**4. Implement Linear Regression using Python Script and identify explanatory variables.**

import pandas as pd

import matplotlib.pyplot as plt

exp=[3,8,9,13,3,6,11,21,1,16]

sal=[30,57,64,72,36,43,59,90,20,83]

df=pd.DataFrame(list(zip(exp,sal)),columns=['year of exp','salary'])

df

df.to\_csv("Rsqr.csv")

plt.scatter(df['year of exp'],df['salary'],color='red',marker='+')

from sklearn import linear\_model

reg=linear\_model.LinearRegression()

reg.fit(df[['year of exp']],df[['salary']])

print(reg.coef\_)

print(reg.intercept\_)

df['y-pred']=reg.predict(df[['year of exp']])

df

plt.xlabel('year of exp',fontsize=10)

plt.ylabel('salary',fontsize=10)

plt.scatter(df[['year of exp']],df[['salary']],color='red',marker='+')

plt.plot(df[['year of exp']],reg.predict(df[['year of exp']]),color='blue')

data=pd.read\_csv("salary.csv")

data

data['salary']=reg.predict(data)

data

**5. Write a program to demonstrate the working of the decision tree.**

import sklearn

import pandas as pd

df=pd.read\_csv("weather-decisiontree.csv")

df

from sklearn.preprocessing import LabelEncoder

df['Outlook']=le\_Outlook.fit\_transform(df['Outlook'])

df['Temperature']=le\_Temperature.fit\_transform(df['Temperature'])

df['Humidity']=le\_Humidity.fit\_transform(df['Humidity'])

df['Wind']=le\_Wind.fit\_transform(df['Wind'])

df['play']=le\_play.fit\_transform(df['play'])

df

x=df.drop(['Day','play'],axis='columns')

y=df['play']

from sklearn.tree import DecisionTreeClassifier

from sklearn.tree import plot\_tree

import matplotlib.pyplot as plt

model=DecisionTreeClassifier()#criterion="entropy",max\_depth=4)

model.fit(x,y)

model.predict([[2,1,0,1]])

plt.figure(figsize=(25,10))

a = plot\_tree(model, filled=True)

**6. Implement clustering technique for a given data set in python.**

**a. Agglomerative clustering**

import pandas as pd

import matplotlib.pyplot as plt

p=['p1','p2','p3','p4','p5','p6']

l1=[0.40,0.22,0.35,0.26,0.08,0.45]

l2=[0.53,0.38,0.32,0.19,0.41,0.30]

df=pd.DataFrame(list(zip(p,l1,l2)),columns=['point','X','Y'])

df=df.set\_index('point')

df

from scipy.spatial.distance import squareform, pdist

dist = pd.DataFrame(squareform(pdist(df[['X', 'Y']]), 'euclidean'), columns=df.index.values, index=df.index.values)

dist

import scipy.cluster.hierarchy as sch

plt.figure(figsize=(10,7))

dendogram=sch.dendrogram(sch.linkage(df, method="single"))

**b. K-means clustering**

import pandas as pd

import matplotlib.pyplot as plt

l1=[185,170,168,179,182,188,180,180,183,180,180,177]

l2=[72,56,60,68,72,77,71,70,84,88,67,76]

data=pd.DataFrame(list(zip(l1,l2)),columns=['Height','Weight'])

data

plt.scatter(data['Height'],data['Weight'])

from sklearn.cluster import KMeans

model=KMeans(n\_clusters=3)

pred=model.fit\_predict(data[['Height','Weight']])

data['class']=pred

data

model.cluster\_centers\_

df1=data[data['class']==1]

df2=data[data['class']==0]

df3=data[data['class']==2]

plt.scatter(df1['Height'],df1['Weight'],color='red')

plt.scatter(df2['Height'],df2['Weight'],color='blue')

plt.scatter(df3['Height'],df3['Weight'],color='green')

model.predict([[171,57]])

**7. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.**

from sklearn.naive\_bayes import GaussianNB

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

from sklearn.metrics import confusion\_matrix

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns; sns.set()

training = pd.read\_csv("Iris.csv")

test.head(20)

plt.scatter(training['SepalLengthCm'],training['PetalLengthCm'])

plt.scatter(training['PetalWidthCm'],training['PetalLengthCm'])

# Create the X, Y, Training and Test

x = training.drop('Species', axis=1)

y = training.loc[:, 'Species']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.2)

# Init the Gaussian Classifier

model = GaussianNB()

# Train the model

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

# Predict Output

pred = model.predict(X\_test)

df = pd.DataFrame({'RealValues':y\_test,'PredictedValues':pred})

print(df)

from sklearn.metrics import accuracy\_score

cm = confusion\_matrix(y\_test,pred)

print(accuracy\_score(y\_test,pred))

print(cm)

**8. Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.**

**import numpy as np**

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0) #maximum of X array longitudinally

y = y/100

#Sigmoid Function

def sigmoid (x):

return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function

def derivatives\_sigmoid(x):

return x \* (1 - x)

#Variable initialization

epoch=5 #Setting training iterations

lr=0.1 #Setting learning rate

inputlayer\_neurons = 2 #number of features in data set

hiddenlayer\_neurons = 3 #number of hidden layers neurons

output\_neurons = 1 #number of neurons at output layer

#weight and bias initialization

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

#draws a random range of numbers uniformly of dim x\*y

for i in range(epoch):

#Forward Propogation

hinp1=np.dot(X,wh)

hinp=hinp1 + bh

hlayer\_act = sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout)

outinp= outinp1+bout

output = sigmoid(outinp)

#Backpropagation

EO = y-output

outgrad = derivatives\_sigmoid(output)

d\_output = EO \* outgrad

EH = d\_output.dot(wout.T)

hiddengrad = derivatives\_sigmoid(hlayer\_act)#how much hidden layer wts contributed to error

d\_hiddenlayer = EH \* hiddengrad

wout += hlayer\_act.T.dot(d\_output) \*lr # dotproduct of nextlayererror and currentlayerop

wh += X.T.dot(d\_hiddenlayer) \*lr

print ("-----------Epoch-", i+1, "Starts----------")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

print ("-----------Epoch-", i+1, "Ends----------\n")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)