2.Write a program to implement pruning of decision tree
CODE:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
data = pd.read_csv('train.csv')
#print(data)
data = data.loc[:,('Survived','Pclass','Sex','Age','SibSp','Parch','Fare')]
data.dropna(inplace=True)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data.Sex = le.fit_transform(data.Sex)
x = data.iloc[:,1:]   # Second column until the last column
y = data.iloc[:,0]    # First column (Survived) is our target
from sklearn.model_selection import train_test_split
#this function randomly split the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.3,random_state=42)
dtree = DecisionTreeClassifier()
dtree.fit(x_train, y_train)  #train parameters: features and target
pred = dtree.predict(x_test)  #parameter: new data to predict
from sklearn.metrics import accuracy_score
accuracy_score(y_test, pred)
```

OUTPUT:

```
0.7441860465116279
```

```
dtree = DecisionTreeClassifier(criterion='gini')
dtree.fit(x_train, y_train)
pred = dtree.predict(x_test)
print('Criterion=gini', accuracy_score(y_test, pred))
dtree = DecisionTreeClassifier(criterion='entropy')
dtree.fit(x_train, y_train)
pred = dtree.predict(x_test)
print('Criterion=entropy', accuracy_score(y_test, pred))
```

OUTPUT:

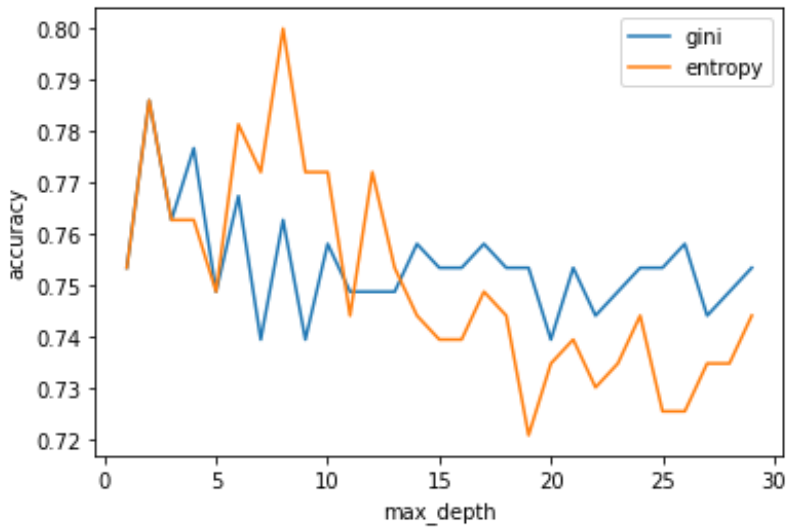
```
Criterion=gini 0.7534883720930232
Criterion=entropy 0.7255813953488373
```

```
max_depth = []
acc_gini = []
acc_entropy = []
for i in range(1,30):
```

```
    dtree = DecisionTreeClassifier(criterion='gini', max_depth=i)
    dtree.fit(x_train, y_train)
    pred = dtree.predict(x_test)
    acc_gini.append(accuracy_score(y_test, pred))
     ####
    dtree = DecisionTreeClassifier(criterion='entropy', max_depth=i)
    dtree.fit(x_train, y_train)
    pred = dtree.predict(x_test)
    acc_entropy.append(accuracy_score(y_test, pred))
     ####
    max_depth.append(i)
    d = pd.DataFrame({'acc_gini':pd.Series(acc_gini),
     'acc_entropy':pd.Series(acc_entropy),
     'max_depth':pd.Series(max_depth)})
# visualizing changes in parameters
plt.plot('max_depth','acc_gini', data=d, label='gini')
plt.plot('max_depth','acc_entropy', data=d, label='entropy')
plt.xlabel('max_depth')
plt.ylabel('accuracy')
plt.legend()
```

OUTPUT:

<matplotlib.legend.Legend at 0x1dd563479a0>




```
dtree = DecisionTreeClassifier(criterion='entropy', max_depth=8)
dtree.fit(x_train, y_train)
pred = dtree.predict(x_test)
accuracy_score(y_test, pred)
```

OUTPUT: 0.8

3. Write a program to implement an Artificial Neural Network by implementing the BackPropogation algorithm and test the same using appropriate dataset

CODE:

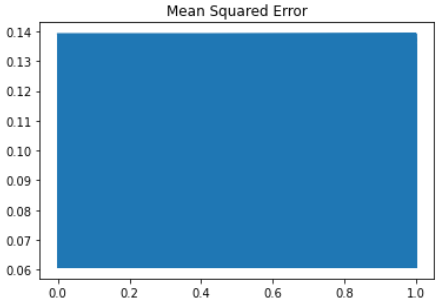
```
# Import Libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
# Load dataset
data = load_iris()
#print(data)
# Get features and target
X=data.data
y=data.target
# Get dummy variable
y = pd.get_dummies(y).values

y[:3]
#Split data into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20, random_state=4)
# Initialize variables
learning_rate = 0.1
iterations = 5000
N = y_train.size

# number of input features
input_size = 4

# number of hidden layers neurons
hidden_size = 2

# number of neurons at the output layer
output_size = 3

results = pd.DataFrame(columns=["mse", "accuracy"])
# Initialize weights
np.random.seed(10)

# initializing weight for the hidden layer
W1 = np.random.normal(scale=0.5, size=(input_size, hidden_size))
```

```
# initializing weight for the output layer
W2 = np.random.normal(scale=0.5, size=(hidden_size , output_size))

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def mean_squared_error(y_pred, y_true):
    return ((y_pred - y_true)**2).sum() / (2*y_pred.size)

def accuracy(y_pred, y_true):
    acc = y_pred.argmax(axis=1) == y_true.argmax(axis=1)
    return acc.mean()
for itr in range(iterations):

    # feedforward propagation
    # on hidden layer
    Z1 = np.dot(X_train, W1)
    A1 = sigmoid(Z1)

    # on output layer
    Z2 = np.dot(A1, W2)
    A2 = sigmoid(Z2)


    # Calculating error
    mse = mean_squared_error(A2, y_train)
    acc = accuracy(A2, y_train)
    #outputxlsx = pd.concat([outputxlsx, pd.DataFrame.from_records(df)])
    df_new_row = pd.DataFrame({"mse":mse, "accuracy":acc},index=range(2))
    results = pd.concat([results, df_new_row])
    #df = pd.DataFrame.from_records()
    #results=results.append({"mse":mse, "accuracy":acc},ignore_index=True )

    # backpropagation
    E1 = A2 - y_train
    dW1 = E1 * A2 * (1 - A2)

    E2 = np.dot(dW1, W2.T)
    dW2 = E2 * A1 * (1 - A1)


    # weight updates
    W2_update = np.dot(A1.T, dW1) / N
    W1_update = np.dot(X_train.T, dW2) / N
```

```
    W2 = W2 - learning_rate * W2_update
    W1 = W1 - learning_rate * W1_update

  results.mse.plot(title="Mean Squared Error")
```

OUTPUT

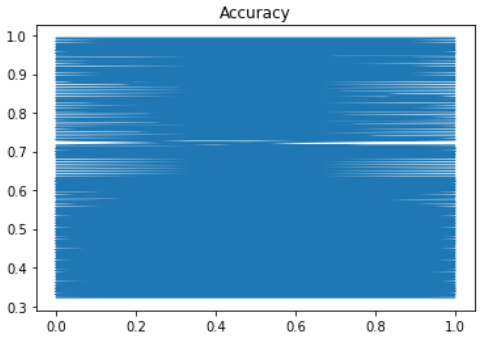
<AxesSubplot:title={'center':'Mean Squared Error'}>




```
results.accuracy.plot(title="Accuracy")
```

OUTPUT

<AxesSubplot:title={'center':'Accuracy'}>




```
# feedforward
Z1 = np.dot(X_test, W1)
A1 = sigmoid(Z1)
Z2 = np.dot(A1, W2)
A2 = sigmoid(Z2)
acc = accuracy(A2, y_test)
print("Accuracy: {}".format(acc))
```

OUTPUT

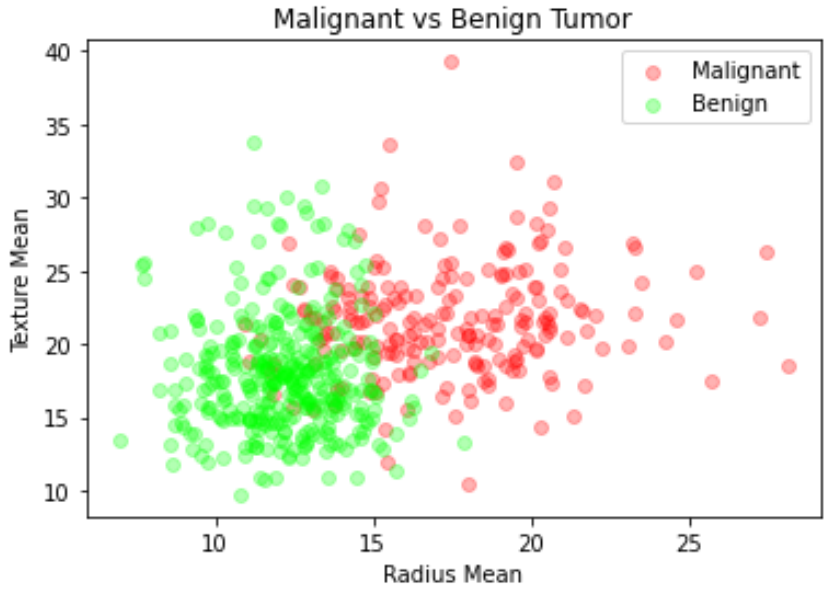
Accuracy: 0.8

4. Write a program to implement naïve Bayesian classifier and compute the accuracy, precision and recall of the classifier considering appropriate data set

CODE:

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

dataset = pd.read_csv("cancer.csv")
dataset.head()
dataset.info()

dataset = dataset.drop(["id"], axis = 1)
dataset = dataset.drop(["Unnamed: 32"], axis = 1)
M = dataset[dataset.diagnosis == "M"]
B = dataset[dataset.diagnosis == "B"]

plt.title("Malignant vs Benign Tumor")
plt.xlabel("Radius Mean")
plt.ylabel("Texture Mean")
plt.scatter(M.radius_mean, M.texture_mean, color = "red", label = "Malignant", alpha = 0.3)
plt.scatter(B.radius_mean, B.texture_mean, color = "lime", label = "Benign", alpha = 0.3)
plt.legend()
plt.show()
```

Output:




```
for i in dataset.diagnosis:
    if i=="M":
        i=1
    else:
        i=0
```

```
y = dataset.diagnosis.values
x = dataset.drop(["diagnosis"], axis = 1)

# Normalization:
x=(x-np.min(x))/(np.max(x)-np.min(x))

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 42)

from sklearn.naive_bayes import GaussianNB
nb=GaussianNB()
nb.fit(x_train,y_train)

print ("Naive Bayes score: ",nb.score(x_test, y_test))
```

Output:

Naive Bayes score: 0.935672514619883

5.Implement K-nearest Neighbour algorithm to classify any given dataset

KNN Regressor:

Code:

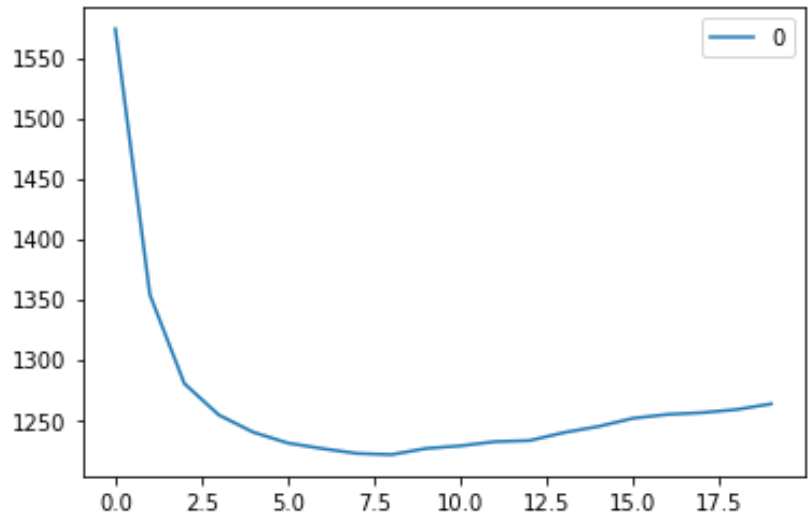
```
import pandas as pd
df = pd.read_csv('train_reg.csv')
df.head()
df.isnull().sum()
#missing values in Item_weight and Outlet_size needs to be imputed
mean = df['Item_Weight'].mean() #imputing item_weight with mean
df['Item_Weight'].fillna(mean, inplace =True)

mode = df['Outlet_Size'].mode() #imputing outlet size with mode
df['Outlet_Size'].fillna(mode[0], inplace =True)
df.drop(['Item_Identifier', 'Outlet_Identifier'], axis=1, inplace=True)
df = pd.get_dummies(df)
from sklearn.model_selection import train_test_split
# x=df.iloc[:,:-1]
# y=df.iloc[:,-1]
train , test = train_test_split(df, test_size = 0.3)
#x_train,y_train,x_test,y_test=train_test_split(x,y,test_size=0.3)
x_train = train.drop('Item_Outlet_Sales', axis=1)
y_train = train['Item_Outlet_Sales']

x_test = test.drop('Item_Outlet_Sales', axis = 1)
y_test = test['Item_Outlet_Sales']

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))

x_train_scaled = scaler.fit_transform(x_train)
x_train = pd.DataFrame(x_train_scaled)

x_test_scaled = scaler.fit_transform(x_test)
x_test = pd.DataFrame(x_test_scaled)

#import required packages
from sklearn import neighbors
from sklearn.metrics import mean_squared_error
from math import sqrt
import matplotlib.pyplot as plt
%matplotlib inline
```

```
rmse_val = [] #to store rmse values for different k
for K in range(20):
    K = K+1
    model = neighbors.KNeighborsRegressor(n_neighbors = K)

    model.fit(x_train, y_train)  #fit the model
    pred=model.predict(x_test) #make prediction on test set
    error = sqrt(mean_squared_error(y_test,pred)) #calculate rmse
    rmse_val.append(error) #store rmse values
    print('RMSE value for k= ' , K , 'is:', error)
```

Output:

```
RMSE value for k=  1 is: 1573.7659725611186
RMSE value for k=  2 is: 1353.6406287340028
RMSE value for k=  3 is: 1279.8567034306532
RMSE value for k=  4 is: 1253.8207188581016
RMSE value for k=  5 is: 1239.5111380734263
RMSE value for k=  6 is: 1230.5880129930883
RMSE value for k=  7 is: 1225.9635162345924
RMSE value for k=  8 is: 1221.9728569751333
RMSE value for k=  9 is: 1221.01213558666
RMSE value for k=  10 is: 1226.0948955842146
RMSE value for k=  11 is: 1228.2958108461166
RMSE value for k=  12 is: 1231.746354212223
RMSE value for k=  13 is: 1232.639801212764
RMSE value for k=  14 is: 1239.3017890174883
RMSE value for k=  15 is: 1244.350354779207
RMSE value for k=  16 is: 1251.1394261776782
RMSE value for k=  17 is: 1254.3289156530218
RMSE value for k=  18 is: 1255.7529906734492
RMSE value for k=  19 is: 1258.3755898697752
RMSE value for k=  20 is: 1263.0568716412893
```

```
#plotting the rmse values against k values
curve = pd.DataFrame(rmse_val) #elbow curve
curve.plot()
```




```
#reading test and submission files
test = pd.read_csv('test_reg.csv')
submission = pd.read_csv('sample_submission_reg.csv')
submission['Item_Identifier'] = test['Item_Identifier']
submission['Outlet_Identifier'] = test['Outlet_Identifier']

#preprocessing test dataset
test.drop(['Item_Identifier', 'Outlet_Identifier'], axis=1, inplace=True)
test['Item_Weight'].fillna(mean, inplace =True)
test = pd.get_dummies(test)
test_scaled = scaler.fit_transform(test)
test = pd.DataFrame(test_scaled)

#predicting on the test set and creating submission file
predict = model.predict(test)
print(predict)
submission['Item_Outlet_Sales'] = predict
submission.to_csv('submit_file.csv',index=False)
```

Output:

```
[2086.9501  1534.80216  573.85302 ... 2065.17844 3055.42278 1448.01513]
```

```
from sklearn.model_selection import GridSearchCV
params = {'n_neighbors':[2,3,4,5,6,7,8,9]}

knn = neighbors.KNeighborsRegressor()

model = GridSearchCV(knn, params, cv=5)
model.fit(x_train,y_train)
model.best_params_
```

Output:
{'n_neighbors': 8}

KNN Classifier:

Code:

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

dataset=sns.load_dataset("iris")
X = dataset.iloc[:, :-1].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)

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

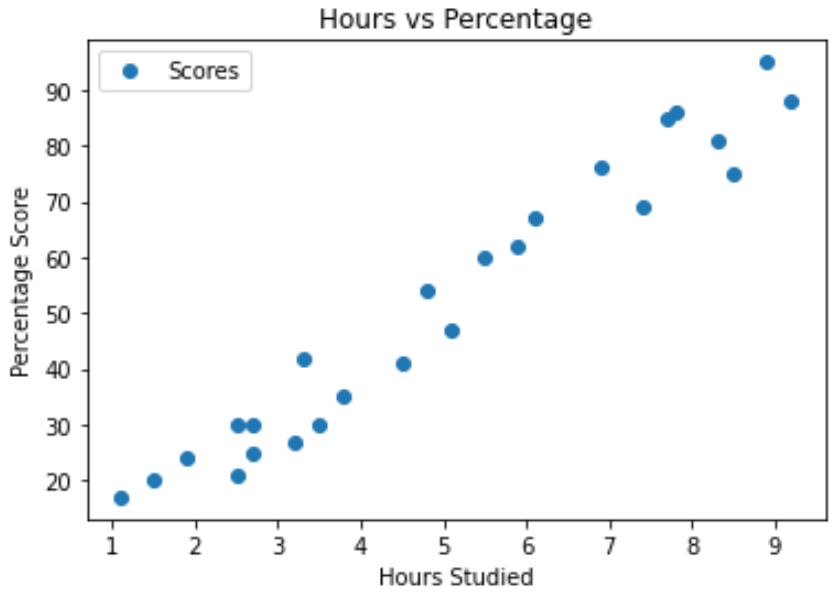
Output:

```
[[10  0  0]
 [ 0  9  0]
 [ 0  1 10]]
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.90      1.00      0.95         9
   virginica       1.00      0.91      0.95        11

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30

0.9666666666666667
```

6. Implement Regression Algorithm in order to fit data points.

Code:

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

dataset = pd.read_csv('student_scores.csv')
dataset.head()
dataset.shape
dataset.describe()

dataset.plot(x='Hours', y='Scores', style='o')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.show()
```




```
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)
```

```
y_pred = regressor.predict(X_test)

df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

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:

```
Mean Absolute Error: 4.183859899002975
Mean Squared Error: 21.598769307217406
Root Mean Squared Error: 4.647447612100367
```

7. Implement Radial Basis Algorithm to classify the given dataset

Code:

```
import math
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import numpy as numpy

Data= pd.read_table("bank-full.csv", sep= None, engine= "python")
print(Data)
cols= ["age","balance","day","duration","campaign","pdays","previous"]
data_encode= Data.drop(cols, axis= 1)
data_encode= data_encode.apply(LabelEncoder().fit_transform)
data_rest= Data[cols]
Data= pd.concat([data_rest,data_encode], axis= 1)

data_train, data_test= train_test_split(Data, test_size= 0.33, random_state= 4)
X_train= data_train.drop("Target", axis= 1)
Y_train= data_train["Target"]
X_test= data_test.drop("Target", axis=1)
Y_test= data_test["Target"]

scaler= StandardScaler()
scaler.fit(X_train)
X_train= scaler.transform(X_train)
X_test= scaler.transform(X_test)

K_cent= 8
km= KMeans(n_clusters= K_cent, max_iter= 100)
km.fit(X_train)
cent= km.cluster_centers_

max=0
for i in range(K_cent):
        for j in range(K_cent):
                d= numpy.linalg.norm(cent[i]-cent[j])
                if(d> max):
                        max= d
d= max
```

```
sigma= d/math.sqrt(2*K_cent)
row= X_train.shape[0]
column= K_cent
G= numpy.empty((row,column), dtype= float)
for i in range(row):
   for j in range(column):
      dist= numpy.linalg.norm(X_train[i]-cent[j])
      G[i][j]= math.exp(-math.pow(dist,2)/math.pow(2*sigma,2))

GTG= numpy.dot(G.T,G)
GTG_inv= numpy.linalg.inv(GTG)
fac= numpy.dot(GTG_inv,G.T)
W= numpy.dot(fac,Y_train)

row= X_test.shape[0]
column= K_cent
G_test= numpy.empty((row,column), dtype= float)
for i in range(row):
        for j in range(column):
                dist= numpy.linalg.norm(X_test[i]-cent[j])
                G_test[i][j]= math.exp(-math.pow(dist,2)/math.pow(2*sigma,2))

prediction= numpy.dot(G_test,W)
prediction= 0.5*(numpy.sign(prediction-0.5)+1)

score= accuracy_score(prediction,Y_test)
print (score.mean())
```

Output:

```
0.8876675603217158
```