Python

Problem Description There is a Human Resource company, which is going to hire a new candidate. The candidate has told his previous salary 160K per annum, and the HR have to check whether he is telling the truth or bluff. So to identify this, they only have a dataset of his previous company in which the salaries of the top 10 positions are mentioned with their levels. By checking the dataset available, we have found that there is a non-linear relationship between the Position levels and the salaries. Our goal is to build a Bluffing detector regression model, so HR can hire an honest candidate. Below are the steps to build such a model.

Data Pre-processing

In the Polynomial Regression model, we will not use feature scaling, and also we will not split our dataset into training and test set. It has two reasons.

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

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

```
#importing datasets
data_set= pd.read_csv('Position_Salaries.csv')

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

Output:

            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 the above lines of code, we have imported the important Python libraries to import dataset and operate on it.
Next, we have imported the dataset 'Position_Salaries.csv', which contains three columns (Position, Levels, and Salary), but we will consider only two columns (Salary and Levels).
After that, we have extracted the dependent(Y) and independent variable(X) from the dataset. For x-variable, we have taken parameters as [:,1:2], because we want 1 index(levels), and included :2 to make it as a matrix.
As we can see in the above output, there are three columns present (Positions, Levels, and Salaries). But we are only considering two columns because Positions are equivalent to the levels or may be seen as the encoded form of Positions.

Building the Linear regression model
Now, we will build and fit the Linear regression model to the dataset. In building polynomial regression, we will take the Linear regression model as reference and compare both the results.

The code is given below

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

In the above code, we have created the Simple Linear model using lin_regs object of LinearRegression class and fitted it to the dataset variables (x and y).

Building the Polynomial regression model
Now we will build the Polynomial Regression model, but it will be a little different from the Simple Linear model. Because here we will use PolynomialFeatures class of preprocessing library. We are using this class to add some extra features to our dataset.

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

Output:

[[  1.   1.   1.]
 [  1.   2.   4.]
 [  1.   3.   9.]
 [  1.   4.  16.]
 [  1.   5.  25.]
 [  1.   6.  36.]
 [  1.   7.  49.]
 [  1.   8.  64.]
 [  1.   9.  81.]
 [  1.  10. 100.]]
```

In the above lines of code, we have used poly_regs.fit_transform(x), because first we are converting our feature matrix into polynomial feature matrix, and then fitting it to the Polynomial regression model. The parameter value(degree= 2) depends on our choice.

We can choose it according to our Polynomial features.

After executing the code, we will get another matrix x_poly, which can be seen under the ouptput.

Next, we have used another LinearRegression object, namely lin_reg_2, to fit our x_poly vector to the linear model.

Visualizing the result for Linear regression
Now we will visualize the result for Linear regression model as we did in Simple Linear Regression. Below is the code of it.

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

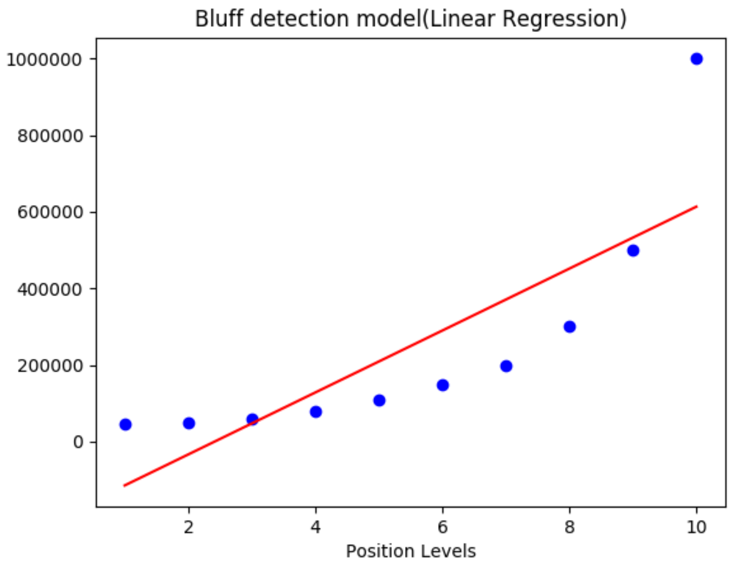

Figure 3:

In the above output image, we can clearly see that the regression line is so far from the datasets. Predictions are in a red straight line, and blue points are actual values. If we consider this output to predict the value of CEO, it will give a salary of approx. 600000$, which is far away from the real value.

So we need a curved model to fit the dataset other than a straight line.

Visualizing the result for Polynomial Regression
Here we will visualize the result of Polynomial regression model, code for which is little different from the above model.
Code for this is given below

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

In the above code, we have taken lin_reg_2.predict(poly_regs.fit_transform(x), instead of x_poly, because we want a Linear regressor object to predict the polynomial features matrix.
As we can see in the above output image, the predictions are close to the real values. The above plot will vary as we will change the degree.

Predicting the final result with the Linear Regression model
Now, we will predict the final output using the Linear regression model to see whether an employee is saying truth or bluff. So, for this, we will use the predict() method and will pass the value 6.5. Below is the code for it.

```
lin_pred = lin_regs.predict([[6.5]])
print(lin_pred)

Output:
[330378.78787879]
```

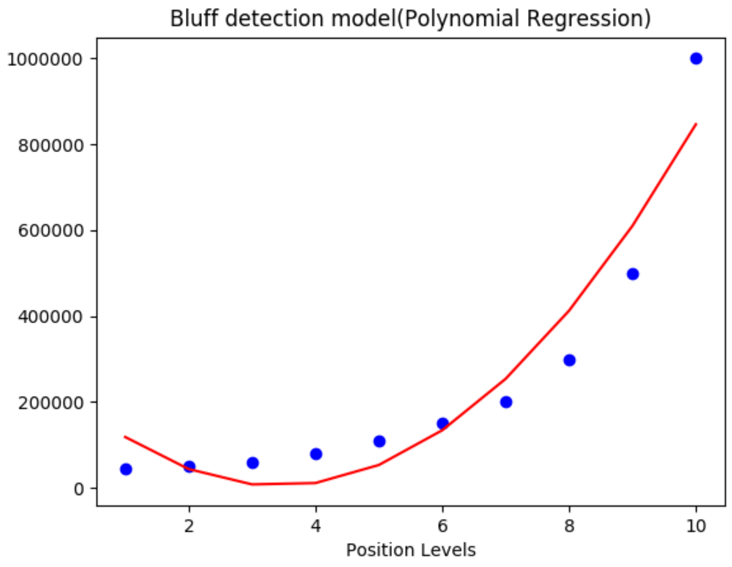

Figure 4:

Predicting the final result with the Polynomial Regression model

Now, we will predict the final output using the Polynomial Regression model to compare with Linear model. Below is the code for it.

```
poly_pred = lin_reg_2.predict(poly_regs.fit_transform([[6.5]]))
print(poly_pred)

Output:
[189498.10606061]
```

As we can see, the predicted output for the Polynomial Regression is [189498.10606061], which is much closer to real value hence, we can say that future employee is saying true.

(b) **Logistic Regression**

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.
In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).As you can see in the below image

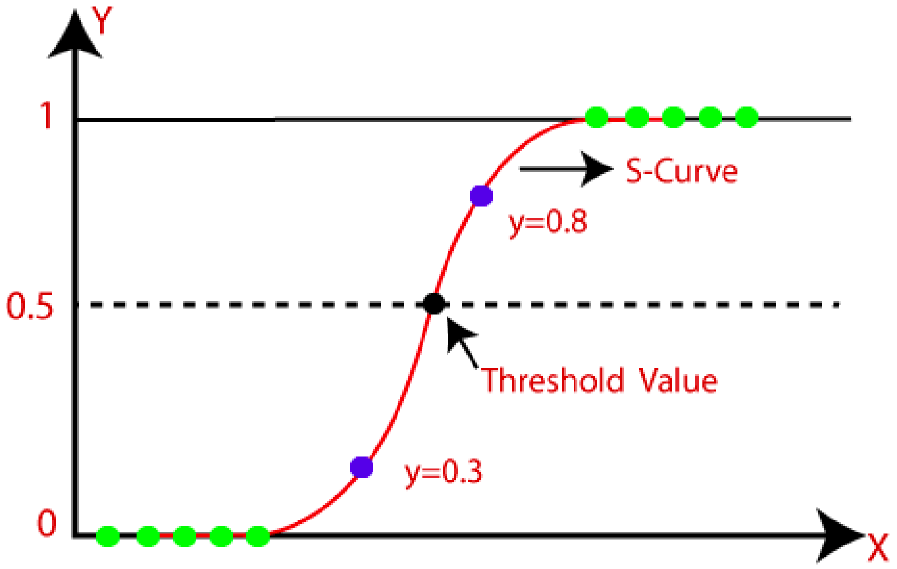

Figure 5:

Logistic Function (Sigmoid Function)
The sigmoid function is a mathematical function used to map the predicted values to probabilities.It maps any real value into another value within a range of 0 and 1.
The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.

In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Type of Logistic Regression
On the basis of the categories, Logistic Regression can be classified into three types.

- **Binomial:**
  In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:**
  In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep".
- **Ordinal:**
  In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

  Python Implementation of Logistic Regression (Binomial)
  There is a dataset given which contains the information of various users obtained from the social networking sites. There is a car making company that has recently launched a new SUV car. So the company wanted to check how many users from the dataset, wants to purchase the car.
  For this problem, we will build a Machine Learning model using the Logistic regression algorithm.In this problem, we will predict the purchased variable (Dependent Variable) by using age and salary (Independent variables).

  Data Pre-processing
  Now, we will extract the dependent and independent variables from the given dataset. Below is the code for it.

```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
```

```
y= data_set.iloc[:, 4].values
```

In the above code, we have taken [2, 3] for x because our independent variables are age and salary, which are at index 2, 3. And we have taken 4 for y variable because our dependent variable purachased which is at index 4.
Now we will split the dataset into a training set and test set. Below is the code of it.

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

In logistic regression, we will do feature scaling because we want accurate result of predictions. Here we will only scale the independent variable because dependent variable have only 0 and 1 values. Below is the code for it.

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

Fitting Logistic Regression to the Training set
We have well prepared our dataset, and now we will train the dataset using the training set. For providing training or fitting the model to the training set, we will import the LogisticRegression class of the sklearn library.
After importing the class, we will create a classifier object and use it to fit the model to the logistic regression. Below is the code for it.

```
#Fitting Logistic Regression to the training set
from sklearn.linear_model import LogisticRegression
classifier= LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
```

Predicting the Test Result
Our model is well trained on the training set, so we will now predict the result by using test set data. Below is the code of it.

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

In the above code, we have created a y_pred vector to predict the test set result.

Test Accuracy of the result
Now we will create the confusion matrix here to check the accuracy of the classification. To create it, we need to import the confusion_matrix function of the sklearn library. After importing the function, we will call it using a new variable cm. The function takes two parameters, mainly y_true(the actual values) and y_pred (the targeted value return by the classifier). Below is the code of it.

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)

Output:
confusion matrix [[65  3]
                  [ 8 24]]
```

By executing the above code, a new confusion matrix will be created.We can find the accuracy of the predicted result by interpreting the confusion matrix. By above output, we can interpret that 65+24= 89 (Correct Output) and 8+3= 11(Incorrect Output).

Visualizing the training set result
Finally, we will visualize the training set result. To visualize the result, we will use ListedColormap class of matplotlib library. Below is the code of it.

```
#Visualizing the training set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1,
 stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max()
 + 1, step = 0.01))
```

```
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
 x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Logistic Regression (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

In the above code, we have imported the ListedColormap class of Matplotlib library to create the colormap for visualizing the result. We have created two new variables x_set and y_set to replace x_train and y_train. After that, we have used the nm.meshgrid command to create a rectangular grid, which has a range of -1(minimum) to 1 (maximum). The pixel points we have taken are of 0.01 resolution.

To create a filled contour, we have used mtp.contourf command, it will create regions of provided colors (purple and green). In this function, we have passed the classifier.predict to show the predicted data points predicted by the classifier.

By executing the above code, we will get the below output:
In the below graph, we can see that there are some Green points within the green region and Purple points within the purple region.All these data points are the observation points from the training set, which shows the result for purchased variables.
This graph is made by using two independent variables i.e., Age on the x-axis and Estimated salary on the y-axis.
The purple point observations are for which purchased (dependent variable) is probably 0, i.e., users who did not purchase the SUV car. The green point observations are for which purchased (dependent variable) is probably 1 means user who purchased the SUV car.
We can also estimate from the graph that the users who are

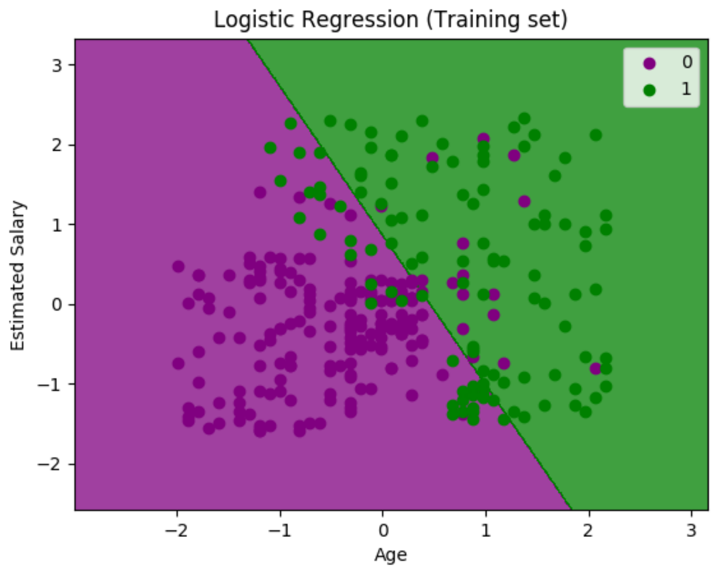

Figure 6:

younger with low salary, did not purchase the car, whereas older users with high estimated salary purchased the car.
But there are some purple points in the green region (Buying the car) and some green points in the purple region(Not buying the car). So we can say that younger users with a high estimated salary purchased the car, whereas an older user with a low estimated salary did not purchase the car.
We have successfully visualized the training set result for the logistic regression, and our goal for this classification is to divide the users who purchased the SUV car and who did not purchase the car. So from the output graph, we can clearly see the two regions (Purple and Green) with the observation points. The Purple region is for those users who didn't buy the car, and Green Region is for those users who purchased the car.

Visualizing the test set result

Our model is well trained using the training dataset. Now, we will visualize the result for new observations (Test set). The code for the test set will remain same as above except that here we will use x_test and y_test instead of x_train and y_train. Below is the code of it.

```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set , y_set = x_test , y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1,
 stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max()
 + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
 x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Logistic Regression (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output The below graph in Figure 7, shows the test set result. As we can see, the graph is divided into two regions (Purple and Green). And Green observations are in the green region, and Purple observations are in the purple region. So we can say it is a good prediction and model. Some of the green and purple data points are in different regions, which can be ignored as we have already calculated this error using the confusion matrix (11 Incorrect output).
Hence our model is pretty good and ready to make new predictions for this classification problem.

(c) **Support Vector Machine Algorithm**
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.
The goal of the SVM algorithm is to create the best line or deci-

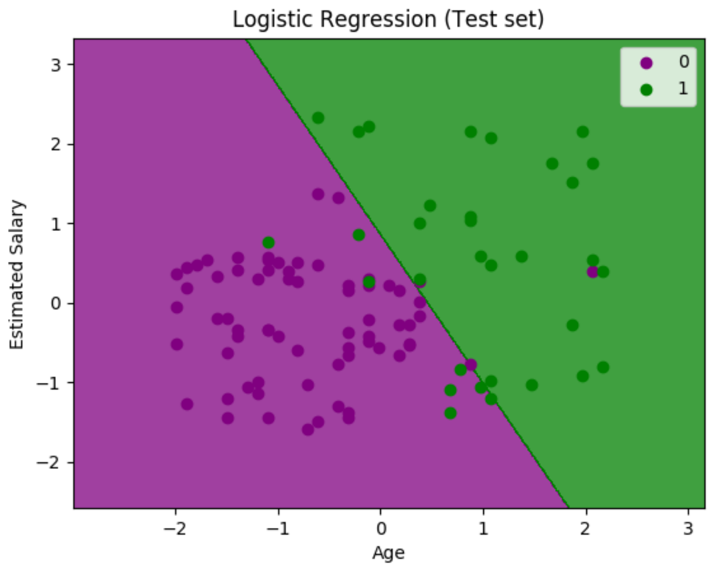

Figure 7:

sion boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.
SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.
Example: Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat.
SVM algorithm can be used for Face detection, image classification, text categorization, etc.

Types of SVM

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

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

How does SVM works?
Linear SVM:
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image Figure 8.

So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image Figure 9

Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a hyperplane. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as margin. And the goal of SVM is to maximize this margin. The hyperplane with maximum margin is called the optimal hyperplane.As you can seen in the below image Figure 10.

Non-Linear SVM:
If data is linearly arranged, then we can separate it by using a

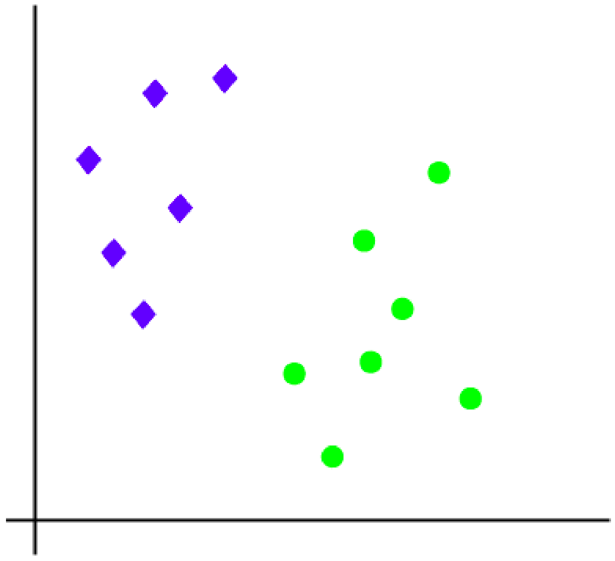

Figure 8:

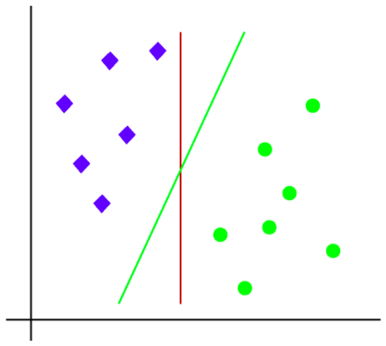

Figure 9:

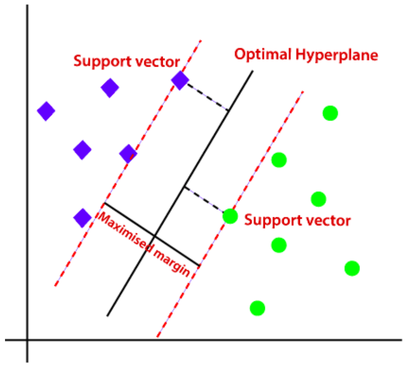

Figure 10:

straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image.

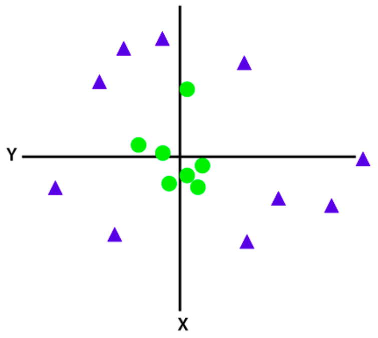

Figure 11:

Python Implementation of Support Vector Machine
Now we will implement the SVM algorithm using python.

Data Pre-processing step
Below the piece of code we will used for data pre-processing.

```
#Data Pre-processing Step
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

```
#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y,
 test_size= 0.25, random_state=0)
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

Fitting the SVM classifier to the training set:
Now the training set will be fitted to the SVM classifier. To create the SVM classifier, we will import SVC class from Sklearn.svm library. Below is the code of it.

```
from sklearn.svm import SVC # "Support vector classifier"
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)
```

In the above code, we have used kernel='linear', as here we are creating SVM for linearly separable data. However, we can change it for non-linear data. And then we fitted the classifier to the training dataset(x_train, y_train).

Predicting the test set result
Now, we will predict the output for test set. For this, we will create a new vector y_pred. Below is the code of it.

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

After getting the y_pred vector, we can compare the result of y_pred and y_test to check the difference between the actual value and predicted value.

Creating the confusion matrix
Now we will see the performance of the SVM classifier that how many incorrect predictions are there as compared to the Logistic regression classifier. To create the confusion matrix, we need to import the confusion_matrix function of the sklearn library. After importing the function, we will call it using a new variable cm. The function takes two parameters, mainly y_true(the actual values) and y_pred (the targeted value return by the classifier). Below is the code of it.

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)

Output:
confusion matrix = [[66  2]
                    [ 8 24]]
```

As we can see in the above output image, there are 66+24= 90 correct predictions and 8+2= 10 correct predictions. Therefore we can say that our SVM model improved as compared to the Logistic regression model.

Visualizing the training set result
Now we will visualize the training set result, below is the code of it.

```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1,
 stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max()
 + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
 x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

By executing the above code, we will get the output as

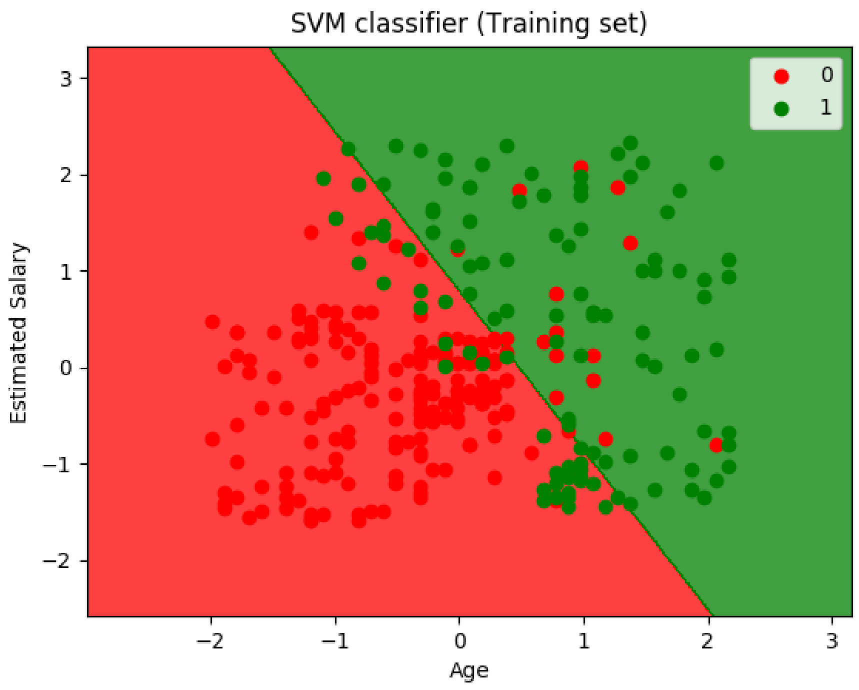

Figure 12:

Visualizing the test set result

```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1,
 stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max()
 + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
 x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

By executing the above code, we will get the output as

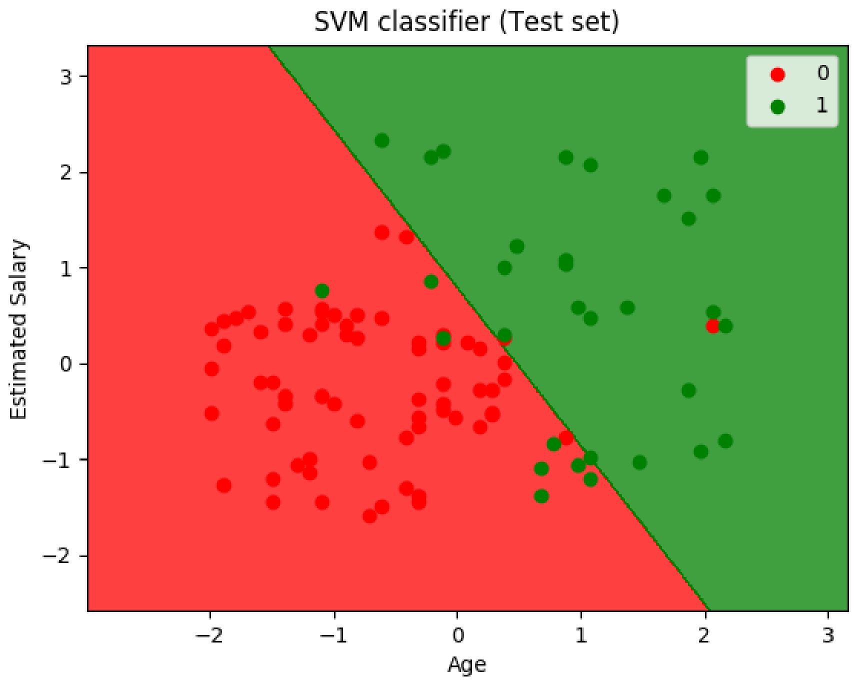

Figure 13:

As we can see in the above output image, the SVM classifier has divided the users into two regions (Purchased or Not purchased).

Users who purchased the SUV are in the red region with the red scatter points. And users who did not purchase the SUV are in the green region with green scatter points. The hyperplane has divided the two classes into Purchased and not purchased variable.

(d) **K-Nearest Neighbor(KNN) Algorithm**

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

K-NN algorithm assumes the similarity between the new case/-data and available cases and put the new case into the category that is most similar to the available categories.

K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

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

Python implementation of the KNN algorithm

To do the Python implementation of the K-NN algorithm, we will use the same problem and dataset which we have used in Logistic Regression. But here we will improve the performance of the model. Below is the problem description.

Problem: There is a Car manufacturer company that has manufactured a new SUV car. The company wants to give the ads to the users who are interested in buying that SUV. So for this problem, we have a dataset that contains multiple user's information through the social network. The dataset contains lots of information but the Estimated Salary and Age we will consider for the independent variable and the Purchased variable is for the dependent variable.

Data Pre-Processing Below is the code of the data processing.

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

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

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

By executing the above code, our dataset is imported to our program and well pre-processed.

Fitting K-NN classifier to the Training data Now we will fit the K-NN classifier to the training data. To do this we will import the KNeighborsClassifier class of Sklearn Neighbors library. After importing the class, we will create the Classifier object of the class. The Parameter of this class will be

- n_neighbors To define the required neighbors of the algorithm. Usually, it takes 5
- metric='minkowski': This is the default parameter and it decides the distance between the points.

- p=2 It is equivalent to the standard Euclidean metric.

And then we will fit the classifier to the training data. Below is the code of it

```
#Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski'
 p=2 )
classifier.fit(x_train, y_train)
```

Predicting the Test Result
To predict the test set result, we will create a y_pred vector as we did in Logistic Regression. Below is the code of it.

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

Creating the Confusion Matrix
Now we will create the Confusion Matrix for our K-NN model to see the accuracy of the classifier. Below is the code of it.

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)

Output:

confusion matrix = [[64  4]
                    [ 3 29]]
```

In above code, we have imported the confusion_matrix function and called it using the variable cm.
By executing the above code, we will get the matrix as shown above under the output.
In the above image, we can see there are 64+29= 93 correct predictions and 3+4= 7 incorrect predictions, whereas, in Logistic Regression, there were 11 incorrect predictions. So we can say that the performance of the model is improved by using the K-NN algorithm.

Visualizing the Training set result
Now, we will visualize the training set result for K-NN model.

The code will remain same as we did in Logistic Regression, except the name of the graph. Below is the code of it.

```
#Visulaizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1,
 stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max()
 + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
 x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:
By executing the above code, we will get the below graph The output graph is different from the graph which we have occurred in Logistic Regression. It can be understood in the below points.

- As we can see the graph is showing the red point and green points. The green points are for Purchased(1) and Red Points for not Purchased(0) variable.
- The graph is showing an irregular boundary instead of showing any straight line or any curve because it is a K-NN algorithm, i.e., finding the nearest neighbor.
- The graph has classified users in the correct categories as most of the users who didn't buy the SUV are in the red region and users who bought the SUV are in the green region.
- The graph is showing good result but still, there are some green points in the red region and red points in the green region. But this is no big issue as by doing this model is

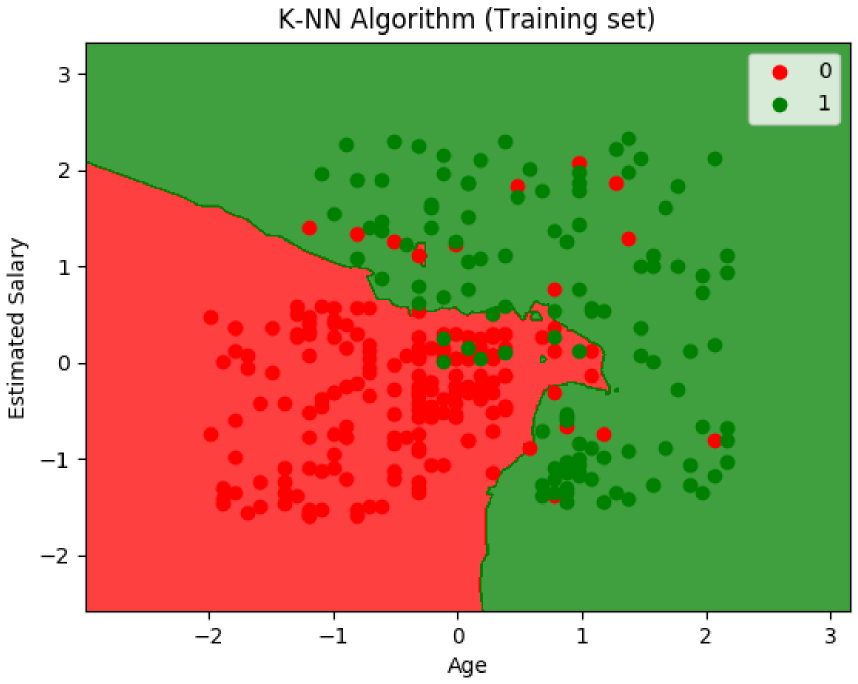
K-NN Algorithm (Training set)
0
1
Estimated Salary
Age
3
2
1
0
−1
−2
−2
−1
0
1
2
3

Figure 14:

prevented from overfitting issues.

Visualizing the Test set result

After the training of the model, we will now test the result by putting a new dataset, i.e., Test dataset. Code remains the same except some minor changes: such as x_train and y_train will be replaced by x_test and y_test.
Below is the code of it.

```
#Visualizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1,
 stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max()
 + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
 x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

.

Output:

The graph in figure 15 is showing the output for the test data set. As we can see in the graph, the predicted output is well good as most of the red points are in the red region and most of the green points are in the green region.

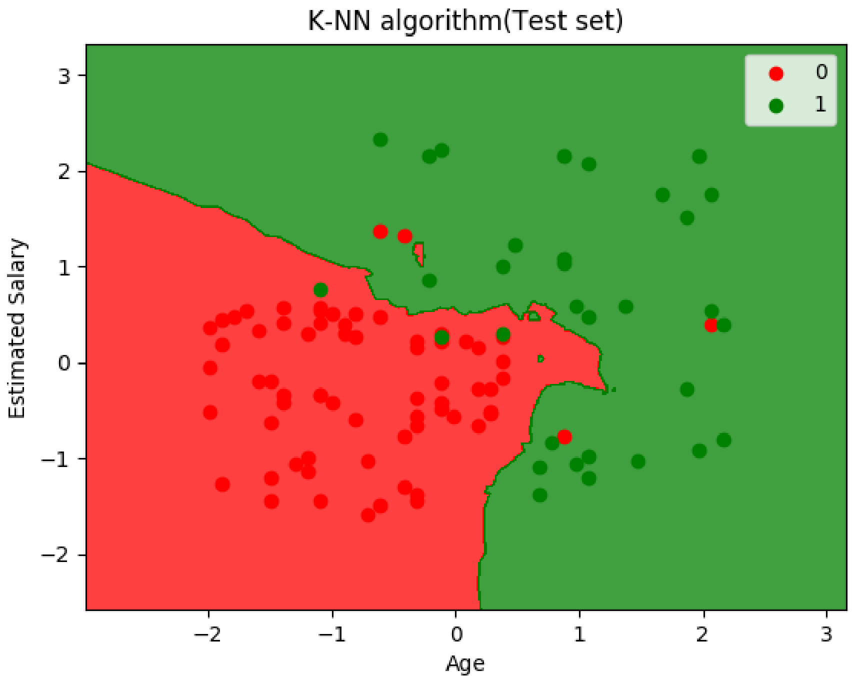
K-NN algorithm(Test set)
0
1
Estimated Salary
Age
3
2
1
0
−1
−2
−2
−1
0
1
2
3

Figure 15:

However, there are few green points in the red region and a few red points in the green region. So these are the incorrect observations that we have observed in the confusion matrix(7 Incorrect output).

2. **Unsupervised Learning Algorithms**
   As the name suggests, unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data.
   Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format.
   For example, Suppose the unsupervised learning algorithm is given an input dataset containing images of different types of cats and dogs. The algorithm is never trained upon the given dataset, which means it does not have any idea about the features of the dataset. The task of the unsupervised learning algorithm is to identify the image features on their own. Unsupervised learning algorithm will perform this task by clustering the image dataset into the groups according to similarities between images.
   Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.

   Type of Unsupervised Machine learning Algorithms

   - **Clustering:**
     Clustering is a method of grouping the objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities.

   Here are the list of some popular unsupervised learning algorithms.K-means clustering, Hierarchal clustering, Neural Networks Principle

Component Analysis, Apriori algorithm, Singular value decomposition.
We will discuss few of them as below.

a) Hierarchical Clustering
Hierarchical clustering is an unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster and also known as hierarchical cluster analysis or HCA.
In this algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree-shaped structure is known as the dendrogram.
The hierarchical clustering technique has two approaches

- Agglomerative is a bottom-up approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.
- Divisive algorithm is the reverse of the agglomerative algorithm as it is a top-down approach.

Agglomerative Hierarchical clustering
The agglomerative hierarchical clustering algorithm is a popular example of HCA. To group the datasets into clusters, it follows the bottom-up approach. It means, this algorithm considers each dataset as a single cluster at the beginning, and then start combining the closest pair of clusters together. It does this until all the clusters are merged into a single cluster that contains all the datasets.
This hierarchy of clusters is represented in the form of the dendrogram.

How the Agglomerative Hierarchical clustering Work?
The working of the AHC algorithm can be explained using the following steps

- Create each data point as a single cluster. Let's say there are N data points, so the number of clusters will also be N.As you can see below in the image Figure 16.
- Take two closest data points or clusters and merge them to form one cluster. So, there will now be N-1 clusters.As you can see below in the image Figure 17.

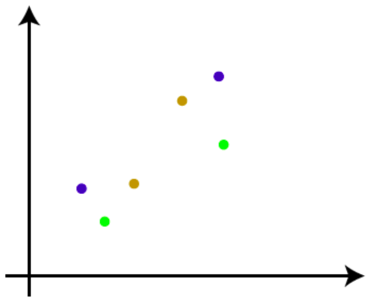

Figure 16:

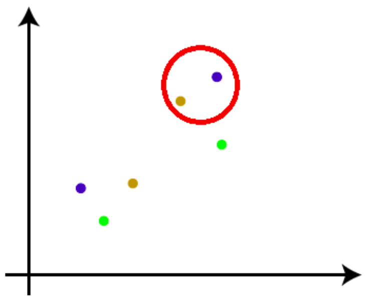

Figure 17:

- Again, take the two closest clusters and merge them together to form one cluster. There will be N-2 clusters.As you can see below in the image Figure 18.

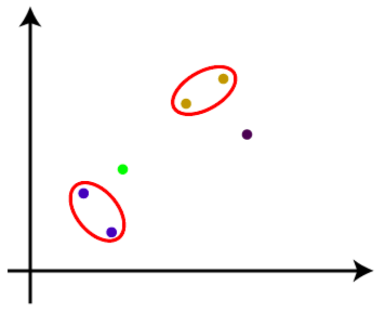

Figure 18:

- Repeat the above step until only one cluster left. So, we will get the following clusters. Consider the below Figures, Fig 19 and Fig 20.

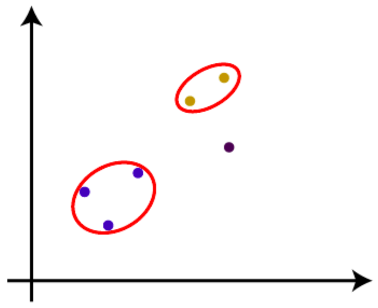

Figure 19:

- Once all the clusters are combined into one big cluster, develop the dendrogram to divide the clusters as per the problem.

Measure for the distance between two clusters
As we have seen, the closest distance between the two clusters is crucial for the hierarchical clustering. There are various ways to calculate the distance between two clusters, and these ways decide the rule for clustering. These measures are called Linkage methods. Some of the popular linkage methods are given below.

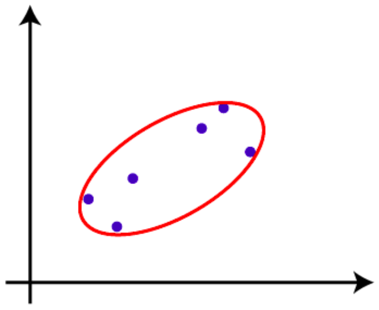

Figure 20:

- **Single Linkage**
  It is the Shortest Distance between the closest points of the clusters. Consider the below image.

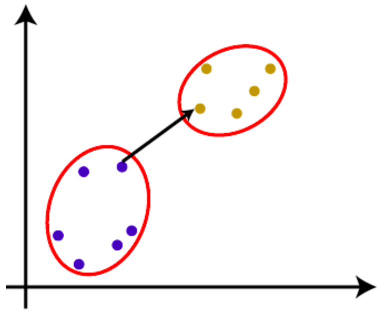

Figure 21:

- **Complete Linkage**
  It is the farthest distance between the two points of two different clusters. It is one of the popular linkage methods as it forms tighter clusters than single-linkage.As we can seen in the Figure 22.
- **Average Linkage**
  It is the linkage method in which the distance between each pair of datasets is added up and then divided by the total number of datasets to calculate the average distance between two clusters. It is also one of the most popular linkage methods.
- **Centroid Linkage**
  It is the linkage method in which the distance between the cen-

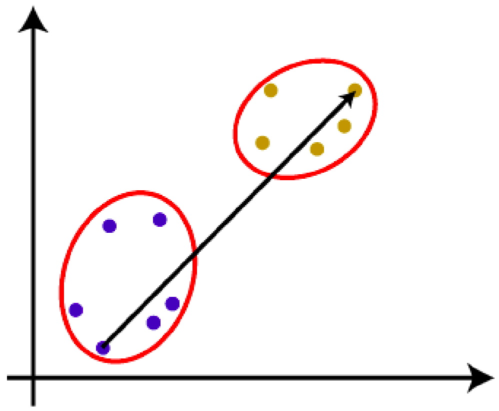

Figure 22:

troid of the clusters is calculated. Consider the below image.

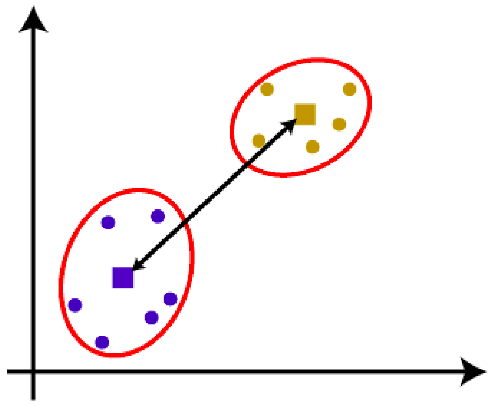

Figure 23:

From the above-given approaches, we can apply any of them according to the type of problem or business requirement.

Woking of Dendrogram in Hierarchical clustering

The dendrogram is a tree-like structure that is mainly used to store each step as a memory that the Hierarchical clustering algorithm performs. In the dendrogram plot, the Y-axis shows the Euclidean distances between the data points, and the x-axis shows all the data

points of the given dataset.
The working of the dendrogram can be explained in the figure 24.

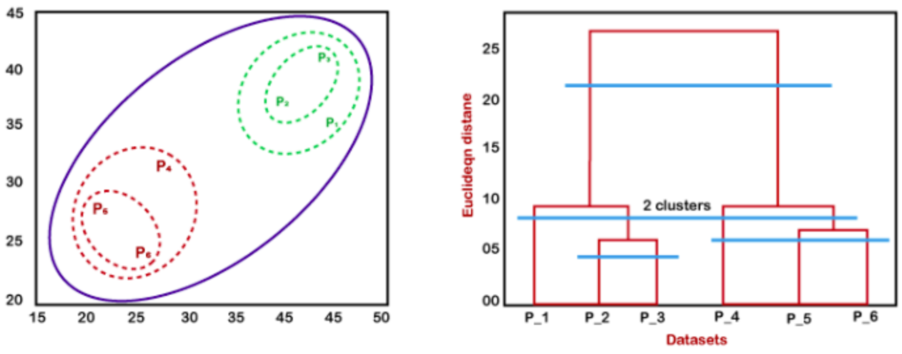

Figure 24:

Python Implementation of Agglomerative Hierarchical Clustering

we have a dataset of Mall_Customers, which is the data of customers who visit the mall and spend there.
In the given dataset, we have Customer_Id, Gender, Age, Annual Income ($), and Spending Score (which is the calculated value of how much a customer has spent in the mall, the more the value, the more he has spent).
The mall owner wants to find some patterns or some particular behavior of his customers using the dataset information.

Data pre-processing

We will import the libraries for our model, which is part of data pre-processing. The code is given below

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

In the above code, the numpy we have imported for the performing mathematics calculation, matplotlib is for plotting the graph, and pandas are for managing the dataset.
Next, we will import the dataset that we need to use. So here, we are

using the Mall_Customers.csv dataset. It can be imported using the below code.

```
# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
```

From the above dataset, we need to find some patterns in it.

Extracting the matrix of features

Here we don't need any dependent variable for data pre-processing step as it is a clustering problem, and we have no idea about what to determine. So we will just add a line of code for the matrix of features.

```
x = dataset.iloc[:, [3, 4]].values
```

Here we have extracted only 3 and 4 columns as we will use a 2D plot to see the clusters. So, we are considering the Annual income and spending score as the matrix of features.

Finding the optimal number of clusters using the Dendrogram

Now we will find the optimal number of clusters using the Dendrogram for our model. For this, we are going to use scipy library as it provides a function that will directly return the dendrogram for our code. Consider the below lines of code.

```
#Finding the optimal number of clusters using the dendrogram
import scipy.cluster.hierarchy as shc
dendro = shc.dendrogram(shc.linkage(x, method="ward"))
mtp.title("Dendrogrma Plot")
mtp.ylabel("Euclidean Distances")
mtp.xlabel("Customers")
mtp.show()
```

In the above lines of code, we have imported the hierarchy module of scipy library. This module provides us a method shc.denrogram(), which takes the linkage() as a parameter. The linkage function is used to define the distance between two clusters, so here we have passed the x(matrix of features), and method "ward," the popular method of linkage in hierarchical clustering.
The remaining lines of code are to describe the labels for the dendrogram plot.

By executing the above lines of code, we will get the output.As you can seen in the figure 25

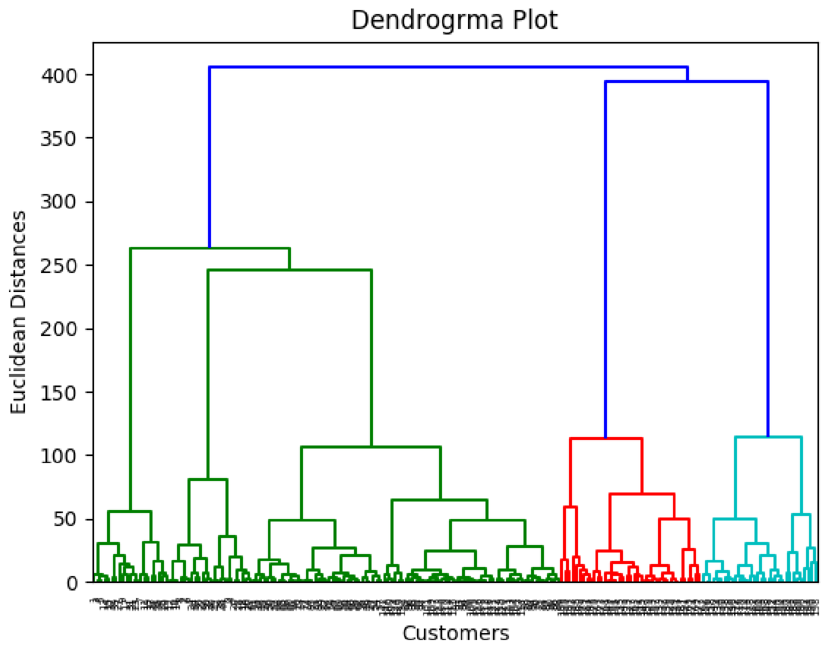

Figure 25:

Using this Dendrogram, we will now determine the optimal number of clusters for our model. For this, we will find the maximum vertical distance that does not cut any horizontal bar. Consider the figure 26.

In the Figure 26, we have shown the vertical distances that are not cutting their horizontal bars. As we can visualize, the 4th distance is looking the maximum, so according to this, the number of clusters will be 5(the vertical lines in this range).

Training the hierarchical clustering model

As we know the required optimal number of clusters, we can now train our model. The code is given below.

```
#training the hierarchical model on dataset
from sklearn.cluster import AgglomerativeClustering
```

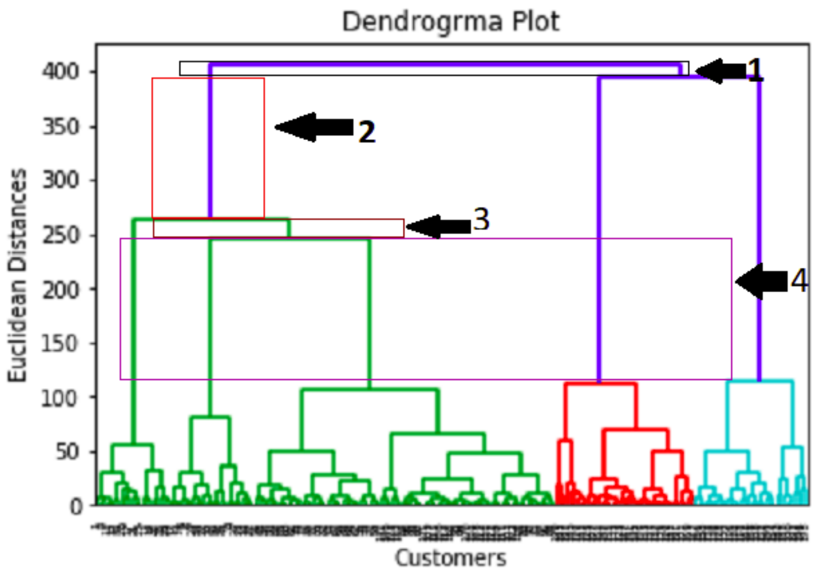

Figure 26:

```
hc= AgglomerativeClustering(n_clusters=5, affinity='euclidean',
linkage='ward')
y_pred= hc.fit_predict(x)
```

In the above code, we have imported the AgglomerativeClustering class of cluster module of scikit learn library.
Then we have created the object of this class named as hc. The AgglomerativeClustering class takes the following parameters.

- n_clusters=5: It defines the number of clusters, and we have taken here 5 because it is the optimal number of clusters.
- affinity='euclidean': It is a metric used to compute the linkage.
- linkage='ward': It defines the linkage criteria, here we have used the "ward" linkage. This method is the popular linkage method that we have already used for creating the Dendrogram. It reduces the variance in each cluster.

In the last line, we have created the dependent variable y_pred to fit or train the model. It does train not only the model but also returns the clusters to which each data point belongs.

Visualizing the clusters
As we have trained our model successfully, now we can visualize the clusters corresponding to the dataset.The code is given below.

```
#visulaizing the clusters
mtp.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s = 100,
 c = 'blue', label = 'Cluster 1')
mtp.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s = 100,
 c = 'green', label = 'Cluster 2')
mtp.scatter(x[y_pred== 2, 0], x[y_pred == 2, 1], s = 100,
 c = 'red', label = 'Cluster 3')
mtp.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s = 100,
 c = 'cyan', label = 'Cluster 4')
mtp.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s = 100,
 c = 'magenta', label = 'Cluster 5')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

Output:
By executing the above lines of code, we will get the diagram as shown in Figure 27.

b) Principal Component Analysis
Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the Principal Components. It is one of the popular tools that is used for exploratory data analysis and predictive modeling. It is a technique to draw strong patterns from the given dataset by reducing the variances.
PCA generally tries to find the lower-dimensional surface to project the high-dimensional data.
Example: We've often seen that certain features are more important or have more explanatory power than other features, for example, the

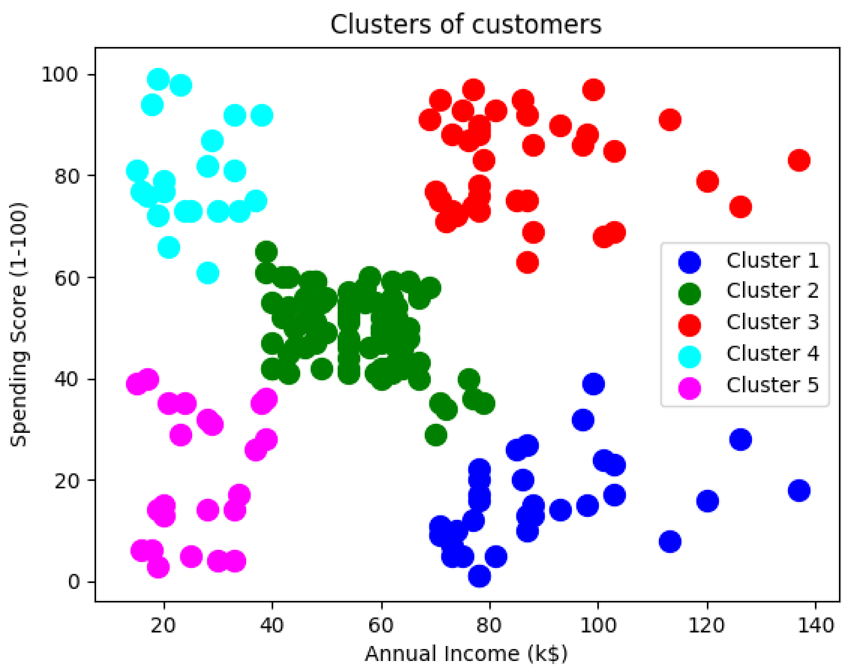
Clusters of customers
Spending Score (1-100)
Annual Income (k$)
0
20
40
60
80
100
20
40
60
80
100
120
140
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5

Figure 27:

size of a house is probably much more important than the color of a house when explaining the price of a house for sale.
Some real-world applications of PCA are image processing, movie recommendation system, optimizing the power allocation in various communication channels. It is a feature extraction technique, so it contains the important variables and drops the least important variable.
Some common terms used in PCA algorithm.

- **Dimensionality:**
  It is the number of features or variables present in the given dataset. More easily, it is the number of columns present in the dataset.
- **Dimensionality Reduction:**
  The number of input features, variables, or columns present in a given dataset is known as dimensionality, and the process to reduce these features is called dimensionality reduction.
  A dataset contains a huge number of input features in various cases, which makes the predictive modeling task more complicated. Because it is very difficult to visualize or make predictions for the training dataset with a high number of features, for such cases, dimensionality reduction techniques are required to use.
  Dimensionality reduction technique can be defined as, "It is a way of converting the higher dimensions dataset into lesser dimensions dataset ensuring that it provides similar information." These techniques are widely used in machine learning for obtaining a better fit predictive model while solving the classification and regression problems.
  It is commonly used in the fields that deal with high-dimensional data, such as speech recognition, signal processing, bioinformatics, etc. It can also be used for data visualization, noise reduction, cluster analysis, etc.
- **Correlation:**
  It signifies that how strongly two variables are related to each other. Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.
- **Orthogonal:**
  It defines that variables are not correlated to each other, and hence the correlation between the pair of variables is zero.

- **Covariance Matrix:**
  A matrix containing the covariance between the pair of variables is called the Covariance Matrix.

Steps for PCA algorithm

- **Getting the dataset:**
  Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.
- **Representing data into a structure:**
  Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.
- **Standardizing the data:**
  In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance. If the importance of features is independent of the variance of the feature, then we will divide each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.
- **Calculating the Covariance of Z:**
  To calculate the covariance of Z, we will take the matrix Z, and will transpose it. After transpose, we will multiply it by Z. The output matrix will be the Covariance matrix of Z.
- **Calculating the Eigen Values and Eigen Vectors:**
  Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.
- **Sorting the Eigen Vectors:**
  In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix M of eigenvalues. The resultant matrix will be named as M^*.
- **Calculating the new features Or Principal Components:**
  Here we will calculate the new features. To do this, we will multiply the M^* matrix to the Z. In the resultant matrix Z^*, each

observation is the linear combination of original features. Each column of the Z^* matrix is independent of each other.

- **Remove less or unimportant features from the new dataset:** The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

6 Extracting Features From text Data in NLP
Most classic machine learning algorithms can't take in raw string text as data.
Instead, what we need to do is perform some sort of feature extraction from the raw text in order to pass numerical features to the machine learning algorithm. So we're talking about feature extraction from raw texts. what we're discussing about is some sort of transformation to go from the variety of strings and words to numbers, something the machine learning model can understand.
Two main method of doing this.

- **Count vectorization:**
  In order to use textual data for predictive modeling, the text must be parsed to remove certain words this process is called tokenization.
  These words need to then be encoded as integers, or floating-point values, for use as inputs in machine learning algorithms. This process is called feature extraction (or vectorization).
  It is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text. This is helpful when we have multiple such texts, and we wish to convert each word in each text into vectors (for using in further text analysis). The code below shows how to use CountVectorizer.

```
from sklearn.feature_extraction.text import CountVectorizer

# list of text documents
text = ["John is a good boy. John watches basketball"]

vectorizer = CountVectorizer()
# tokenize and build vocab
```

```
vectorizer.fit(text)

print(vectorizer.vocabulary_)

# encode document
vector = vectorizer.transform(text)
# summarize encoded vector
print(vector.shape)
print(vector.toarray())

Output:
{'is': 3, 'john': 4, 'basketball': 0, 'boy': 1, 'good': 2,
 'watches': 5}
(1, 6)
[[1 1 1 1 2 1]]
```

- **TF-IDF(Term frequency - inverse document frequency)**
  TF-IDF model is one such method to represent words in numerical values. TF-IDF stands for Term Frequency Inverse Document Frequency.
  It does not assign equal value to all the words, hence important words that occur a few times will be assigned high weights.
  TF-IDF is the product of Term Frequency and Inverse Document Frequency. Heres the formula for TF-IDF calculation.
  TF-IDF = Term Frequency (TF) * Inverse Document Frequency (IDF)

  Term Frequency (TF) is the measure of the frequency of words in a document. It is the ratio of the number of times the word appears in a document compared to the total number of words in that document.
  tf(t,d) = count of t in d / number of words in d

  Inverse Document Frequency: The words that occur rarely in the corpus have a high IDF score. It is the log of the ratio of the number of documents to the number of documents containing the word.
  We take log of this ratio because when the corpus becomes large IDF values can get large causing it to explode hence taking log will dampen this effect. we cannot divide by 0, we smoothen the

value by adding 1 to the denominator.
idf(t) = log(N/(df + 1))

7 Where is Machine Learning used in Data Science?

The use of machine learning in data science can be understood by the development process or life cycle of Data Science. The different steps that occur in Data science lifecycle are as follows

- **Business Requirements:**
  In this step, we try to understand the requirement for the business problem for which we want to use it. Suppose we want to create a recommendation system, and the business requirement is to increase sales.
- **Data Acquisition:**
  In this step, the data is acquired to solve the given problem. For the recommendation system, we can get the ratings provided by the user for different products, comments, purchase history, etc.
- **Data Processing:**
  In this step, the raw data acquired from the previous step is transformed into a suitable format, so that it can be easily used by the further steps.
- **Data Exploration:**
  It is a step where we understand the patterns of the data, and try to find out the useful insights from the data.
- **Modeling:**
  The data modeling is a step where machine learning algorithms are used. So, this step includes the whole machine learning process. The machine learning process involves importing the data, data cleaning, building a model, training the model, testing the model, and improving the model's efficiency.
- **Deployment:**
  This is the last step where the model is deployed on an actual project, and the performance of the model is checked.