1. Write a program to implement Decision tree using Python

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

# Importing the dataset using pandas module
dataset = pd.read_csv('D:\mca pdf\Decision_tree.csv')

# splitting the dataset into input and output datasets
X = dataset.iloc[:, [0,1]].values
y = dataset.iloc[:, 2].values

# splitting the dataaset into Training and Testing Data
from sklearn.model_selection import train_test_split

# random state is 0 and test size if 25%
X_train, X_test, y_train, y_test =train_test_split(X,y,test_size=0.25, random_state=0)

# importing standard scalling method from sklearn
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

# providing the inputs for the scalling purpose
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# importing decision tree algorithm
from sklearn.tree import DecisionTreeClassifier
```

```
# entropy means information gain
classifer=DecisionTreeClassifier(criterion='entropy', random_state=0)

# providing the training dataset
classifer.fit(X_train,y_train)
y_pred= classifer.predict(X_test)

# creating confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)

from sklearn.metrics import accuracy_score
print("Accuracy: ", accuracy_score(y_test,y_pred))
```

#OUTPUT:-

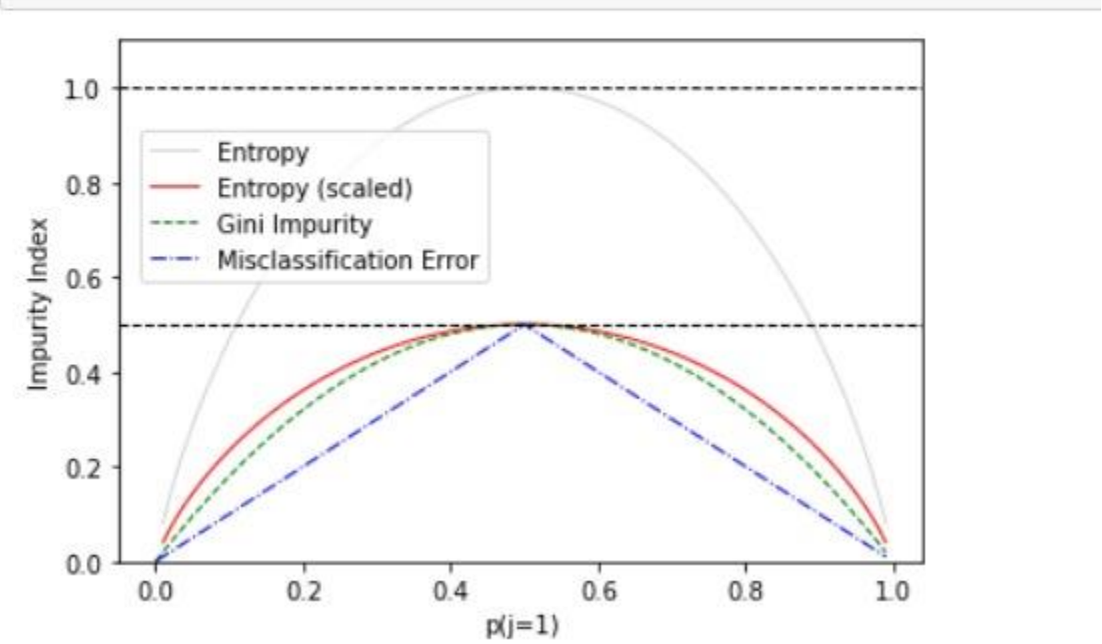
Accuracy: 0.506

2. Write program to calculate any one of the attribute selection measures (ASM) like Information Gain, Gain Ratio, and Gini Index etc. for decision tree.

```
import matplotlib.pyplot as plt

import numpy as np


def gini(p):

    return (p)*(1 - (p)) + (1 - p)*(1 - (1-p))


def entropy(p):

    return - p*np.log2(p) - (1 - p)*np.log2((1 - p))


def classification_error(p):

    return 1 - np.max([p, 1 - p])


x = np.arange(0.0, 1.0, 0.01)

ent = [entropy(p) if p != 0 else None for p in x]

scaled_ent = [e*0.5 if e else None for e in ent]

c_err = [classification_error(i) for i in x]


fig = plt.figure()

ax = plt.subplot(111)


for j, lab, ls, c, in zip(

        [ent, scaled_ent, gini(x), c_err],

        ['Entropy', 'Entropy (scaled)', 'Gini Impurity', 'Misclassification Error'],

        ['-', '-', '--', '-.'],

        ['lightgray', 'red', 'green', 'blue']):

    line = ax.plot(x, j, label=lab, linestyle=ls, lw=1, color=c)
```

```
ax.legend(loc='upper left', bbox_to_anchor=(0.01, 0.85),
          ncol=1, fancybox=True, shadow=False)

ax.axhline(y=0.5, linewidth=1, color='k', linestyle='--')
ax.axhline(y=1.0, linewidth=1, color='k', linestyle='--')

plt.ylim([0, 1.1])
plt.xlabel('p(j=1)')
plt.ylabel('Impurity Index')
plt.show()
```

Output :-



3.Implement simple KNN using Euclidean distance in python.

```
import math
import csv
with open(r'D:\iris.data') as csvfile:
    lines = csv.reader(csvfile)


import random


def handleDataset(filename, split, trainingSet=[], testSet=[]):
    with open(filename, 'r') as csvfile:
        lines = csv.reader(csvfile)
        dataset = list(lines)
        for x in range(len(dataset)-1):
            for y in range(4):
                dataset[x][y] = float(dataset[x][y])
            if random.random() < split:
                trainingSet.append(dataset[x])
            else:
                testSet.append(dataset[x])


trainingSet = []
testSet = []
handleDataset(r'D:\iris.data.', 0.66, trainingSet, testSet)
print('Train: ' + repr(len(trainingSet)))
print('Test: ' + repr(len(testSet)))
```

```
def euclideanDistance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow((instance1[x] - instance2[x]), 2)
    return math.sqrt(distance)


data1 = [2, 2, 2, 'a']
data2 = [4, 4, 4, 'b']
distance = euclideanDistance(data1, data2, 3)
print('Distance: ' + repr(distance))
```

OUTPUT :-

Train: 108

Test: 42

Distance: 3.4641016151377544

4. Write a program to implement k-Nearest Neighbour algorithm.

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

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

datasets = pd.read_csv("D:\machine Learning\Csv files\iris.data",names = names)

datasets.head()

x = datasets.iloc[:, :-1].values
y = datasets.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=0.20)


from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(x_train)

x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)


from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
```

```
classifier.fit(x_train, y_train)

y_pred = classifier.predict(x_test)

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

OUTPUT :-

```
[[9 0 0]
 [0 9 3]
 [0 1 8]]
              precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         9
Iris-versicolor       0.90      0.75      0.82        12
 Iris-virginica       0.73      0.89      0.80         9

       accuracy                           0.87        30
      macro avg       0.88      0.88      0.87        30
   weighted avg       0.88      0.87      0.87        30
```

5. Write a program to implement the naïve Bayesian classifier for a sample training dataset.

```
from sklearn.datasets import load_iris
iris = load_iris()

# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target

# splitting X and y into training and testing sets
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 on training set
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# making predictions on the testing set
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)
```

OUTPUT :-

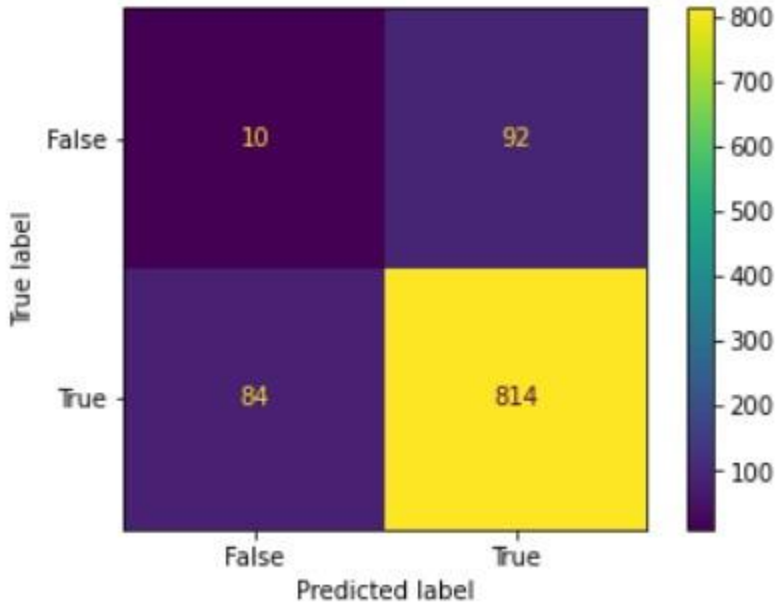
Gaussian Naive Bayes model accuracy(in %): 95.0

6 Program for Conffusion Matrix and calculate training dataset.

```
import matplotlib.pyplot as plt

import numpy

from sklearn import metrics


actual = numpy.random.binomial(1,.9,size = 1000)

predicted = numpy.random.binomial(1,.9,size = 1000)


confusion_matrix = metrics.confusion_matrix(actual, predicted)


cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = [False, True])


cm_display.plot()

plt.show()



#precision recall and f1 measure

Precision = metrics.precision_score(actual, predicted)

print(Precision)


Sensitivity_recall = metrics.recall_score(actual, predicted)


print(Sensitivity_recall)


F1_measure = metrics.f1_score(actual, predicted)
```

print(F1_measure)

OUTPUT :-




```
0.8984547461368654
0.9064587973273942
0.9024390243902439
```

7. Write program for linear regression and find parameters like Sum of Squared Errors (SSE)

```
import matplotlib
from matplotlib import style
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
#%matplotlib inline

dataset =pd.read_csv("D:\iris.cvs")
dataset.shape

dataset.head()
dataset.describe()

x= dataset.iloc[:,:-1].values
y=dataset.iloc[:,1].values

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =train_test_split(x,y,test_size=0.2, random_state=0)

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

print (regressor.intercept_)
print(regressor.coef_)
```

```
y_pred=regressor.predict(X_test)
df=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
df
from sklearn import metrics
print('Mean Absolute Error:',metrics.mean_absolute_error(y_test,y_pred))
print('Mean Squared Error:',metrics.mean_squared_error(y_test,y_pred))
print('Root Mean Squared Error:',np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

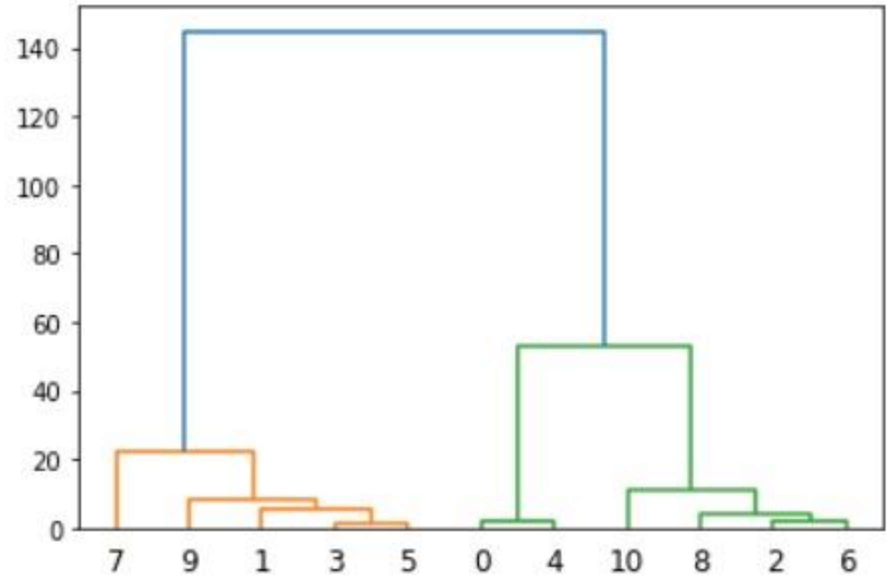
OUTPUT :-

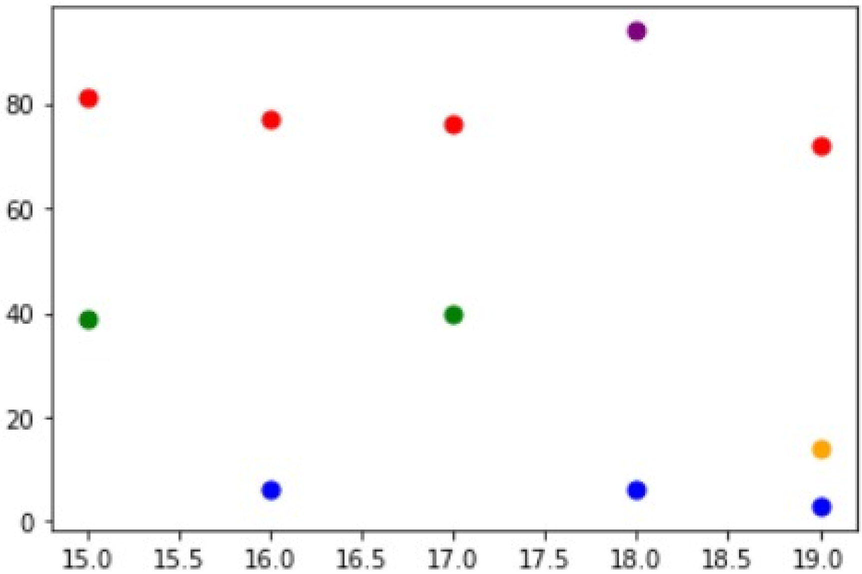
```
-4.440892098500626e-16
[-5.64989286e-17  1.00000000e+00  3.33066907e-16 -5.55111512e-17]
Mean Absolute Error: 9.917992353318065e-16
Mean Squared Error: 1.3607850615062454e-30
Root Mean Squared Error: 1.1665269227524264e-15
```

8. Implement Agglomerative Clustering in python.

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch

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

X = dataset.iloc[:, [3, 4]].values
dendrogram = sch.dendrogram(sch.linkage(X, method='ward'))
model = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
model.fit(X)
labels = model.labels_
plt.scatter(X[labels==0, 0], X[labels==0, 1], s=50, marker='o', color='red')
plt.scatter(X[labels==1, 0], X[labels==1, 1], s=50, marker='o', color='blue')
plt.scatter(X[labels==2, 0], X[labels==2, 1], s=50, marker='o', color='green')
plt.scatter(X[labels==3, 0], X[labels==3, 1], s=50, marker='o', color='purple')
plt.scatter(X[labels==4, 0], X[labels==4, 1], s=50, marker='o', color='orange')
plt.show()
```

OUTPUT :-





PROGRAM 8 OUTPUT

9. Write a Program to implement SVM.

```
import pandas as pd

import numpy as np

dataset =pd.read_csv("D:\iris.data")

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

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=0.20, random_state=0)


from sklearn.preprocessing import StandardScaler

sc_x= StandardScaler()

X_train = sc_x.fit_transform(X_train)

X_test = sc_x.transform(X_test)


from sklearn.svm import SVC

classifier = SVC(kernel="linear",random_state=0)

classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

y_pred

from sklearn.metrics import confusion_matrix

cm= confusion_matrix(y_test,y_pred)


from sklearn.metrics import accuracy_score

print("Accuracy: ",accuracy_score(y_test,y_pred))
```

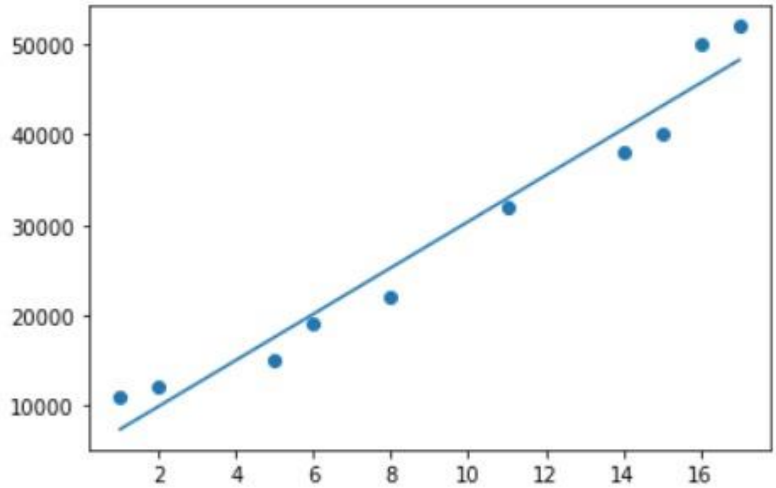
OUTPUT :-

Accuracy: 0.9

10. Implement Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

```
import matplotlib.pyplot as plt

from scipy import stats


x = [1,2,5,6,8,11,14,15,16,17]

y = [11000,12000,15000,19000,22000,32000,38000,40000,50000,52000]


slope, intercept, r, p, std_err = stats.linregress(x, y)


def myfunc(x):
  return slope * x + intercept


mymodel = list(map(myfunc, x))


plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()


#future prediction at 20 and 26 experience
def myfunc(x):
  return slope * x + intercept


speed = myfunc(20)
print(speed)
speed = myfunc(25)
print(speed)
```

OUTPUT :-




```
55925.91414944356
68700.15898251193
```

PROGRAM NO 1O OUTPUT

12. Implement K-means Clustering in python.

```
import matplotlib.pyplot as plt

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

plt.scatter(x, y)
plt.show()
from sklearn.cluster import KMeans

    data = list(zip(x, y))
    inertias = []

    for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

    plt.plot(range(1,11), inertias, marker='o')
    plt.title('Elbow method')
    plt.xlabel('Number of clusters')
    plt.ylabel('Inertia')
    plt.show()
```

PROGRAM 12 OUT PUT

OUTPUT :-