**EX NO: 1A**

**DATE:**

**UNIFORMED SEARCH TECHNIQUES -BFS**

**AIM:**

To write and implement the python program for breadth first search using uniformed searching techniques.

**PROGRAM:**

def bfs (G, S):

visited = []

queue = []

queue. append (S)

visited. append (S)

bfstree = []

while queue:

t = queue.pop (0)

bfstree. append (t)

for n in G[t]:

if n not in visited:

visited. append(n)

queue. Append(n)

return bfstree

graph = {}

n=int (input (“Enter number of vertices in graph:”))

for i in range (n):

neighbour = []

node = input (“Enter name of node: “)

neighbour = input (“Enter name of neighbour node “). split ()

graph [node]=neighbour

s=input (“Enter starting node name:”)

print (graph)

print (“BFS traversal”)

print (bfs (graph, S))

**OUTPUT:**

Enter number of vertices in graph: 6

Enter name of node: A

Enter name of neighbour node B C

Enter name of node: B

Enter name of neighbour node A D F

Enter name of node: C

Enter name of neighbour node A D

Enter name of node: D

Enter name of neighbour node B C E F

Enter name of node: E

Enter name of neighbour node D

Enter name of node: F

Enter name of neighbour node B D

Enter starting node name: A

{'A': ['B', 'C'], 'B': ['A', 'D', 'F'], 'C ': ['A', 'D'], 'D': ['B', 'C', 'E', 'F'], 'E': ['D'], 'F': ['B', 'D']}BFS traversal

[‘A’,’B’,’C’,’D’,’F’,’E’]

**RESULT:**

Thus, the program to demonstrate breadth first search was written and executed.

**EX NO: 1B**

**DATE:**

**UNIFORMED SEARCH TECHNIQUES – DFS**

**AIM:**

To write and implement the python program for breadth first search using uniformed searching techniques.

**PROGRAM:**

def dfs (G, S, dfstree):

for n in G[S]:

if n not in dfstree:

dfstree[n]=S

dfs (G, n, dfstree)

return dfstree

graph= {}

n=int (input (“enter number of vertices in graph: “))

for i in range (n):

neighbour= []

node=input (“enter name of node: “)

neighbour=input (“enter name of neighbour node:”). split ()

graph [node]=neighbour

s=input (“enter starting node name:”)

print (“DFS traversal”)

dfstree= {S: None}

print (dfs (graph, S, dfstree))

**OUTPUT:**

enter number of vertices in graph: 6

enter name of node: A

enter name of neighbour node:B C

enter name of node: B

enter name of neighbour node:A D F

enter name of node: C

enter name of neighbour node:A D

enter name of node: D

enter name of neighbour node:B C E F

enter name of node: E

enter name of neighbour node:D

enter name of node: F

enter name of neighbour node:B D

enter starting node name:A

DFS traversal

{'A': None, 'B': 'A', 'D': 'B', 'C': 'D', 'E': 'D', 'F': 'D'}

**RESULT:**

Thus, the program to demonstrate depth first search was written and executed.

**EX NO:2**

**DATE:**

**A\* ALGORITHM**

**AIM:**

To find the most cost-effective path to reach the final state from initial state using A\* algorithm.

**PROGRAM:**

def aStarAlgo (start\_node, stop\_node)

open\_list = set(start\_node)

closed\_list = set ()

g = {}

parents = {}

g[start\_node] = 0

parents[start\_node] = start\_node

while len (open\_list) > 0:

n = None

for v in open\_list:

if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):

n = v

if n == stop\_node or Graph\_nodes[n] == None:

pass

else:

for (m, weight) in get\_neighbors(n):

if m not in open\_list and m not in closed\_list:

open\_list.add(m)

parents[m] = n

g[m] = g[n] + weight

else:

if g[m] > g[n] + weight:

g[m] = g[n] + weight

parents[m] = n

if m in closed\_list:

closed\_list.remove(m)

open\_list.add(m)

if n == None

print (“Path does not exist!”)

return None

if n == stop\_node:

path = []

while parents[n]! = n:

path.append(n)

n = parents[n]

path.append(start\_node)

path.reverse()

print (“Path found: {}”. format(path))

return path

open\_list.remove(n)

closed\_list.add(n)

print (“Path does not exist!”)

return None

def get\_neighbors(v):

if v in Graph\_nodes:

return Graph\_nodes[v]

else:

return None

def heuristic(n):

H\_dist = {‘A’: 11, ‘B’: 6, ‘C’: 5, ‘D’: 7, ‘E’: 3, ‘F’: 6, ‘G’: 5, ‘H’: 3, ‘I’: 1, ‘J’: 0}

return H\_dist[n]

Graph\_nodes = {‘A’: [(‘B’, 6), (‘F’, 3)], ‘B’: [(‘A’, 6), (‘C’, 3), (‘D’, 2)], ‘C’: [(‘B’, 3), (‘D’, 1), (‘E’, 5)], ‘D’: [(‘B’, 2), (‘C’, 1), (‘E’, 8)], ‘E’: [(‘C’, 5), (‘D’, 8), (‘I’, 5), (‘J’, 5)],’F’: [(‘A’, 3), (‘G’, 1), (‘H’, 7)], ‘G’: [(‘F’, 1), (‘I’, 3)], ‘H’: [(‘F’, 7), (‘I’, 2)],’I’: [(‘E’,5), (‘G’, 3), (‘H’, 2), (‘J’, 3)]}

S=str (input (“Enter the starting node :”))

G=str (input (“Enter the goal node :”))

aStarAlgo (S, G)

**OUTPUT:**

Enter the starting node : A

Enter the goal node : J

[‘A’, ‘F’,’G’, ‘I’, ‘J’]

**RESULT:**

Thus, the program to demonstrate A\* algorithm was written and executed.

**EX NO:3A**

**DATE:**

**IMPLEMENTATION OF LINEAR REGRESSION**

**AIM:**

To write a python program to implement the linear regression.

**PROGRAM:**

import numpy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

#number of observations/points

n= np.size(x)

# mean of x and y vector

m\_x = np.mean(x)

m\_y = np.mean(y)

# calculating cross-deviation and deviation about x

SS\_xy = np.sum(y\*x) – n\*m\_y\*m\_x

SS\_xx = np.sum(x\*x) – n\*m\_x\*m\_x

# calculating regression coefficients

B\_1 = SS\_xy / SS\_xx

B\_0 = m\_y – b\_1\*m\_x

return (b\_0, b\_1)

def plot\_regression\_line(x, y, b):

# plotting the actual points as scatter plot

plt.scatter(x, y, color = “m”,Marker = “o”, s = 30)

y\_pred = b[0] + b[1]\*x

plt.plot(x, y\_pred, color = “g”)

plt.xlabel(‘x’)

plt.ylabel(‘y’)

plt.show()

def main ():

# observations / data

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

# estimating coefficients

b = estimate\_coef(x, y)

print (“Estimated coefficients:\nb\_0 = {} \\nb\_1 = {}”.format(b[0], b[1]))

# plotting regression line

plot\_regression\_line(x, y, b)

if \_\_name\_\_ == “\_\_main\_\_”:

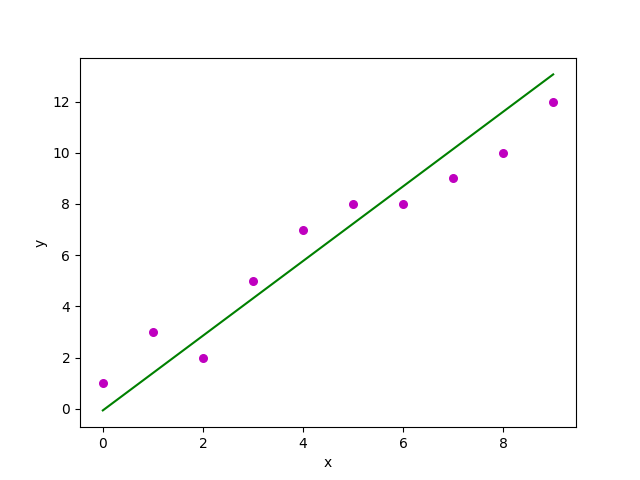
main ()

**OUTPUT:**

Estimated coefficients:

b\_0 = -0.0586206896552

b\_1 = 1.45747126437



**RESULT:**

Thus, the program to implement the linear regression was written and executed.

**EX NO: 9B**

**DATE:**

**MULTIPLE VARIABLE LINEAR REGRESSION**

**AIM:**

To write a python program to demonstrate multiple variable linear regression.

**PROGRAM:**

import numpy as np

import pandas as pd

import statistics

import math

from matplotlib import pyplot as plt

import statsmodels.formula.api as smf

import requests # Module to process http/https requests

remote\_url =” http: //54.243.252.9/engr-1330-webroot/8-Labs/Lab29/heart.data.csv

rget = requests.get(remote\_url, allow\_redirects=True)

open(‘heart.data.csv’,’wb’). write(rget.content);

heartattack = pd.read\_csv(‘heart.data.csv’)

data=heartattack.rename (columns={“biking”:”Bike”,”smoking”:”Smoke”,”heart.disease”:”Disease”})

print(data.head(3))

# Initialise and fit linear regression model using `statsmodels`

model = smf.ols(‘Disease ~ Bike + Smoke’, data=data)

model = model.fit()

print(model.summary())

dir(model)

# activate to find attributes

intercept = model.params[0]

slope = model.params[1]

rsquare = model.rsquared

RMSE = math.sqrt(model.mse\_total)

# Predict values

heartfail = model.predict()

titleline = ‘Disease Index versus Lifestyle Variables \n’ + ‘R squared = ‘ + str(round(Rsquare,3)) + ‘ \n RMSE = ‘ + str(round(RMSE,2))

# Plot regression against actual data

plt.figure(figsize=(12, 6))

plt.plot(data[‘Bike’], data[‘Disease’], ‘o’) # scatter plot showing actual data

plt.plot(data[‘Bike’], heartfail, marker = ‘s’ ,color =’r’, linewidth=0) # regression line

plt.xlabel(‘Biking (miles/week)’)

plt.ylabel(‘Disease Index (Admissions/100,000 as per MMWR)’)

plt.legend([‘Observations’,’Model Prediction’])

plt.title(titleline)

plt.show()

titleline = ‘Disease Index versus Lifestyle Variables \n’ + ‘R squared = ‘ + str(round(Rsquare,3)) + ‘ \n RMSE = ‘ + str(round(RMSE,2))

# Plot regression against actual data – What do we see?

plt.figure(figsize=(12, 6))

plt.plot(data[‘Smoke’], data[‘Disease’], ‘o’) # scatter plot showing actual data

plt.plot(data[‘Smoke’], heartfail, marker = ‘s’ ,color =’r’, linewidth=0) # regression line

plt.xlabel(‘Smoking (packs/week)’)

plt.ylabel(‘Disease Index (Admissions/100,000 as per MMWR)’)

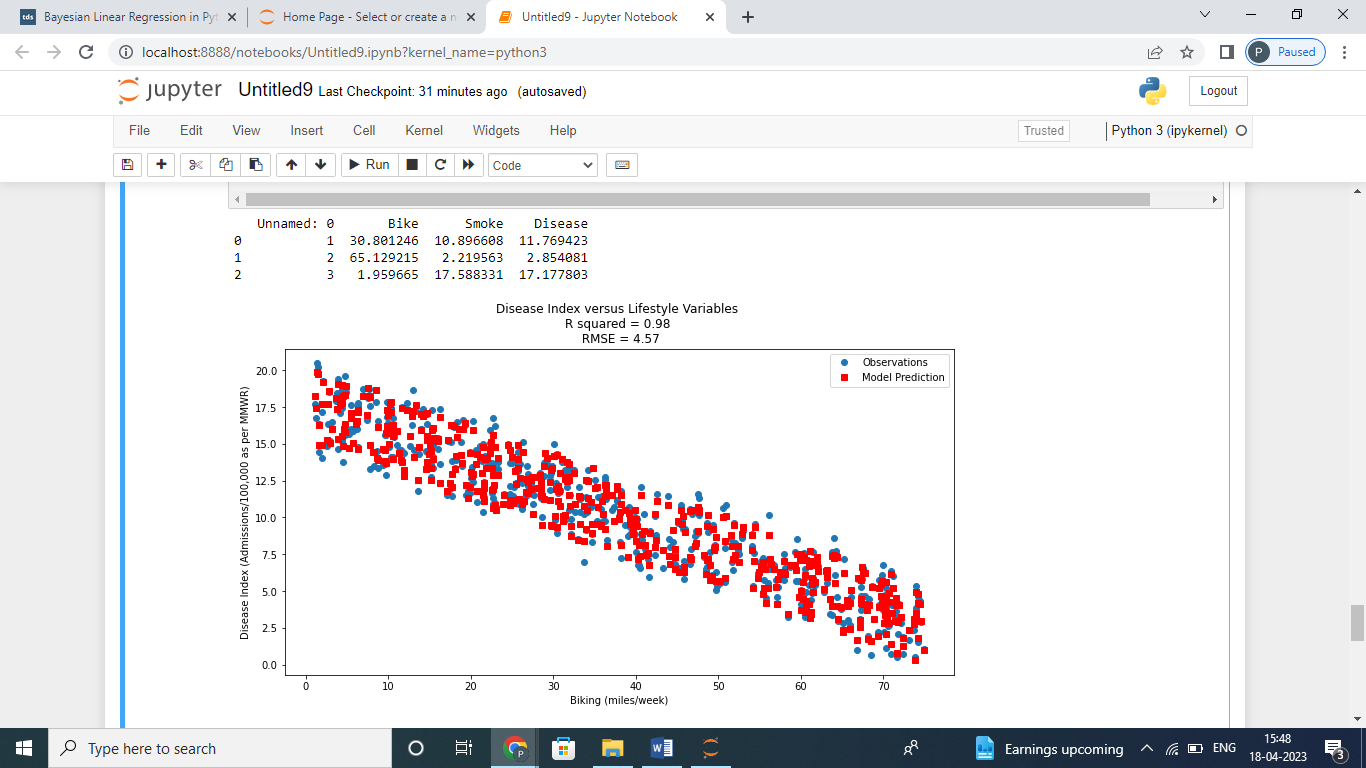
plt.legend([‘Observations’,’Model Prediction’])

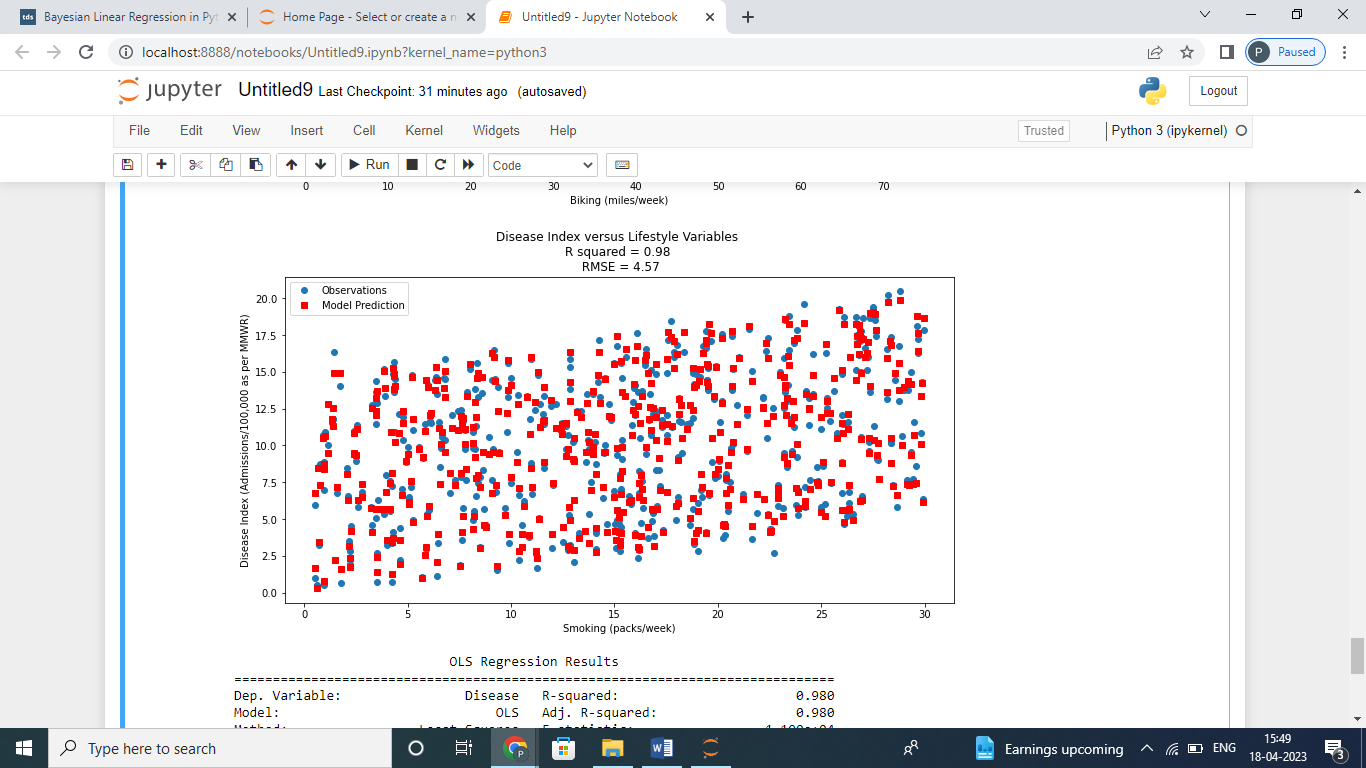
plt.title(titleline)

plt.show()

print(model.summary())

**OUTPUT:**





OLS Regression Results

==============================================================================

Dep. Variable: Disease R-squared: 0.980

Model: OLS Adj. R-squared: 0.980

Method: Least Squares F-statistic: 1.190e+04

Date: Tue, 18 Apr 2023 Prob (F-statistic): 0.00

Time: 15:39:07 Log-Likelihood: -493.68

No. Observations: 498 AIC: 993.4

Df Residuals: 495 BIC: 1006.

Df Model: 2

Covariance Type: nonrobust

==============================================================================

coef std err t P>|t| [0.025 0.975]

------------------------------------------------------------------------------

Intercept 14.9847 0.080 186.988 0.000 14.827 15.142

Bike -0.2001 0.001 -146.525 0.000 -0.203 -0.197

Smoke 0.1783 0.004 50.387 0.000 0.171 0.185

==============================================================================

Omnibus: 2.794 Durbin-Watson: 1.917

Prob(Omnibus): 0.247 Jarque-Bera (JB): 2.582

Skew: -0.141 Prob(JB): 0.275

Kurtosis: 3.211 Cond. No. 125.

==============================================================================

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

**RESULT:**

Thus, the program to demonstrate multiple variable linear regression

was written and executed.

**EX NO: 4**

**DATE:**

**NAÏVE BAYES MODEL**

**AIM:**

To write a python program to demonstrate naïve bayes model using a given dataset.

PROGRAM:

importing library

import math

import random

import csv

# the categorical class names are changed to numeric data

# eg: yes and no encoded to 1 and 0

def encode\_class(mydata):

classes = []

for i in range (len (mydata)):

if mydata[i] [-1] not in classes:

Classes.append(mydata[i] [-1])

for i in range(len(classes)):

for j in range(len(mydata)):

if mydata[j][-1] == classes[i]:

mydata[j][-1] = i

return mydata

# Splitting the data

def splitting (mydata, ratio):

train\_num = int(len(mydata) \* ratio)

train = []

# initially testset will have all the dataset

test = list(mydata)

while len(train) < train\_num:

# index generated randomly from range 0

# to length of testset

index = random.randrange(len(test))

# from testset, pop data rows and put it in train

train.append(test.pop(index))

return train, test

# Group the data rows under each class yes or

# no in dictionary eg: dict[yes] and dict[no]

def groupUnderClass(mydata):

dict = {}

for i in range(len(mydata)):

if (mydata[i] [-1] not in dict):

dict[mydata[i] [-1]] = []

dict[mydata[i] [-1]]. append(mydata[i])

return dict

# Calculating Mean

def mean(numbers):

return sum(numbers) / float(len(numbers))

# Calculating Standard Deviation

def std\_dev(numbers):

avg = mean(numbers)

variance = sum([pow(x – avg, 2) for x in numbers]) / float(len(numbers) – 1)

return math.sqrt(variance)

def MeanAndStdDev(mydata):

info = [(mean(attribute), std\_dev(attribute)) for attribute in zip(\*mydata)]

# eg: list = [ [a, b, c], [m, n, o], [x, y, z]]

# here mean of 1st attribute =(a + m+x), mean of 2nd attribute = (b + n+y)/3

# delete summaries of last class

del info[-1]

return info

# find Mean and Standard Deviation under each class

def MeanAndStdDevForClass(mydata):

info = {}

dict = groupUnderClass(mydata)

for classValue, instances in dict.items():

info[classValue] = MeanAndStdDev(instances)

return info

# Calculate Gaussian Probability Density Function

def calculateGaussianProbability(x, mean, stdev):

expo = math.exp(-(math.pow(x – mean, 2) / (2 \* math.pow(stdev, 2))))

return (1 / (math.sqrt(2 \* math.pi) \* stdev)) \* expo

# Calculate Class Probabilities

def calculateClassProbabilities(info, test):

probabilities = {}

for classValue, classSummaries in info.items():

probabilities[classValue] = 1

for I in range(len(classSummaries)):

mean, std\_dev = classSummaries[i]

X = test[i]

probabilities[classValue] \*= calculateGaussianProbability(x, mean, std\_dev)

return probabilities

# Make prediction – highest probability is the prediction

def predict (info, test):

probabilities = calculateClassProbabilities(info, test)

bestLabel, bestProb = None, -1

for classValue, probability in probabilities.items():

if bestLabel is None or probability > bestProb:

bestProb = probability

bestLabel = classValue

return bestLabel

# returns predictions for a set of examples

def getPredictions(info, test):

predictions = []

for i in range(len(test)):

result = predict (info, test[i])

predictions.append(result)

return predictions

# Accuracy score

def accuracy\_rate(test, predictions):

correct = 0

for i in range(len(test)):

if test[i][-1] == predictions[i]:

correct += 1

return (correct / float(len(test))) \* 100.0

# driver code

# add the data path in your system

filename = r’E: \user\MACHINE LEARNING\machine learning algos\Naive bayes\filedata.csv’

# load the file and store it in mydata list

mydata = csv.reader(open(filename, “rt”))

mydata = list(mydata)

mydata = encode\_class(mydata)

for i in range(len(mydata)):

mydata[i] = [float(x) for x in mydata[i]]

# split ratio = 0.7

# 70% of data is training data and 30% is test data used for testing

ratio = 0.7

train\_data, test\_data = splitting (mydata, ratio)

print (‘Total number of data: ‘, len(mydata))

print (‘Training data: ‘, len(train\_data))

print (“Test data: “, len(test\_data))

# prepare model

info = MeanAndStdDevForClass(train\_data)

# test model

predictions = getPredictions(info, test\_data)

accuracy = accuracy\_rate(test\_data, predictions)

print(“Accuracy of the model is: “, accuracy)

# Dataset: Indian\_Diabetes Dataset (pima-indians-diabetes.csv)

**OUTPUT:**

Total number of data: 200

Training data: 140

Test data: 60

Accuracy of the model is: 71.2376788

**RESULT:**

Thus, the program to implement a Naïve Bates Model was written and executed.

**EX NO: 5**

**DATE:**

**SUPPORT VECTOR MACHINE**

**AIM:**

To write a python program to demonstrate support vector machine by finding a decision boundary.

**PROGRAM:**

# Basic packages

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Sklearn modules & classes

from sklearn.linear\_model import Perceptron, LogisticRegression

from sklearn.svm import SVC

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

from sklearn.preprocessing import StandardScaler

from sklearn import datasets

from sklearn import metrics

# Load the data set; In this example, the breast cancer dataset is loaded.

bc = datasets.load\_breast\_cancer()

X = bc.data

Y = bc.target

 # Create training and test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1, stratify=y)

# feature scaling

sc = StandardScaler()

sc.fit(X\_train)

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

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

#Instantiate the Support Vector Classifier (SVC)

svc = SVC (C=1.0, random\_state=1, kernel=’linear’)

# Fit the model

svc.fit(X\_train\_std, y\_train)

#make the prediction

Y\_predict = svc.predict(X\_test\_std)

# Measure the performance

print (“Accuracy score %.3f” %metrics.accuracy\_score(y\_test, y\_predict))

**OUTPUT:**

Accuracy score 0.953

**RESULT:**

Thus, the program to demonstrate support vector machine by finding a decision boundary was written and executed.

**EX NO: 6**

**DATE:**

**BAYESIAN NETWORK**

**AIM:**

To write a python program to demonstrate Bayesian network.

**PROGRAM:**

import numpy as np

from pgmpy.models import BayesianModel

from pgmpy.factors.discrete import TabularCPD

# Define the model structure

model = BayesianModel([(‘Burglary’, ‘Alarm’), (‘Earthquake’, ‘Alarm’), (‘Alarm’, ‘JohnCalls’), (‘Alarm’, ‘MaryCalls’)])

# Define the conditional probability distributions

cpd\_burglary=TabularCPD(variable=’Burglary’,variable\_card=2,values=[[0.50],[0.50]])

cpd\_earthquake = TabularCPD(variable=’Earthquake’, variable\_card=2, values=[[0.998], [0.002]])

cpd\_alarm = TabularCPD(variable=’Alarm’, variable\_card=2, Evidence= [‘Burglary’, ‘Earthquake’], Evidence\_card= [2, 2], Values= [[0.999, 0.71, 0.06, 0.05],[0.001, 0.29, 0.94, 0.95]])

cpd\_john\_calls = TabularCPD(variable=’JohnCalls’, variable\_card=2,

Evidence=[‘Alarm’], evidence\_card=[2],

Values=[[0.95, 0.1], [0.05, 0.9]])

cpd\_mary\_calls = TabularCPD(variable=’MaryCalls’, variable\_card=2,

Evidence=[‘Alarm’], evidence\_card=[2],

Values=[[0.99, 0.3], [0.01, 0.7]])

# Add the conditional probability distributions to the model

model.add\_cpds(cpd\_burglary, cpd\_earthquake, cpd\_alarm, cpd\_john\_calls, cpd\_mary\_calls)

# Check if the model is valid

if model.check\_model():

print (“Model is valid”)

else:

print (“Model is not valid”)

print(alarm\_model.local\_independencies(‘Burglary’))

print(alarm\_model.edges())

print(alarm\_model.nodes())

**OUTPUT:**

Model is valid

(Burglary *|* Earthquake)

OutEdgeView([('Burglary', 'Alarm'), ('Alarm', 'JohnCalls'), ('Alarm', 'MaryCalls'), ('Earthquake', 'Alarm')])

NodeView(('Burglary', 'Alarm', 'Earthquake', 'JohnCalls', 'MaryCalls'))

**RESULT:**

Thus, the program to demonstrate Bayesian network was written and executed.

**EX NO: 7A**

**DATE:**

**DECISION TREE**

**AIM:**

To write a python program to demonstrate decision tree to classify a given dataset.

**PROGRAM:**

# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.preprocessing import StandardScaler

from matplotlib.colors import ListedColormap

#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.

X\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.25, random\_state=0)

#feature Scaling

st\_x= StandardScaler()

X\_train= st\_x.fit\_transform(x\_train)

X\_test= st\_x.transform(x\_test)

classifier= DecisionTreeClassifier(criterion=’entropy’, random\_state=0)

classifier.fit(x\_train, y\_train)

#Predicting the test set result

Y\_pred= classifier.predict(x\_test)

from sklearn.metrics import confusion\_matrix

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

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(‘Decision Tree Algorithm (Training set)’)

mtp.xlabel(‘Age’)

mtp.ylabel(‘Estimated Salary’)

mtp.legend()

mtp.show()

#Visualizing the test set result

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(‘Decision Tree Algorithm (Test set)’)

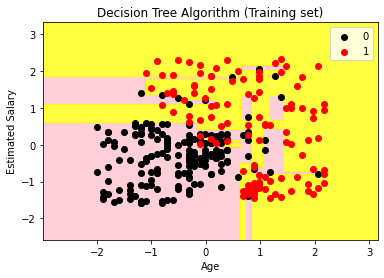
mtp.xlabel(‘Age’)

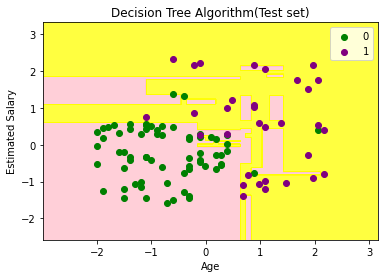
mtp.ylabel(‘Estimated Salary’)

mtp.legend()

mtp.show()

**OUTPUT:**





**RESULT:**

Thus, the program to demonstrate decision tree to classify a given dataset was written and executed.

**EX NO: 7B**

**DATE:**

**RANDOM FOREST**

**AIM:**

To write a python program to demonstrate a random forest.

**PROGRAM:**

# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

from matplotlib.colors import ListedColormap

from sklearn.preprocessing import StandardScaler

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

from sklearn.ensemble import RandomForestClassifier  from sklearn.metrics import confusion\_matrix

#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.

X\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.25, random\_state=0)

st\_x= StandardScaler()

X\_train= st\_x.fit\_transform(x\_train)

X\_test= st\_x.transform(x\_test)

#Fitting Decision Tree classifier to the training set

classifier= RandomForestClassifier (n\_estimators= 10, criterion =” entropy”)

classifier.fit(x\_train, y\_train)

#Predicting the test set result

Y\_pred= classifier.predict(x\_test)

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

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(‘Random Forest Algorithm (Training set)’)

mtp.xlabel(‘Age’)

mtp.ylabel(‘Estimated Salary’)

mtp.legend()

mtp.show()

#Visulaizing the test set result

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(‘Random Forest Algorithm(Test set)’)

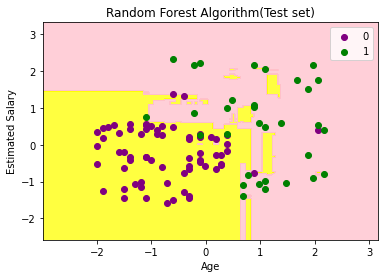
mtp.xlabel(‘Age’)

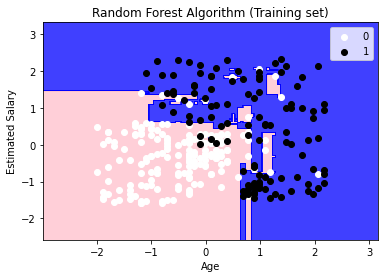
mtp.ylabel(‘Estimated Salary’)

mtp.legend()

mtp.show()

**OUTPUT:**





**RESULT:**

Thus, the program to demonstrate a random forest was written and executed.

**EX NO: 8**

**DATE:**

**BUILDING EM FOR BAYESIAN NETWORK**

**AIM:**

To write a python program to demonstrate and building EM for Bayesian network.

**PROGRAM:**

import numpy as np

import pandas as pd

import csv

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination

heartDise = pd.read\_csv(“heartdis.csv”)

heartDise = heartDise.replace(‘?’, np.nan)

print (‘Sample instances from the dataset are given below’)

print (heartDise.head())

print (‘\n Attributes and datatypes’)

print (heartDise.dtypes)

model=BayesianModel([(‘age’,’heartdisease’),(‘gender’,’heartdisease’),(‘exang’,’heartdisease’),(‘cp’,’heartdisease’),(‘heartdisease’,’restecg’),(‘heartdisease’,’chol’)])

print (‘\nLearning CPD using Maximum likelihood estimators’)

model.fit (heartDise, estimator=MaximumLikelihoodEstimator)

print (‘\n Inferencing with Bayesian Network:’)

HeartDiseasetest\_infer = VariableElimination(model)

print (‘\n 1. Probability of HeartDisease given evidence= restecg’)

q1=HeartDiseasetest\_infer.query (variables=[‘heartdisease’],evidence={‘restecg’:1})

print(q1)

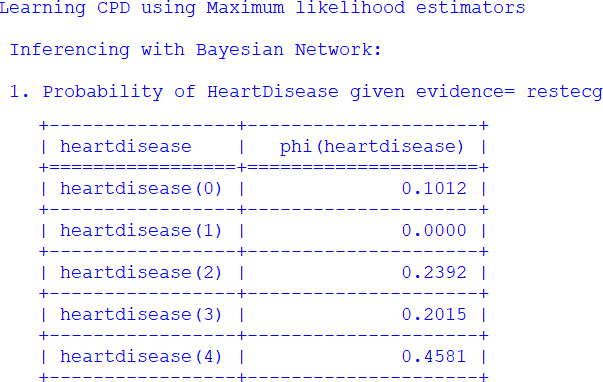
print (‘\n 2. Probability of HeartDisease given evidence= cp ‘)

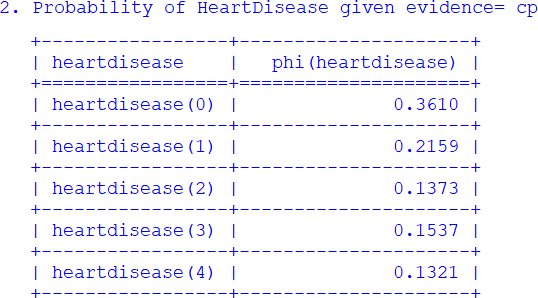
q2=HeartDiseasetest\_infer.query (variables=[‘heartdisease’],evidence={‘cp’:2})

print(q2)

# Dataset: Heart Disease Dataset

**OUTPUT:**





**RESULT:**

Thus, the program to demonstrate EM for Bayesian network was written and executed.

**EX NO: 9**

**DATE:**

**BAGGING USING DECISION TREE CLASSIFICATION**

**AIM:**

To write a python program to implement bagging using decision tree classification and evaluate its performance

**PROGRAM:**

from sklearn import datasets

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

from sklearn.metrics import accuracy\_score

from sklearn.tree import DecisionTreeClassifier

data = datasets.load\_wine(as\_frame = True)

X = data.data

Y = data.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 22)

dtree = DecisionTreeClassifier(random\_state = 22)

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

Y\_pred = dtree.predict(X\_test)

print (“Train data accuracy:”,accuracy\_score(y\_true = y\_train, y\_pred = dtree.predict(X\_train)))

print (“Test data accuracy:”,accuracy\_score(y\_true = y\_test, y\_pred = y\_pred))

#Performance evaluation of bagging

import matplotlib.pyplot as plt

from sklearn import datasets

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

from sklearn.metrics import accuracy\_score

from sklearn.ensemble import BaggingClassifier

data = datasets.load\_wine(as\_frame = True)

X = data.data

Y = data.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 22)

estimator\_range = [2,4,6,8,10,12,14,16]

models = []

scores = []

for n\_estimators in estimator\_range:

 # Create bagging classifier

    clf = BaggingClassifier(n\_estimators = n\_estimators, random\_state = 22)

 # Fit the model

    clf.fit(X\_train, y\_train)

 # Append the model and score to their respective list

    models.append(clf)

    scores.append(accuracy\_score(y\_true = y\_test, y\_pred = clf.predict(X\_test)))

Generate the plot of scores against number of estimators

plt.figure(figsize=(9,6))

plt.plot(estimator\_range, scores)

# Adjust labels and font (to make vispble)

plt.xlabel(“n\_estimators”, fontsize = 18)

plt.ylabel(“score”, fontsize = 18)

plt.tick\_params(labelsize = 16)

# Visualize plot

plt.show()

#Generate Decision Trees from Bagging Classifier

from sklearn import datasets

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

from sklearn.ensemble import BaggingClassifier

from sklearn.tree import plot\_tree

X = data.data

Y = data.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 22)

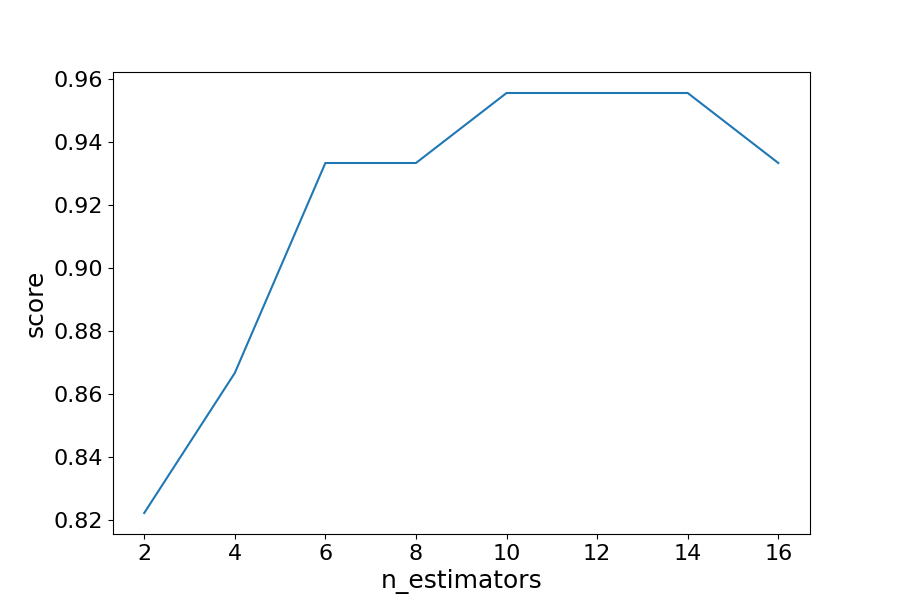
clf = BaggingClassifier(n\_estimators = 12, oob\_score = True,random\_state = 22)

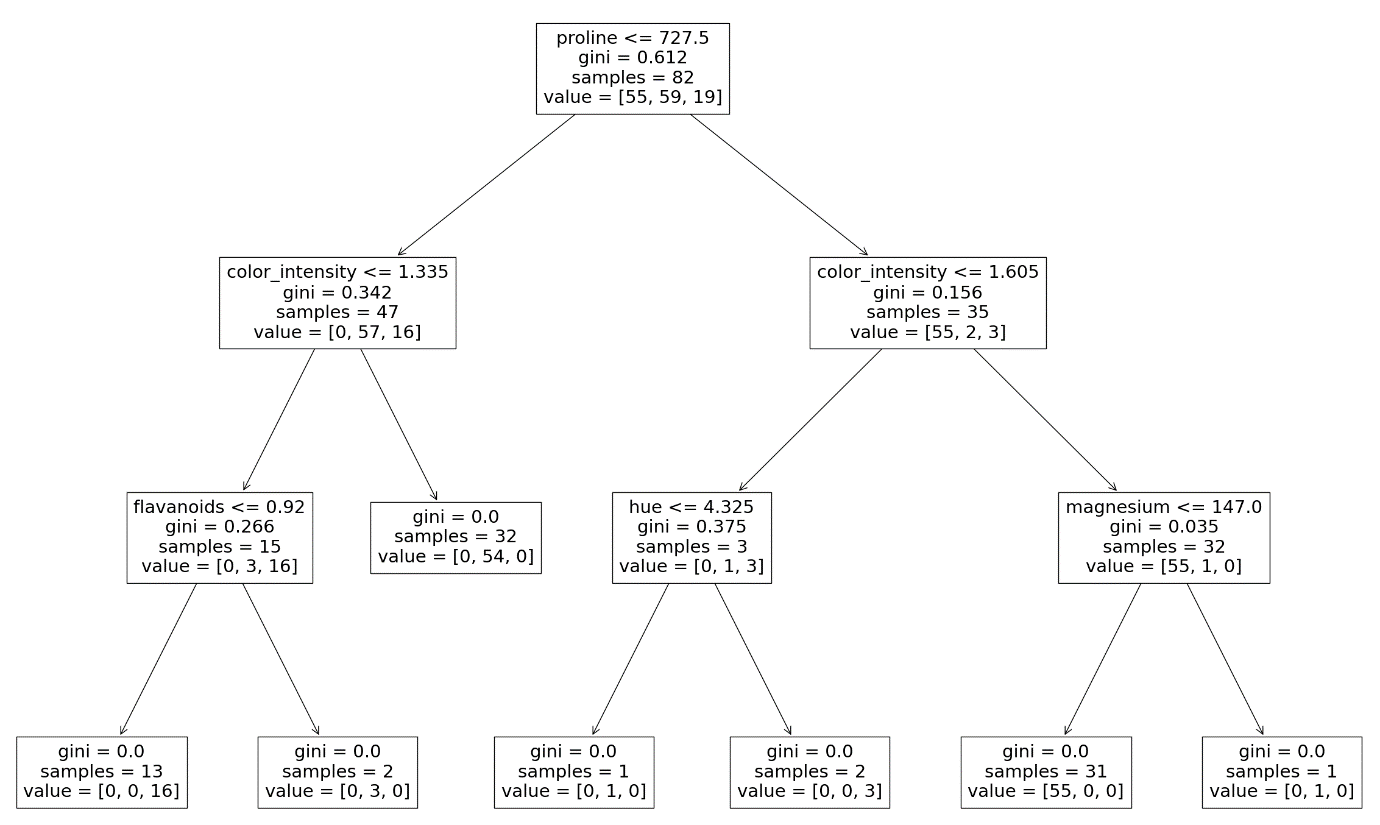
clf.fit(X\_train, y\_train)

plt.figure(figsize=(30, 20))

plot\_tree(clf.estimators\_[0], feature\_names = X.columns)

**OUTPUT:**





**RESULT:**

Thus, the program to implement bagging using decision tree classification and evaluate its performance was written and executed.

**EX NO: 10**

**DATE:**

**KMEANS CLUSTERING**

**AIM:**

To write a python program to demonstrate k means clustering.

**PROGRAM:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

np.random.seed(0)

X = np.random.randn(200, 2) + np.array([2, 2])

X = np.vstack((X, np.random.randn(200, 2) + np.array([-2, -2])))

X = np.vstack((X, np.random.randn(200, 2) + np.array([2, -2])))

X = np.vstack((X, np.random.randn(200, 2) + np.array([-2, 2])))

kmeans = KMeans(n\_clusters=4)

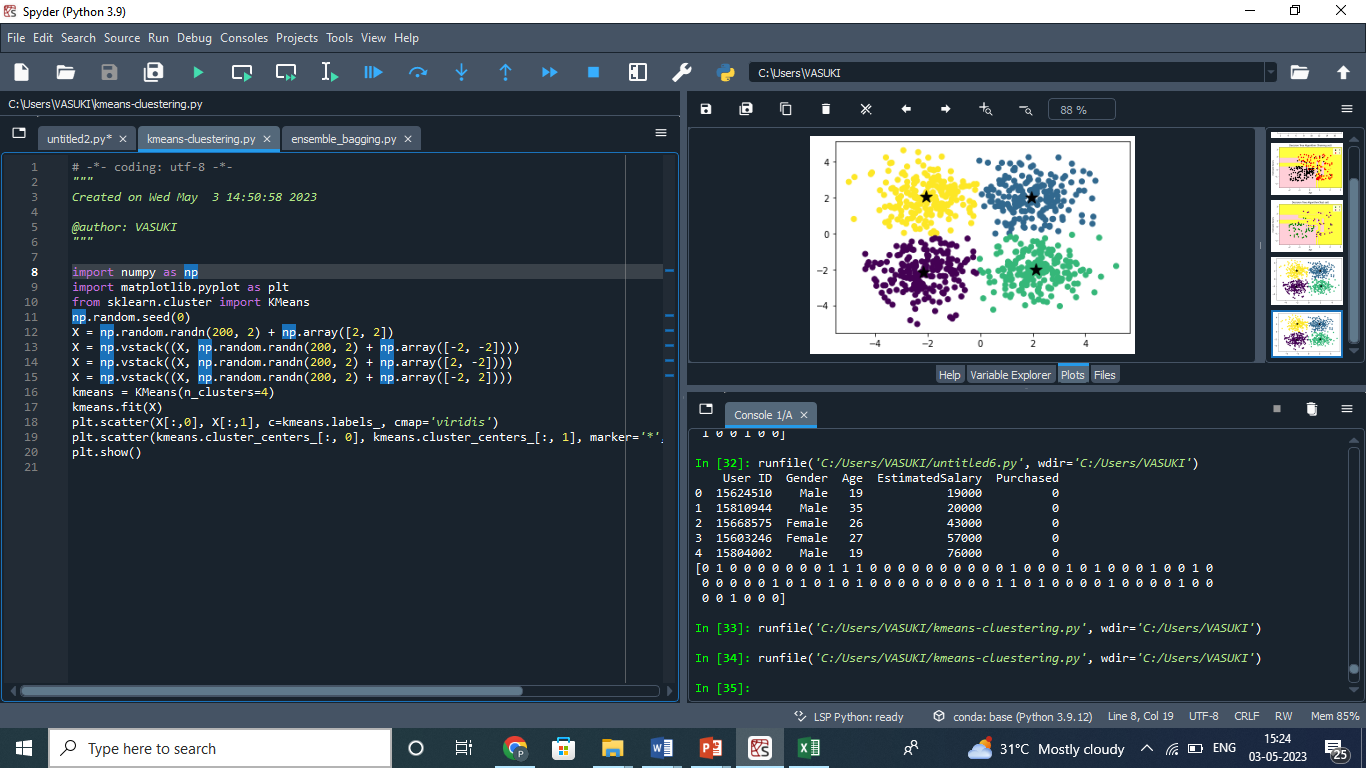
kmeans.fit(X)

plt.scatter(X[:,0], X[:,1], c=kmeans.labels\_, cmap=’viridis’)

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], marker=’\*’, s=200, color=’black’)

plt.show()

**OUTPUT:**



**RESULT:**

Thus, the program to demonstrate k means clustering was written and executed.

**EX NO: 11**

**DATE:**

**NEURAL NETWORK BACK PROPAGATION**

**AIM:**

To write a python program to demonstrate Neural Network Back Propagation.

**PROGRAM:**

Import numpy as np

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

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

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=10000 #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)

d\_hiddenlayer = EH \* hiddengrad

wout += hlayer\_act.T.dot(d\_output) \*lr

bout += np.sum(d\_output, axis=0,keepdims=True) \*lr

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

bh += np.sum(d\_hiddenlayer, axis=0,keepdims=True) \*lr

print (“Input: \n” + str(X))

print (“Actual Output: \n” + str(y))

print (“Predicted Output:\n” ,output)

**OUTPUT:**

Input:

[[2. 9.]

[1. 5.]

[3. 6.]]

Actual Output:

[[0.92]

[0.86]

[0.89]]

('Predicted Output:\n', array([[0.89678563],

[0.87179214],

[0.89995692]]))

**RESULT:**

Thus, the program to demonstrate Neural Network Back Propagation was written and executed.

**EX NO: 12**

**DATE:**

**DEEP NEURAL NETWORK MODEL**

**AIM:**

To write a python program to demonstrate Deep Neural Network model using keras

**PROGRAM:**

from numpy import loadtxt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# load the dataset

dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')

# split into input (X) and output (y) variables

X = dataset[:,0:8]

y = dataset[:,8]

# define the keras model

model = Sequential()

model.add(Dense(12, input\_shape=(8,), activation='relu'))

model.add(Dense(8, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

# compile the keras model

model.compile(loss='binary\_crossentropy',optimizer='adam',metrics=['accuracy'])

# fit the keras model on the dataset

model.fit(X, y, epochs=150, batch\_size=10, verbose=0)

# make class predictions with the model

predictions = (model.predict(X) > 0.5).astype(int)

# evaluate the keras model

\_, accuracy = model.evaluate(X, y, verbose=0)

print(“Accuracy”, accuracy)

# summarize the first 5 cases

for i in range(5):

print('%s => %d (expected %d)' % (X[i].tolist(), predictions[i], y[i]))

# Dataset: Indian\_Diabetes Dataset (pima-indians-diabetes.csv)

**OUTPUT:**

Accuracy for 5 executions,

Accuracy: 75.00

Accuracy: 77.73

Accuracy: 77.60

Accuracy: 78.12

Accuracy: 76.17

[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] => 0 (expected 1)

[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] => 0 (expected 0)

[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] => 1 (expected 1)

[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] => 0 (expected 0)

[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] => 1 (expected 1)

**RESULT:**

Thus, the program to demonstrate Deep Neural Network using Keras was written and executed.