 **Lab Record**

**Introduction To Data Science**

**Course Code: CSE435**

***DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING***

**Amity School of Engineering & Technology**

**Amity University Uttar Pradesh**

**Kartik Bhatt**

**A2345919022**

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No** | **Programs** | **Date** | **Page no.** |
| **1** | Data Pre-processing (Find Outliers, Remove Outliers, Find Null values and Remove Null values) | 11/Aug/2022 | 3 |
| **2** | WAP to implement Simple linear regression | 31/Aug/2022 | 6 |
| **3** | WAP to implement k-means, DBSCAN, hierarchal clustering | 7/Sept/2022 | 11 |
| **4** | WAP to implement Logistic regression | 15/Sept/2022 | 14 |
| **5** | WAP to implement Principal Component Analysis (PCA) for dimensionality reduction. | 22/Sept/2022 | 17 |
| **6** | WAP to implement Naive Bayes Classifier | 29/Sept/2022 | 20 |
| **7** | WAP to implement Support Vector Machines (SVM) | 8/Oct/2022 | 22 |
| **8** | WAP to implement multivariate regression | 22/Oct/2022 | 25 |
| **9** | WAP to implement KNN | 28/Oct/2022 | 29 |
| **10** | WAP to implement Decision Tree | 3/Nov/2022 | 32 |

**Experiment 1**

**Aim-** Write a program for the following.

1. Find Outliers
2. Remove Outliers
3. Find Null values
4. Remove Null values

**Source Code and Output –**

1. Find and Remove Outliers

import numpy

import pandas

# Outlier Function start

def Outlier(temp, type):

print(f'Total number of entires in {type} is {len(temp)}')

# Q1 and Q1 are provided by pandas as 25 percentile and 75 percentile

Q1 = temp.quantile(0.25)

Q3 = temp.quantile(0.75)

IQR = Q3 - Q1

LowerLimit = Q1 - 1.5 \* IQR

UpperLimit = Q3 + 1.5 \* IQR

outlier = []

filtered = []

for x in temp:

if((x > UpperLimit) or (x < LowerLimit)):

outlier.append(x)

else:

filtered.append(x)

temp = filtered

print(f'Total number of Outliers in {type} is {len(outlier)}')

print(f'Total number of entires after removing outliers in {type} is {len(temp)}\n\n') # Outlier Function end

# Read file and make copy

data = pandas.read\_csv('Toyota.csv')

dataCopy = data.copy()

# Find Most occuring Fuel

mostFuel = dataCopy['FuelType'].value\_counts().idxmax()

# Replace Nan with most Occuring Fuel

dataCopy['FuelType'] = dataCopy['FuelType'].replace(

numpy.nan, mostFuel)

# Replacing Diesel with 1, Petrol with 3 and CNG with 3

dataCopy['FuelType'] = dataCopy['FuelType'].replace(

{'Diesel': 1, 'Petrol': 2, 'CNG': 3})

# Parsing to int

dataCopy['FuelType'] = dataCopy['FuelType'].astype(numpy.int64)

# Replace ?? with NaN and then parse as float

dataCopy['KM'] = dataCopy['KM'].replace({'??': numpy.nan})

dataCopy['KM'] = dataCopy['KM'].astype(float)

# Replace ???? with NaN and then parse as float

dataCopy['HP'] = dataCopy['HP'].replace({'????': numpy.nan})

dataCopy['HP'] = dataCopy['HP'].astype(float)

# Replace string with number and parse to int

dataCopy['Doors'] = dataCopy['Doors'].replace(

{'three': 3, 'four': 4, 'five': 5})

dataCopy['Doors'] = dataCopy['Doors'].astype(numpy.int64)

NumValue = ['Price', 'Age', 'KM', 'FuelType', 'HP',

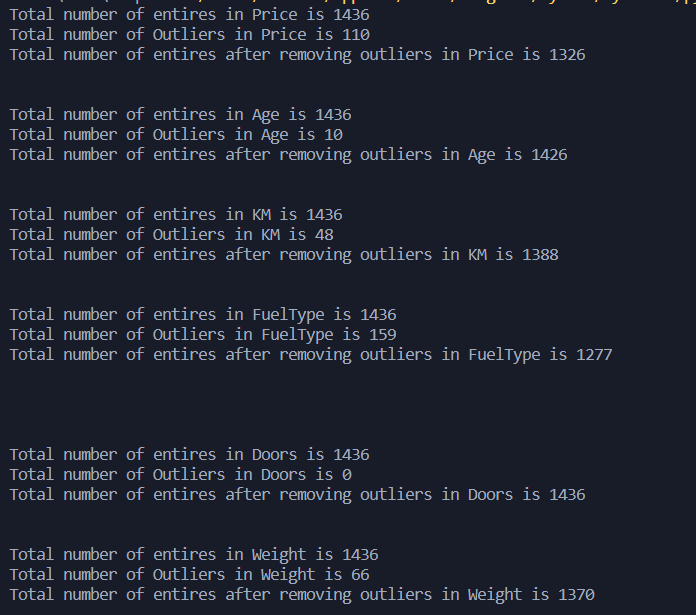
'MetColor', 'Automatic', 'CC', 'Doors', 'Weight']

# Removing NAN from numeric

for x in NumValue:

dataCopy[x] = dataCopy[x].replace(numpy.nan, dataCopy[x].mean())

Outlier(dataCopy[x], x)



**Experiment 2**

Aim: WAP to implement Simple linear regression

Theory:

Simple Linear Regression is a type of Regression algorithms that models the relationship between a dependent variable and a single independent variable. The relationship shown by a Simple Linear Regression model is linear or a sloped straight line, hence it is called Simple Linear Regression.

The key point in Simple Linear Regression is that the ***dependent variable must be a continuous/real value***. However, the independent variable can be measured on continuous or categorical values.

Simple Linear regression algorithm has mainly two objectives:

* **Model the relationship between the two variables.** Such as the relationship between Income and expenditure, experience and Salary, etc.
* **Forecasting new observations.** Such as Weather forecasting according to temperature, Revenue of a company according to the investments in a year, etc.

The Simple Linear Regression model can be represented using the below equation:

y= a0+a1x+ ε

Where,

**a0= It is the intercept of the Regression line (can be obtained putting x=0)**  
**a1= It is the slope of the regression line, which tells whether the line is increasing or decreasing.**  
**ε = The error term. (For a good model it will be negligible)**

Tools used: pandas, numpy, scipy

Text

Description automatically generatedText

Description automatically generatedChart, scatter chart

Description automatically generated

**Experiment 3**

Aim- WAP to implement k-means, DBSCAN, hierarchal clustering

Library Used – pandas, numpy , scipy

Theory:

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

K-Means Clustering is an [Unsupervised Learning algorithm](https://www.javatpoint.com/unsupervised-machine-learning), which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means [clustering](https://www.javatpoint.com/clustering-in-machine-learning) algorithm mainly performs two tasks:

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

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

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



K-means:

from sklearn.cluster import KMeans

import matplotlib.pyplot as plot

import pandas

import numpy

data = pandas.read\_csv('toyota.csv')

# Replace ?? with NaN and then parse as float

data['KM'] = data['KM'].replace({'??': numpy.nan})

data['KM'] = data['KM'].astype(float)

# Removing NAN from KM

data['KM'] = data['KM'].replace(numpy.nan, data['KM'].mean())

data['Price'] = data['Price'].astype(int)

data['KM'] = data['KM'].astype(int)

data.head()

feature = data.iloc[:, [1, 3]].values

wcss = []  # Within-Cluster Sum of Square

for i in range(1, 11):

    kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state=42)

    kmeans.fit(feature)

    wcss.append(kmeans.inertia\_)

plot.plot(range(1, 11), wcss)

plot.title('The Elobw Method Graph')

plot.xlabel('Number of clusters(k)')

plot.ylabel('wcss')

plot.show()

kmeans = KMeans(n\_clusters=3, init='k-means++', random\_state=42)

yPredict = kmeans.fit\_predict(feature)

plot.scatter(feature[yPredict == 0, 0], feature[yPredict == 0, 1], s = 10, c = 'red', label = 'Cluster 1') # first cluster

plot.scatter(feature[yPredict == 1, 0], feature[yPredict == 1, 1], s = 10, c = 'green', label = 'Cluster 2') # second cluster

plot.scatter(feature[yPredict== 2, 0], feature[yPredict == 2, 1], s = 10, c = 'blue', label = 'Cluster 3') # third cluster

plot.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 30, c = 'cyan', label = 'Centroid')   # centroid

plot.title('Clusters of Toyota')

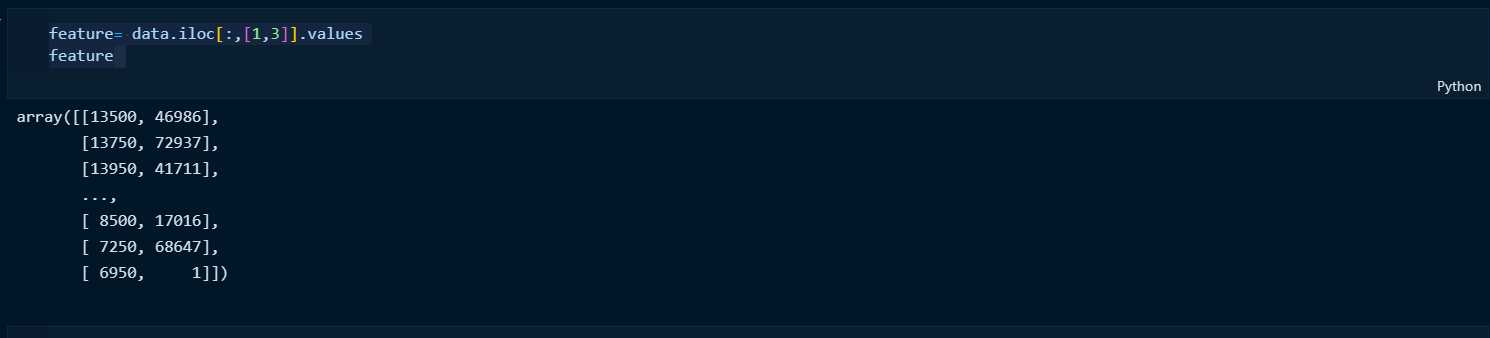
plot.xlabel('Price')

plot.ylabel('KM')

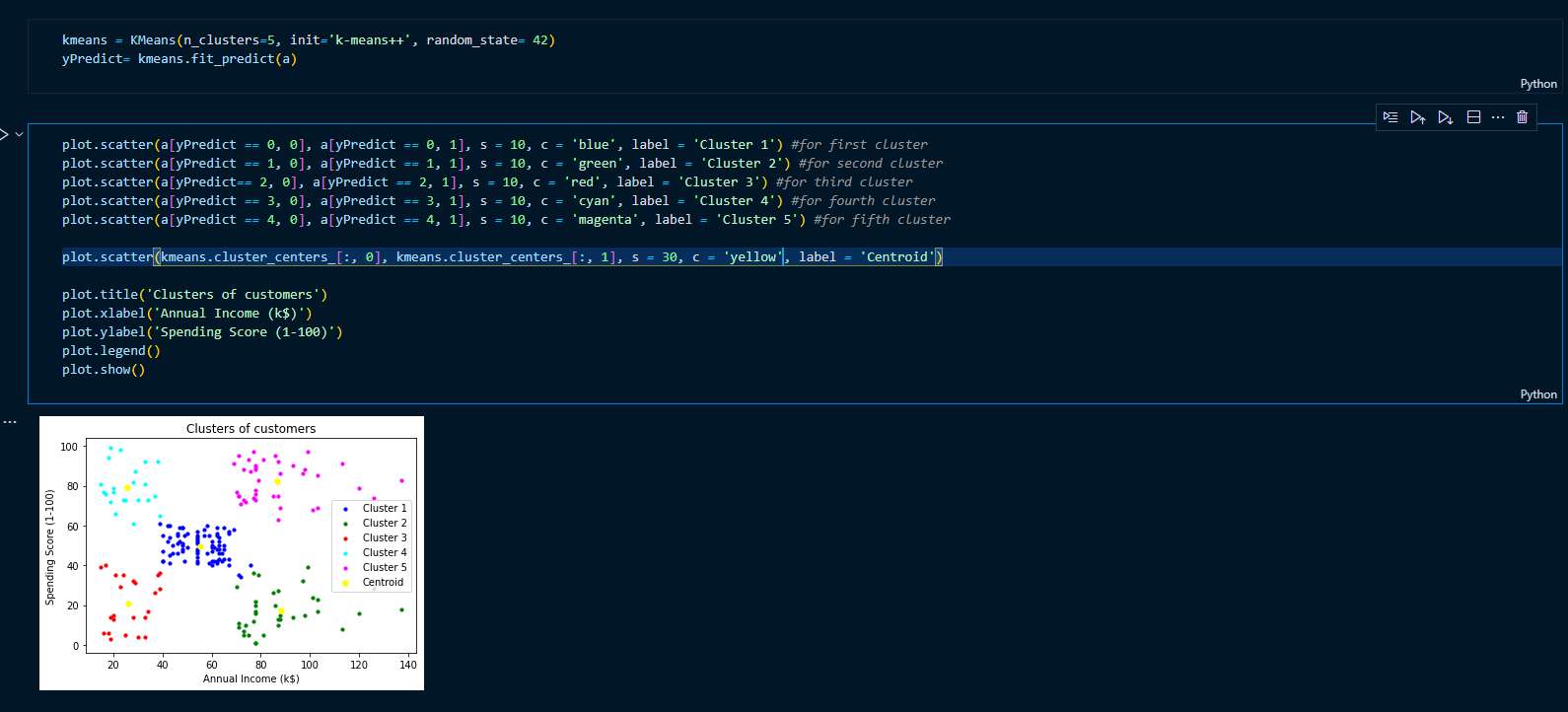
plot.legend()

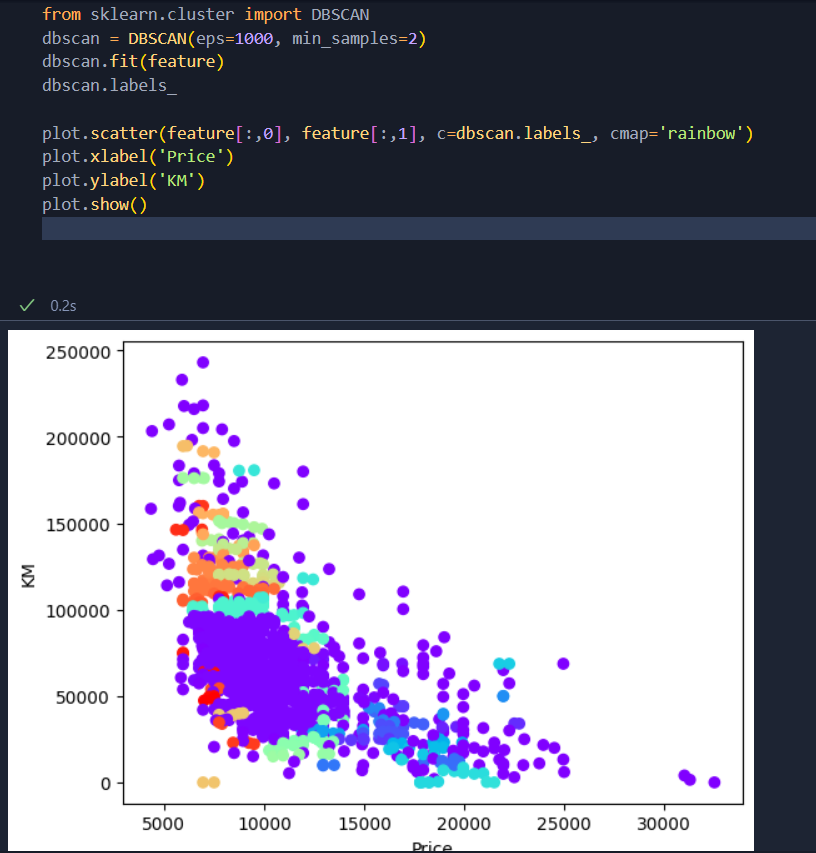
plot.show()









DBSCAN: 

Hierarchical clustering:

Hierarchical clustering is another 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**.

Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. As there is no requirement to predetermine the number of clusters as we did in the K-Means algorithm.

The hierarchical clustering technique has two approaches:

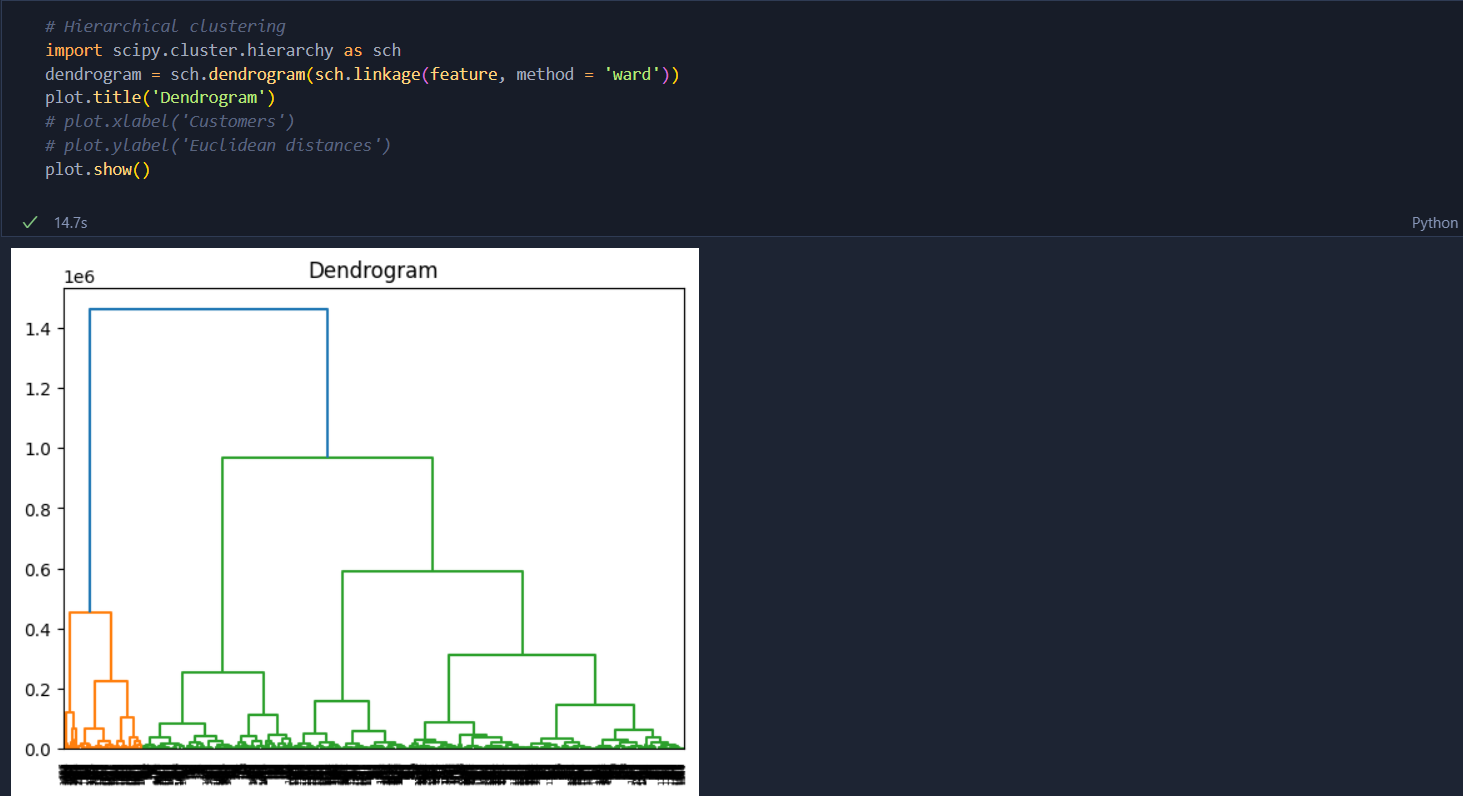
**Agglomerative:** Agglomerative is a **bottom-up** approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.

**Divisive:** Divisive algorithm is the reverse of the agglomerative algorithm as it is a **top-down approach.**

As we already have other [clustering](https://www.javatpoint.com/clustering-in-machine-learning) algorithms such as [**K-Means Clustering**](https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning), then why we need hierarchical clustering? So, as we have seen in the K-means clustering that there are some challenges with this algorithm, which are a predetermined number of clusters, and it always tries to create the clusters of the same size. To solve these two challenges, we can opt for the hierarchical clustering algorithm because, in this algorithm, we don't need to have knowledge about the predefined number of clusters.

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.



**Experiment 4**

Aim: WAP to implement Logistic regression

Tools used: pandas, numpy, sklearn, matplotlib,

Theory:

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). The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.

Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets. Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



## Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

* We know the equation of the straight line can be written as:

Logistic Regression in Machine Learning

* In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

Logistic Regression in Machine Learning

* But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

Logistic Regression in Machine Learning

Code:

dataset = pd.read\_csv("User\_Data.csv")

# input

x = dataset.iloc[:, [2, 3]].values

# output

y = dataset.iloc[:, 4].values

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)

from sklearn.preprocessing import StandardScaler

sc\_x = StandardScaler()

xtrain = sc\_x.fit\_transform(xtrain)

xtest = sc\_x.transform(xtest)

print (xtrain[0:10, :])

Output[1]:

[[ 0.58164944 -0.88670699]  
 [-0.60673761 1.46173768]  
 [-0.01254409 -0.5677824 ]  
 [-0.60673761 1.89663484]  
 [ 1.37390747 -1.40858358]  
 [ 1.47293972 0.99784738]  
 [ 0.08648817 -0.79972756]  
 [-0.01254409 -0.24885782]  
 [-0.21060859 -0.5677824 ]  
 [-0.21060859 -0.19087153]]

#Training the model

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

classifier.fit(xtrain, ytrain)

y\_pred = classifier.predict(xtest)

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(ytest, y\_pred)

print ("Confusion Matrix : \n", cm)

Output:

Confusion Matrix :   
 [[65 3]  
 [ 8 24]]

Code:

from matplotlib.colors import ListedColormap

X\_set, y\_set = xtest, ytest

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1,

stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1,

stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(

np.array([X1.ravel(), X2.ravel()]).T).reshape(

X1.shape), 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)

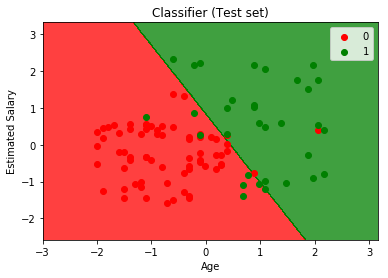
plt.title('Classifier (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()



**Experiment 5**

Aim: WAP to implement Principal Component Analysis (PCA) for dimensality reduction.

Tools used: pandas, numpy, sklearn, matplotlib

Theory:

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in [machine learning](https://www.javatpoint.com/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.

PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. 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.

The PCA algorithm is based on some mathematical concepts such as:

* Variance and Covariance
* Eigenvalues and Eigen factors

PCA is mainly used as the dimensionality reduction technique in various AI applications such **as computer vision, image compression, etc.**

It can also be used for finding hidden patterns if data has high dimensions. Some fields where PCA is used are Finance, data mining, Psychology, etc.

Code:

import pandas as pd

import matplotlib.pyplot as plt

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# load dataset into Pandas DataFrame

df = pd.read\_csv(url, names=['sepal length','sepal width','petal length','petal width','target'])

from sklearn.preprocessing import StandardScaler

features = ['sepal length', 'sepal width', 'petal length', 'petal width']

# Separating out the features

x = df.loc[:, features].values

# Separating out the target

y = df.loc[:,['target']].values

# Standardizing the features

x = StandardScaler().fit\_transform(x)

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

principalComponents = pca.fit\_transform(x)

principalDf = pd.DataFrame(data = principalComponents

, columns = ['principal component 1', 'principal component 2'])

finalDf = pd.concat([principalDf, df[['target']]], axis = 1)

FinalDf

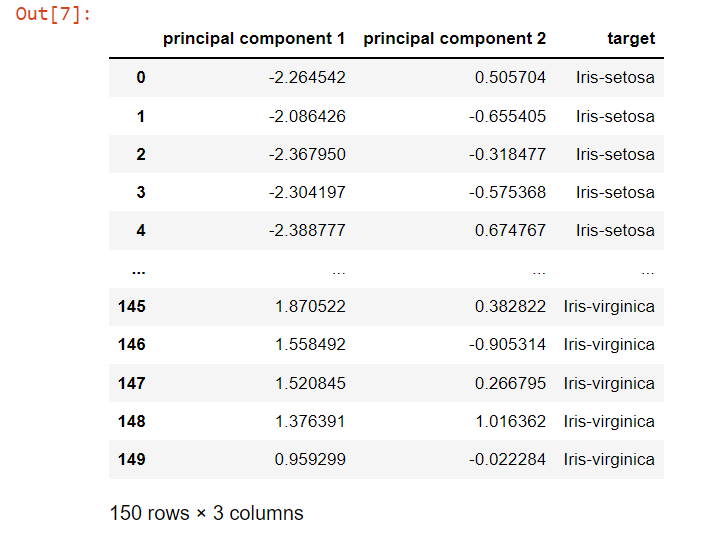


fig = plt.figure(figsize = (8,8))

ax = fig.add\_subplot(1,1,1)

ax.set\_xlabel('Principal Component 1', fontsize = 15)

ax.set\_ylabel('Principal Component 2', fontsize = 15)

ax.set\_title('2 component PCA', fontsize = 20)

targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

colors = ['r', 'g', 'b']

for target, color in zip(targets,colors):

indicesToKeep = finalDf['target'] == target

ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']

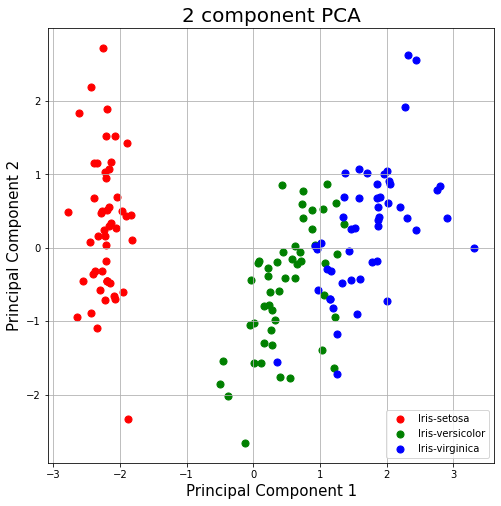
, finalDf.loc[indicesToKeep, 'principal component 2']

, c = color

, s = 50)

ax.legend(targets)

ax.grid()



pca.explained\_variance\_ratio\_



**Experiment 6**

Aim: WAP to implement Naive Bayes Classifier

Tools used: Sklearn, GaussianNB, sklearn.naive\_bayes

Theory:

Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems. It is mainly used in *text classification* that includes a high-dimensional training dataset.

Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.

**It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.

Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

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

* **Naïve**: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
* **Bayes**: It is called Bayes because it depends on the principle of [Bayes' Theorem](https://www.javatpoint.com/bayes-theorem-in-artifical-intelligence).

## Bayes' Theorem:

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

Naïve Bayes Classifier Algorithm

**Where,**

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

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

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

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

Code:

# Importing the dataset.

from sklearn.datasets import load\_iris

iris = load\_iris()

# Assigning Variables to store data

X = iris.data

y = iris.target

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=1)

# training the model

from sklearn.naive\_bayes import GaussianNB

gnb = GaussianNB()

gnb.fit(X\_train, y\_train)



# making predictions

y\_pred = gnb.predict(X\_test)

# comparing actual response values (y\_test) with predicted response values (y\_pred)

from sklearn import metrics

print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy\_score(y\_test, y\_pred)\*100)



**Experiment 7**

Aim: WAP to implement Support Vector Machines (SVM)

Tools used: numpy, cvxopt, sklearn, matplotlib

Theory:

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

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

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



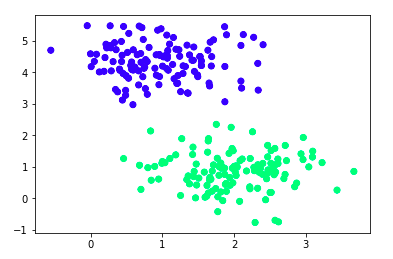
Code:

import numpy as np  
import cvxopt  
from sklearn.datasets.samples\_generator import make\_blobs  
from sklearn.model\_selection import train\_test\_split  
from matplotlib import pyplot as plt  
from sklearn.svm import LinearSVC  
from sklearn.metrics import confusion\_matrix

class SVM:def fit(self, X, y):  
 n\_samples, n\_features = X.shape# P = X^T X  
 K = np.zeros((n\_samples, n\_samples))  
 for i in range(n\_samples):  
 for j in range(n\_samples):  
 K[i,j] = np.dot(X[i], X[j])P = cvxopt.matrix(np.outer(y, y) \* K)# q = -1 (1xN)  
 q = cvxopt.matrix(np.ones(n\_samples) \* -1)# A = y^T   
 A = cvxopt.matrix(y, (1, n\_samples))# b = 0   
 b = cvxopt.matrix(0.0)# -1 (NxN)  
 G = cvxopt.matrix(np.diag(np.ones(n\_samples) \* -1))# 0 (1xN)  
 h = cvxopt.matrix(np.zeros(n\_samples))solution = cvxopt.solvers.qp(P, q, G, h, A, b)# Lagrange multipliers  
 a = np.ravel(solution['x'])# Lagrange have non zero lagrange multipliers  
 sv = a > 1e-5  
 ind = np.arange(len(a))[sv]  
 self.a = a[sv]  
 self.sv = X[sv]  
 self.sv\_y = y[sv]# Intercept  
 self.b = 0  
 for n in range(len(self.a)):  
 self.b += self.sv\_y[n]  
 self.b -= np.sum(self.a \* self.sv\_y \* K[ind[n], sv])  
 self.b /= len(self.a)# Weights  
 self.w = np.zeros(n\_features)  
 for n in range(len(self.a)):  
 self.w += self.a[n] \* self.sv\_y[n] \* self.sv[n]  
   
 def project(self, X):  
 return np.dot(X, self.w) + self.b  
   
   
 def predict(self, X):  
 return np.sign(self.project(X))

X, y = make\_blobs(n\_samples=250, centers=2,  
 random\_state=0, cluster\_std=0.60)y[y == 0] = -1tmp = np.ones(len(X))y = tmp \* y

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='winter')



X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)

svm = SVM()

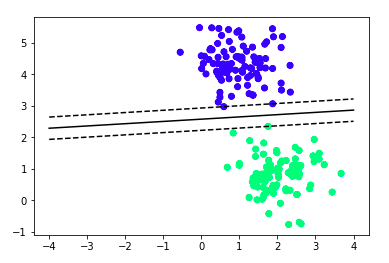
svm.fit(X\_train, y\_train)

def f(x, w, b, c=0):  
 return (-w[0] \* x - b + c) / w[1]plt.scatter(X\_train[:, 0],

X\_train[:, 1], c=y\_train, cmap='winter')# w.x + b = 0  
a0 = -4; a1 = f(a0, svm.w, svm.b)  
b0 = 4; b1 = f(b0, svm.w, svm.b)

plt.plot([a0,b0], [a1,b1], 'k')# w.x + b = 1  
a0 = -4; a1 = f(a0, svm.w, svm.b, 1)  
b0 = 4; b1 = f(b0, svm.w, svm.b, 1)

plt.plot([a0,b0], [a1,b1], 'k--')# w.x + b = -1  
a0 = -4; a1 = f(a0, svm.w, svm.b, -1)  
b0 = 4; b1 = f(b0, svm.w, svm.b, -1)  
plt.plot([a0,b0], [a1,b1], 'k--')



y\_pred = svm.predict(X\_test)confusion\_matrix(y\_test, y\_pred)

Output: array ([29, 0],

[0, 34]}

**Experiment 8**

Aim: WAP to implement multivariate regression

Tools used: pandas, numpy, scipy

import matplotlib.pyplot as plot

import pandas

import numpy

data = pandas.read\_csv('toyota.csv')

*# Replace ?? with NaN and then parse as float*

data['KM'] = data['KM'].replace({'??': numpy.nan})

data['KM'] = data['KM'].astype(float)

*# Replace ???? with NaN and then parse as float*

data['HP'] = data['HP'].replace({'????': numpy.nan})

data['HP'] = data['HP'].astype(float)

*# Replace string with number and parse to int*

data['Doors'] = data['Doors'].replace(

    {'three': 3, 'four': 4, 'five': 5})

data['Doors'] = data['Doors'].astype(numpy.int64)

*# Remove NAN*

data['KM'] = data['KM'].replace(numpy.nan, data['KM'].mean())

data['HP'] = data['HP'].replace(numpy.nan, data['HP'].mean())

data['Doors'] = data['Doors'].replace(numpy.nan, data['Doors'].mean())

data['Weight'] = data['Weight'].replace(numpy.nan, data['Weight'].mean())

from sklearn import linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

*# Split data into training and test sets*

train = data.sample(frac=0.8, random\_state=200)

test = data.drop(train.index)

*# Multiple linear regression*

regr = linear\_model.LinearRegression()

train\_x = numpy.array(train[['KM', 'HP', 'Doors', 'Weight']])

train\_y = numpy.array(train['Price']).reshape(-1, 1)

regr.fit(train\_x, train\_y)

*# Check accurary of the model*

test\_x = numpy.array(test[['KM', 'HP', 'Doors', 'Weight']])

test\_y = numpy.array(test['Price']).reshape(-1, 1)

test\_y\_pred = regr.predict(test\_x)

*# The coefficients*

print('Coefficients: ', regr.coef\_)

print('Intercept: ', regr.intercept\_)

*# The mean squared error*

print("Mean squared error: %.2f" % mean\_squared\_error(test\_y, test\_y\_pred))

*# Explained variance score: 1 is perfect prediction*

print('Variance score: %.2f' % r2\_score(test\_y, test\_y\_pred))

*# Plot outputs*

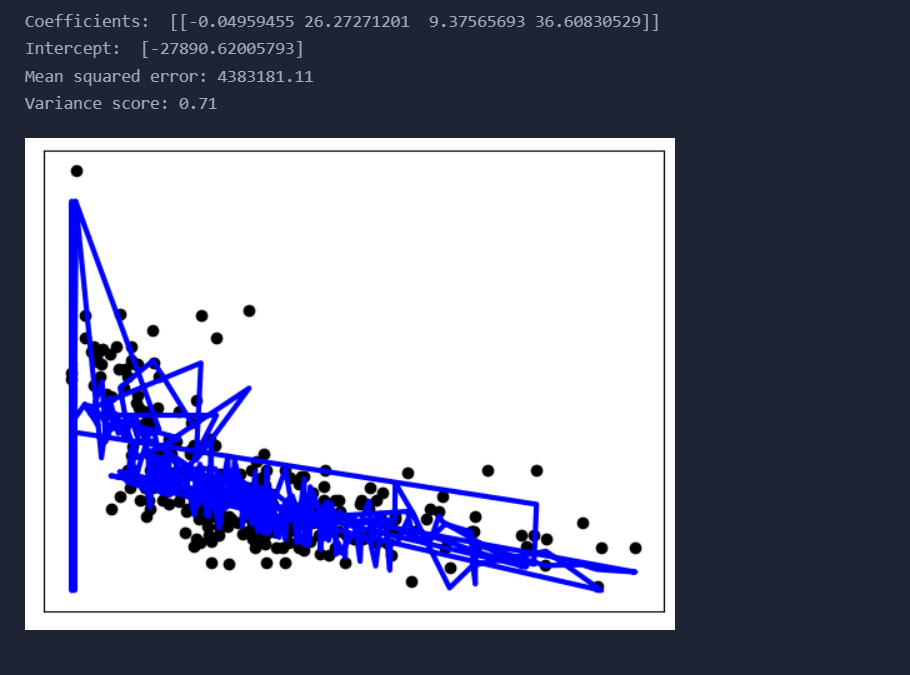
plot.scatter(test['KM'], test['Price'], color='black')

plot.plot(test\_x, test\_y\_pred, color='blue', linewidth=3)

plot.xticks(())

plot.yticks(())

plot.show()



**Experiment 9**

Aim: WAP to implement KNN

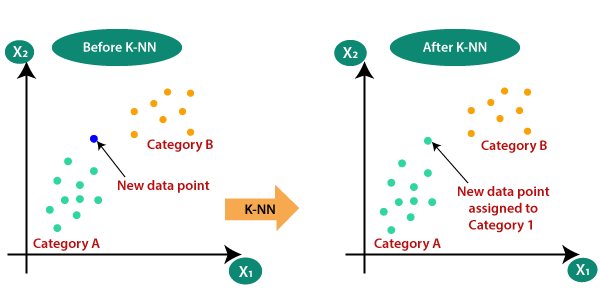
Tools used: Sklearn

Theory:

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm. K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data. It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

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



Code:

import pandas as pd  
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_classification

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn import metrics

X,Y=make\_classification(n\_samples= 200,n\_features=8,n\_informative=8,n\_redundant=0,n\_repeated=0,n\_classes=2,random\_state=14)

X\_train, X\_test, y\_train, y\_test= train\_test\_split(X, Y, test\_size= 0.2,random\_state=32)

sc= StandardScaler()

sc.fit(X\_train)

X\_train= sc.transform(X\_train)

sc.fit(X\_test)

X\_test= sc.transform(X\_test)

print(X.shape)

error1= []  
error2= []  
for k in range(1,15):  
 knn= KNeighborsClassifier(n\_neighbors=k)  
 knn.fit(X\_train,y\_train)  
 y\_pred1= knn.predict(X\_train)  
 error1.append(np.mean(y\_train!= y\_pred1))  
 y\_pred2= knn.predict(X\_test)  
 error2.append(np.mean(y\_test!= y\_pred2))  
# plt.figure(figsize(10,5))  
plt.plot(range(1,15),error1,label="train")  
plt.plot(range(1,15),error2,label="test")  
plt.xlabel('k Value')  
plt.ylabel('Error')  
plt.legend()

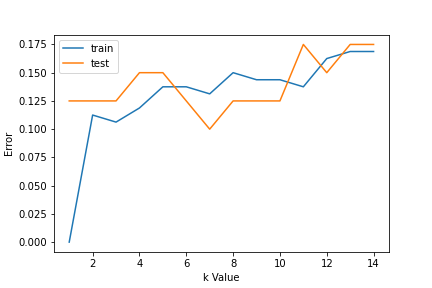


Fig: train and test error curve

knn= KNeighborsClassifier(n\_neighbors=7)  
knn.fit(X\_train,y\_train)  
y\_pred= knn.predict(X\_test)  
metrics.accuracy\_score(y\_test,y\_pred)

OUTPUT: 0.9

**Experiment - 10**

**Aim: WAP to implement Decision Tree**

**Tools used: Sklearn**

**Theory:**

Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**

In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

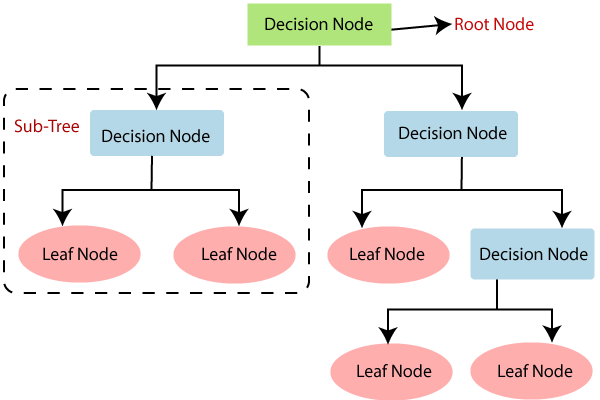
The decisions or the test are performed on the basis of features of the given dataset.

***It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.***

It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

In order to build a tree, we use the **CART algorithm,** which stands for **Classification and Regression Tree algorithm.**

A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.



While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

* **Information Gain**
* **Gini Index**

Code:

**import** pydotplus  
 **from** sklearn.datasets **import** load\_iris  
 **from** sklearn **import** tree  
 **from** IPython.display **import** Image, display  
\_\_author\_\_ = "Mayur Kulkarni [<mayur.kulkarni@xoriant.com](mailto:%3cmayur.kulkarni@xoriant.com)>"  
   
 **def load\_data\_set**():  
   
iris = load\_iris()  
 **return** iris  
   
 **def train\_model**(iris):  
   
clf = tree.DecisionTreeClassifier()  
 clf = clf.fit(iris.data, iris.target)  
 **return** clf  
   
 **def display\_image**(clf, iris):  
   
dot\_data = tree.export\_graphviz(clf, out\_file=None,  
 feature\_names=iris.feature\_names,  
 class\_names=iris.target\_names,  
 filled=True, rounded=True)  
   
 graph = pydotplus.graph\_from\_dot\_data(dot\_data)  
 display(Image(data=graph.create\_png()))  
   
 **if** \_\_name\_\_ == '\_\_main\_\_':  
 iris\_data = load\_iris()  
 decision\_tree\_classifier = train\_model(iris\_data)  
 display\_image(clf=decision\_tree\_classifier, iris=iris\_data)

