**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**MACHINE LEARNING**

***Submitted by***

**Jahnavi Satish Shanbhag (1BM19CS065)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**May-2022 to July-2022**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “MACHINE LEARNING” carried out by **JAHNAVI SATISH SHANBHAG (1BM19CS065),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a Machine Learning **- (20CS6PCMAL)** work prescribed for the said degree.

Name of the Lab-Incharge:

**Dr G R Asha**           **Dr. Jyothi S Nayak**

Designation:

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Experiment Title** | **Page No.** |
| **1** | **FIND-S Algorithm** | **4** |
| **2** | **Candidate Elimination Algorithm** | **6** |
| **3** | **Decision tree based ID3 Algorithm** | **9** |
| **4** | **Linear Regression** | **17** |
| **5** | **Naive Bayesian Classifier** | **20** |
| **6** | **Bayesian Network** | **23** |
| **7** | **KMeans Clustering Algorithm** | **26** |
| **8** | **EM Algorithm** | **29** |
| **9** | **KNN Algorithm** | **31** |
| **10** | **Weighted Linear Regression Algorithm** | **34** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Course Outcome**

|  |  |
| --- | --- |
|  |  |

## EXPERIMENT 1

1

**CODE:**

import csv

import pandas as pd

import numpy as np

data = pd.read\_csv("Desktop/data.csv")

print(data,"\n")

#array of all the attributes

d = np.array(data)[:,:-1]

print("\n The attributes are: ",d)

target = np.array(data)[:,-1]

print("\n The target is: ",target)

def findS(c,t):

for i, val in enumerate(t):

if val == "Yes":

specific\_hypothesis = c[i].copy()

break

for i, val in enumerate(c):

if t[i] == "Yes":

for x in range(len(specific\_hypothesis)):

if val[x] != specific\_hypothesis[x]:

specific\_hypothesis[x] = '?'

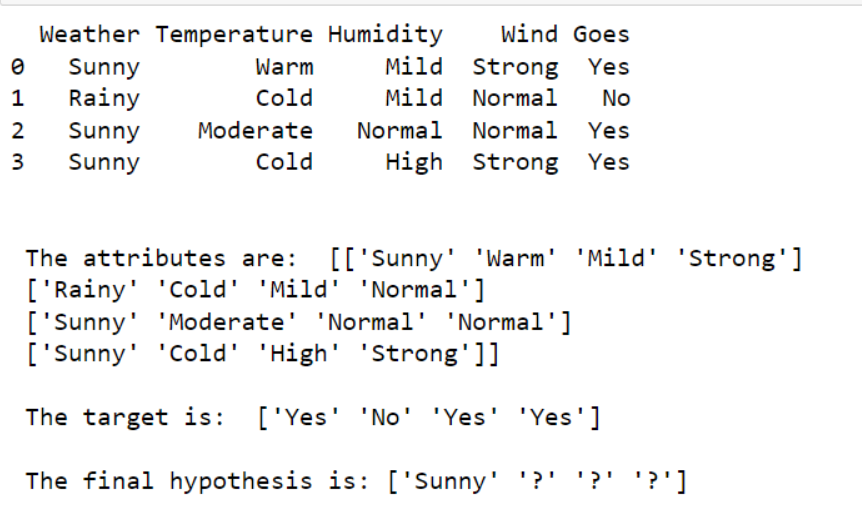
else:

pass

return specific\_hypothesis

print("\n The final hypothesis is:",findS(d,target))

**OUTPUT:**

****

## EXPERIMENT 2

2

**CODE:**

import numpy as np

import pandas as pd

data = pd.read\_csv('Desktop/shape.csv')

concepts = np.array(data.iloc[:,0:-1])

print("\nInstances are:\n",concepts)

target = np.array(data.iloc[:,-1])

print("\nTarget Values are: ",target)

def learn(concepts, target):

specific\_h = concepts[0].copy()

print("\nInitialization of specific\_h and genearal\_h")

print("\nSpecific Boundary: ", specific\_h)

general\_h = [["?" for i in range(len(specific\_h))] for i in range(len(specific\_h))]

print("\nGeneric Boundary: ",general\_h)

for i, h in enumerate(concepts):

print("\nInstance", i+1 , "is ", h)

if target[i] == "yes":

print("Instance is Positive ")

for x in range(len(specific\_h)):

if h[x]!= specific\_h[x]:

specific\_h[x] ='?'

general\_h[x][x] ='?'

if target[i] == "no":

print("Instance is Negative ")

for x in range(len(specific\_h)):

if h[x]!= specific\_h[x]:

general\_h[x][x] = specific\_h[x]

else:

general\_h[x][x] = '?'

print("Specific Bundary after ", i+1, "Instance is ", specific\_h)

print("Generic Boundary after ", i+1, "Instance is ", general\_h)

print("\n")

indices = [i for i, val in enumerate(general\_h) if val == ['?', '?', '?', '?', '?', '?']]

for i in indices:

general\_h.remove(['?', '?', '?', '?', '?', '?'])

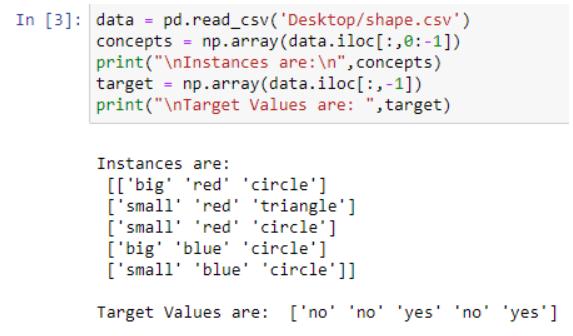
return specific\_h, general\_h

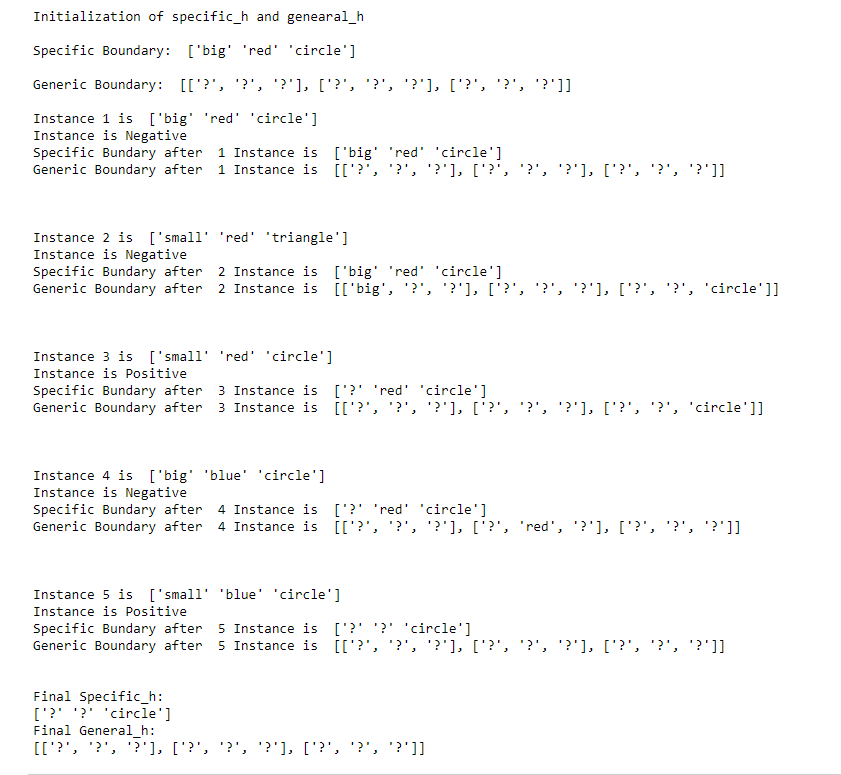
s\_final, g\_final = learn(concepts, target)

print("Final Specific\_h: ", s\_final, sep="\n")

print("Final General\_h: ", g\_final, sep="\n")

**OUTPUT:**

****

****

## EXPERIMENT 3

3

**CODE:**

WITHOUT ALGO:

import csv

def load\_csv(filename):

lines=csv.reader(open(filename,"r"));

dataset = list(lines)

headers = dataset.pop(0)

return dataset,headers

class Node:

def \_\_init\_\_(self,attribute):

self.attribute=attribute

self.children=[]

self.answer=""

def subtables(data,col,delete):

dic={}

coldata=[row[col] for row in data]

attr=list(set(coldata))

counts=[0]\*len(attr)

r=len(data)

c=len(data[0])

for x in range(len(attr)):

for y in range(r):

if data[y][col]==attr[x]:

counts[x]+=1

for x in range(len(attr)):

dic[attr[x]]=[[0 for i in range(c)] for j in range(counts[x])]

pos=0

for y in range(r):

if data[y][col]==attr[x]:

if delete:

del data[y][col]

dic[attr[x]][pos]=data[y]

pos+=1

return attr,dic

def entropy(S):

attr=list(set(S))

if len(attr)==1:

return 0

counts=[0,0]

for i in range(2):

counts[i]=sum([1 for x in S if attr[i]==x])/(len(S)\*1.0)

sums=0

for cnt in counts:

sums+=-1\*cnt\*math.log(cnt,2)

return sums

def compute\_gain(data,col):

attr,dic = subtables(data,col,delete=False)

total\_size=len(data)

entropies=[0]\*len(attr)

ratio=[0]\*len(attr)

total\_entropy=entropy([row[-1] for row in data])

for x in range(len(attr)):

ratio[x]=len(dic[attr[x]])/(total\_size\*1.0)

entropies[x]=entropy([row[-1] for row in dic[attr[x]]])

total\_entropy-=ratio[x]\*entropies[x]

return total\_entropy

def build\_tree(data,features):

lastcol=[row[-1] for row in data]

if(len(set(lastcol)))==1:

node=Node("")

node.answer=lastcol[0]

return node

n=len(data[0])-1

gains=[0]\*n

for col in range(n):

gains[col]=compute\_gain(data,col)

split=gains.index(max(gains))

node=Node(features[split])

fea = features[:split]+features[split+1:]

attr,dic=subtables(data,split,delete=True)

for x in range(len(attr)):

child=build\_tree(dic[attr[x]],fea)

node.children.append((attr[x],child))

return node

def print\_tree(node,level):

if node.answer!="":

print(" "\*level,node.answer)

return

print(" "\*level,node.attribute)

for value,n in node.children:

print(" "\*(level+1),value)

print\_tree(n,level+2)

def classify(node,x\_test,features):

if node.answer!="":

print(node.answer)

return

pos=features.index(node.attribute)

for value, n in node.children:

if x\_test[pos]==value:

classify(n,x\_test,features)

'''Main program'''

dataset,features=load\_csv("data.csv")

node1=build\_tree(dataset,features)

print("The decision tree for the dataset using ID3 algorithm is")

print\_tree(node1,0)

testdata,features=load\_csv("test.csv")

for xtest in testdata:

print("The test instance:",xtest)

print("The label for test instance:",end=" ")

classify(node1,xtest,features)

WITH ALGO:

import numpy as np

import pandas as pd

import math

data = pd.DataFrame(data=pd.read\_csv('data.csv'))

print(data)

def countPosNeg(data):

pos = data.iloc[:,-1:].value\_counts()['yes']

neg = len(data) - pos

return pos, neg

def calcEntropy(pos, neg):

entropy = -(pos/(pos+neg))\*math.log2(pos/(pos+neg)) -(neg/(pos+neg))\*math.log2(neg/(pos+neg))

return entropy

def calcAverageInformation(data):

# iterate through each attribute (col)

attribs = data.iloc[:0,:-1].columns.values

print(attribs)

for attrib in attribs:

# get possible values

values = data[attrib].unique()

valueEntropies = pd.DataFrame(0, columns=['p','n','entropy'], index=values)

print()

print(attrib)

print(valueEntropies)

# iterate through whole dataframe

for i in data.index:

print(data['Answer'][i])

if data['Answer'][i] == 'yes':

valueEntropies[data[attrib]]['p'] += 1

elif data['Answer'][i] == 'no':

valueEntropies[data[attrib]]['n'] += 1

for value in valueEntropies:

value['entropy'] = calcEntropy(value['p'], value['n'])

print(valueEntropies)

return 10

calcAverageInformation(data)

def calcGain(entropy, avg\_info):

return entropy - avg\_info

# data for the total dataset

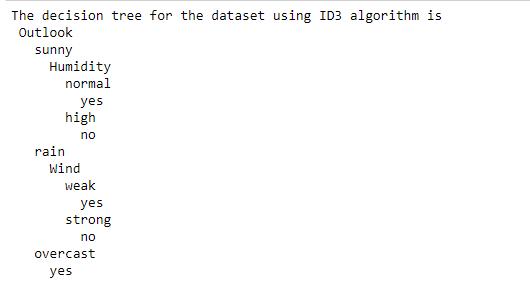
tot\_pos, tot\_neg = countPosNeg(data)

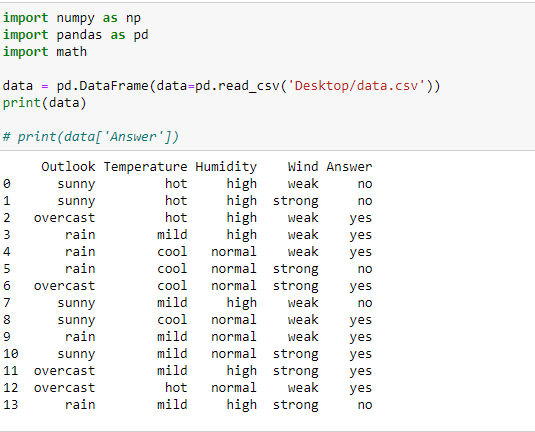
tot\_entropy = calcEntropy(tot\_pos, tot\_neg)

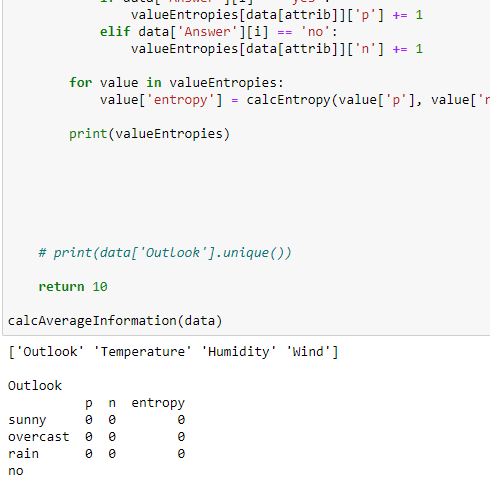
print(tot\_entropy)

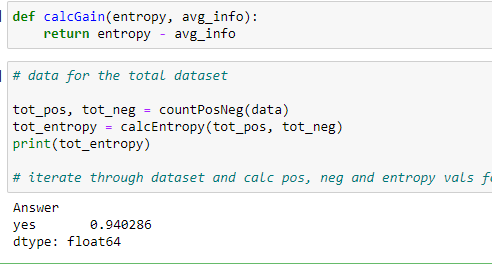
# iterate through dataset and calc pos, neg and entropy vals for each column

**OUTPUT:**

****

****

****

****

## EXPERIMENT 4

4

**CODE:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('salary\_data.csv')

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=1/3, random\_state=0)

# Fitting Simple Linear Regression to the Training set

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

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

# Predicting the Test set results

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

# Visualizing the Training set results

viz\_train = plt

viz\_train.scatter(X\_train, y\_train, color='red')

viz\_train.plot(X\_train, regressor.predict(X\_train), color='blue')

viz\_train.title('Salary VS Experience (Training set)')

viz\_train.xlabel('Year of Experience')

viz\_train.ylabel('Salary')

viz\_train.show()

# Visualizing the Test set results

viz\_test = plt

viz\_test.scatter(X\_test, y\_test, color='red')

viz\_test.plot(X\_train, regressor.predict(X\_train), color='blue')

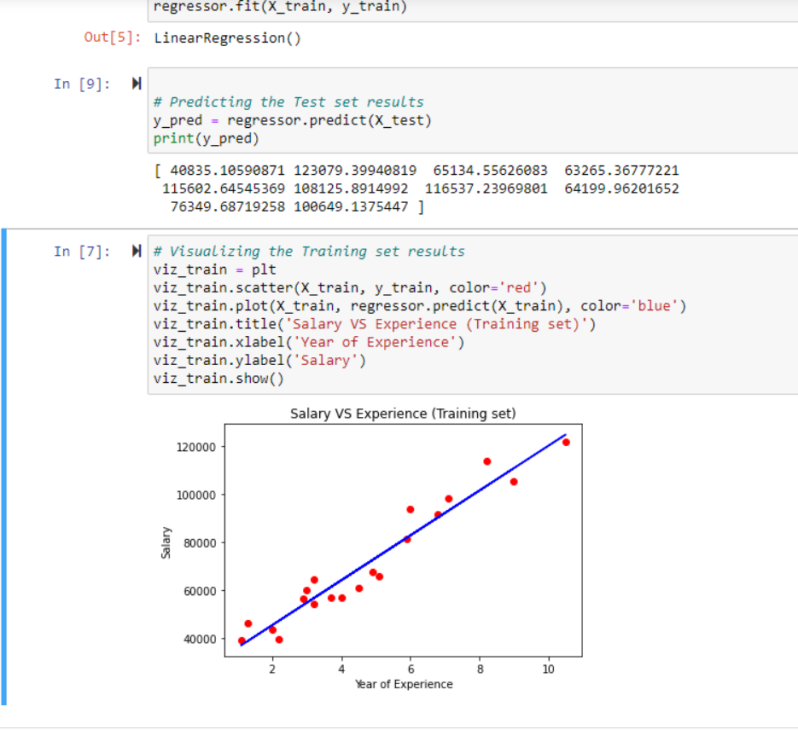
viz\_test.title('Salary VS Experience (Test set)')

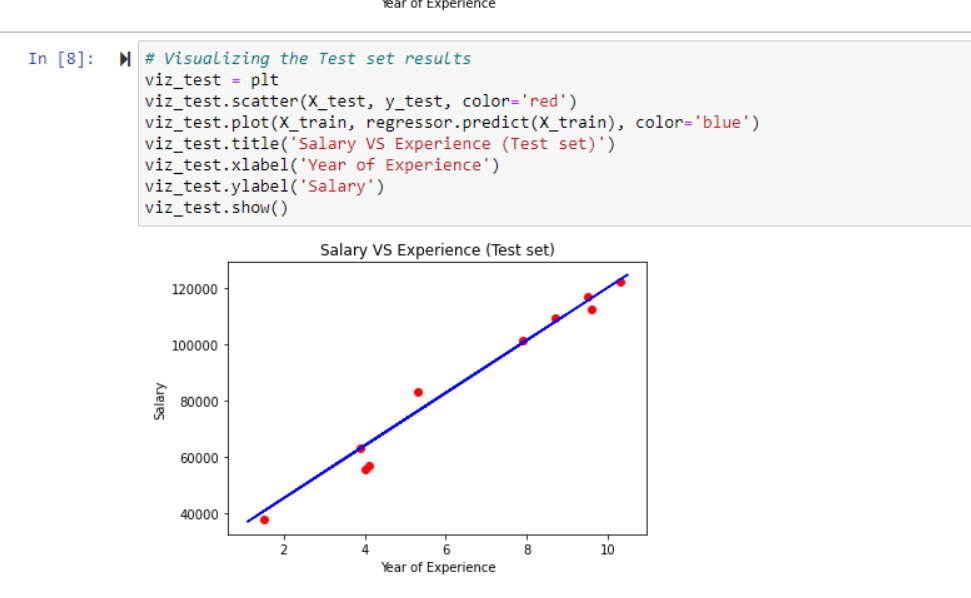
viz\_test.xlabel('Year of Experience')

viz\_test.ylabel('Salary')

viz\_test.show()

**OUTPUT:**

****

****

## EXPERIMENT 5

5

**CODE:**

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

from sklearn import metrics

df = pd.read\_csv("Downloads/data.csv")

feature\_col\_names = ['num\_preg', 'glucose\_conc', 'diastolic\_bp', 'thickness', 'insulin', 'bmi', 'diab\_pred', 'age']

predicted\_class\_names = ['diabetes']

X = df[feature\_col\_names].values

y = df[predicted\_class\_names].values

print(df.head)

xtrain,xtest,ytrain,ytest=train\_test\_split(X,y,test\_size=0.40)

print ('\n the total number of Training Data :',ytrain.shape)

print ('\n the total number of Test Data :',ytest.shape)

clf = GaussianNB().fit(xtrain,ytrain.ravel())

predicted = clf.predict(xtest)

predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])

print('\n Confusion matrix')

print(metrics.confusion\_matrix(ytest,predicted))

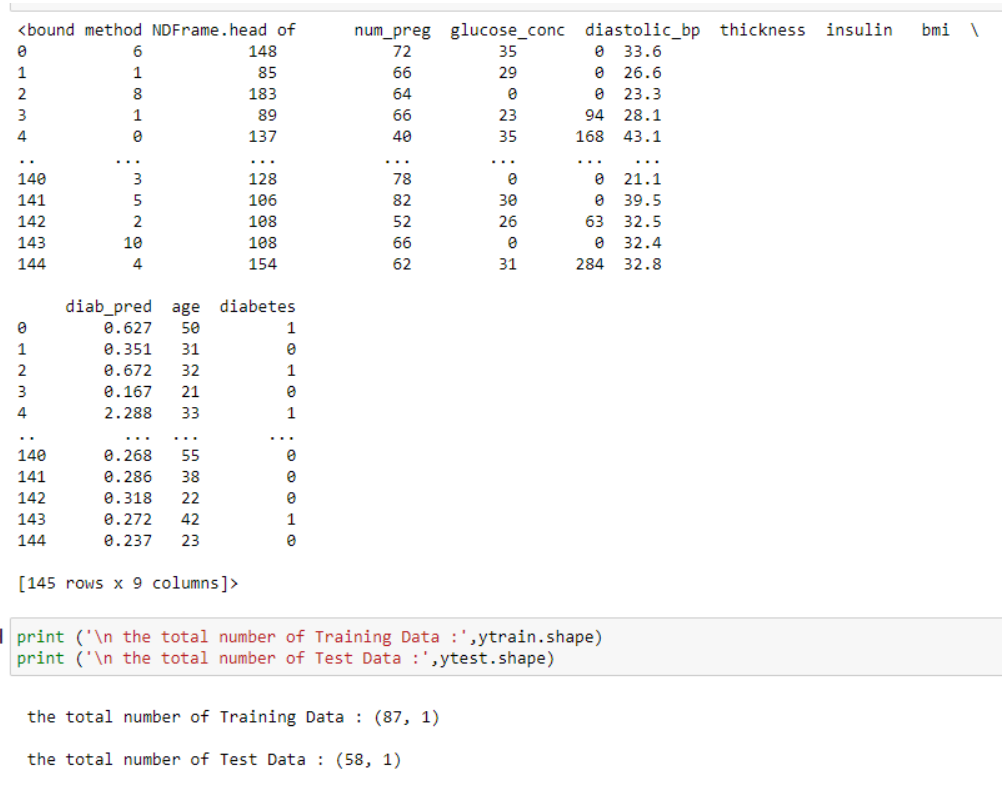
print('\n Accuracy of the classifier is',metrics.accuracy\_score(ytest,predicted))

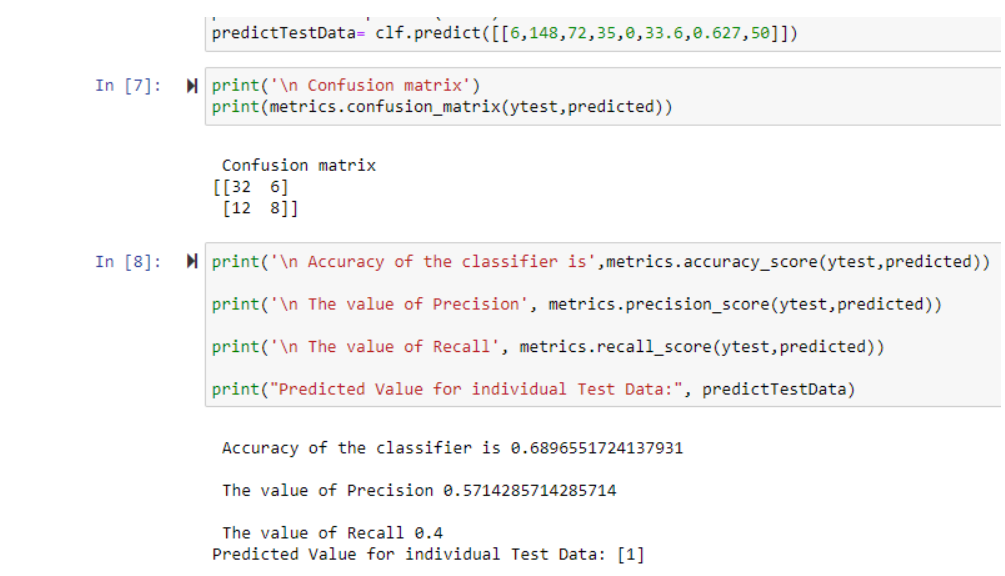
print('\n The value of Precision', metrics.precision\_score(ytest,predicted))

print('\n The value of Recall', metrics.recall\_score(ytest,predicted))

print("Predicted Value for individual Test Data:", predictTestData)

**OUTPUT:**

****

****

## Experiment 6

6

**CODE:**

import numpy as np

import pandas as pd

import csv

import pgmpy

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination

#read Cleveland Heart Disease data

heartDisease = pd.read\_csv('Downloads/data.csv')

heartDisease = heartDisease.replace('?',np.nan)

#display the data

print('Sample instances from the dataset are given below')

print(heartDisease.head())

#display the Attributes names and datatypes

print('\n Attributes and datatypes')

print(heartDisease.dtypes)

#Create Model-Bayesian Network

model = BayesianModel([('age','heartDisease'),('sex','heartDisease'),('exang','heartDisease'),('cp','heartDisease'),('restecg','heartDisease'),('heartDisease','chol')])

#Learning CPDs using Maximum Likelihood Estimators

print('\n Learning CPD using Maximum likelihood estimators')

model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

#Inferencing with Bayesian Network

print('\n Inferencing with Bayesian Network:')

heartDiseasetest\_infer = VariableElimination(model)

#computing the Probability of heartDisease given restecg

print('\n 1.Probability of heartDisease given evidence= restecg :1')

q1=heartDiseasetest\_infer.query(variables=['heartDisease'],evidence={'restecg':1})

print(q1)

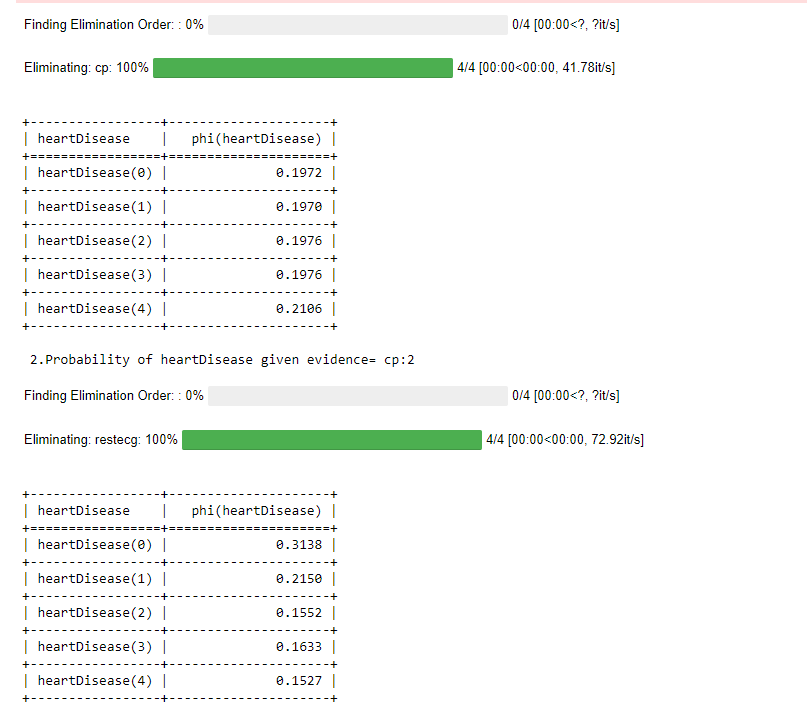
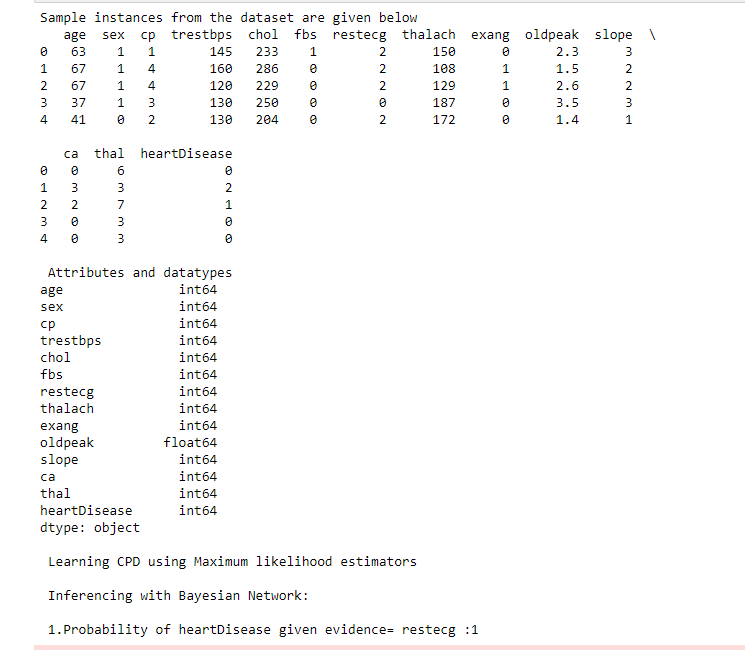
#computing the Probability of heartDisease given cp

print('\n 2.Probability of heartDisease given evidence= cp:2 ')

q2=heartDiseasetest\_infer.query(variables=['heartDisease'],evidence={'cp':2})

print(q2)

**OUTPUT:**

****

## Experiment 7

7

**CODE:**

import matplotlib

import matplotlib.pyplot as plt

import seaborn as sns; sns.set()

import numpy as np

from sklearn.datasets import make\_blobs

X, y\_true = make\_blobs(n\_samples=300, centers=4,

cluster\_std=0.60, random\_state=0)

plt.scatter(X[:, 0], X[:, 1], s=50)

from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters=4)

kmeans.fit(X)

y\_kmeans = kmeans.predict(X)

plt.scatter(X[:, 0], X[:, 1], c=y\_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster\_centers\_

plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)

import pandas as pd

import numpy as np

heartDisease = pd.read\_csv('Downloads/data.csv')

heartDisease = heartDisease.replace('?',np.nan)

heartDisease.head()

trestbpsX = heartDisease.loc[:,'trestbps']

cholY = heartDisease.loc[:,'chol']

plt.scatter(trestbpsX, cholY, s=50)

kmeans2 = KMeans(n\_clusters=2)

combined\_list = list(zip(trestbpsX, cholY))

kmeans2.fit(combined\_list)

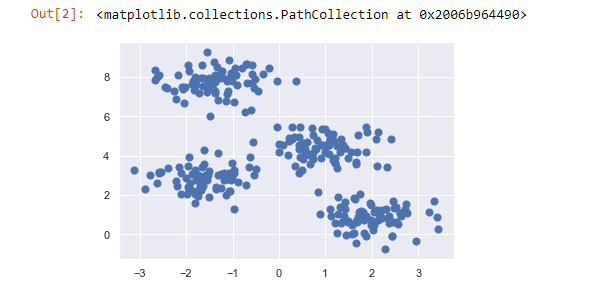
y\_kmeans2 = kmeans2.predict(combined\_list)

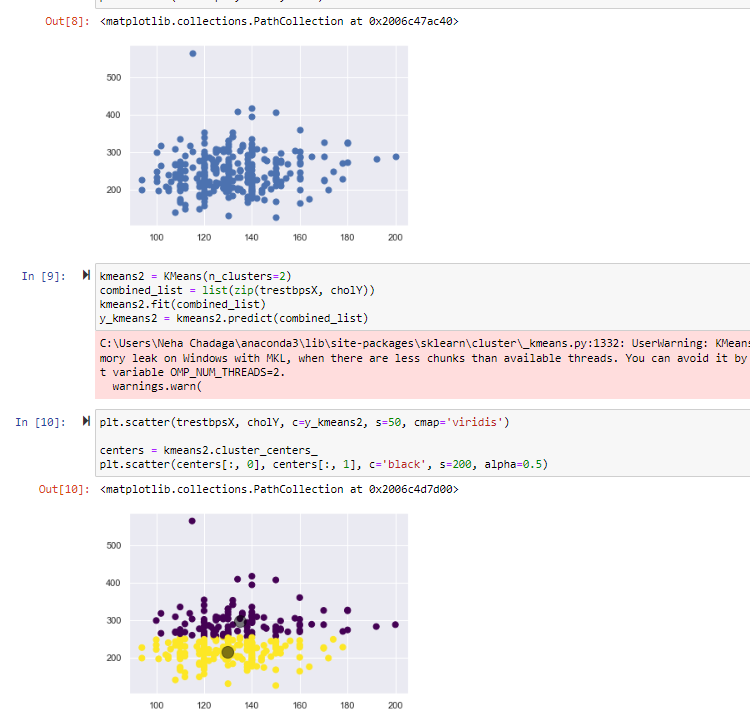
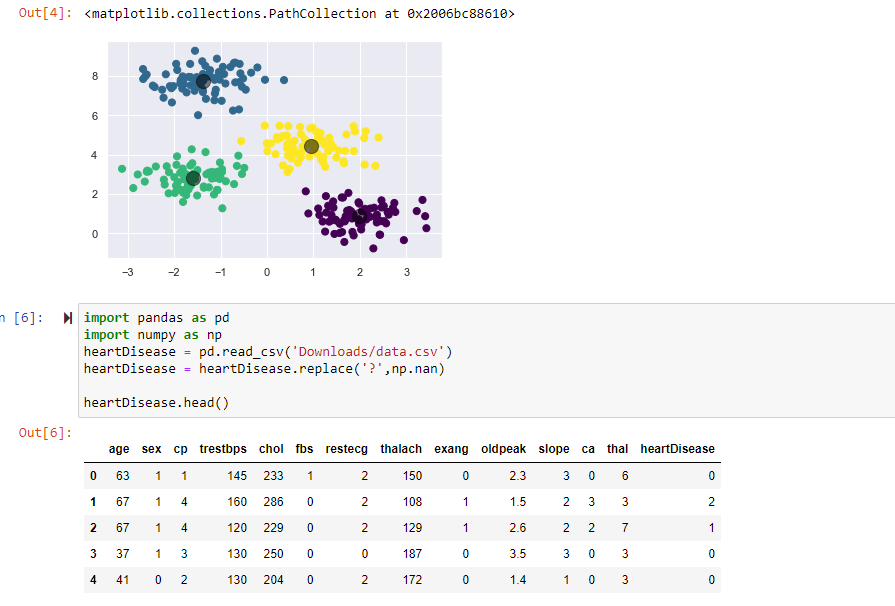
plt.scatter(trestbpsX, cholY, c=y\_kmeans2, s=50, cmap='viridis')

centers = kmeans2.cluster\_centers\_

plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)

**OUTPUT:**

****

****

## Experiment 8

8

**CODE:**

from sklearn import datasets

from sklearn.cluster import KMeans

from sklearn.utils import shuffle

import numpy as np

import pandas as pd

iris=datasets.load\_iris()

X=iris.data

Y=iris.target

#Shuffle of Data

X,Y = shuffle(X,Y)

model=KMeans(n\_clusters=3,init='k-means++',max\_iter=10,n\_init=1,random\_state=3425)

#Training of the model

model.fit(X)

# This is what KMeans thought (Prediction)

Y\_Pred=model.labels\_

from sklearn.metrics import confusion\_matrix

cm=confusion\_matrix(Y,Y\_Pred)

print(cm)

from sklearn.metrics import accuracy\_score

print(accuracy\_score(Y,Y\_Pred))

#Defining EM Model

from sklearn.mixture import GaussianMixture

model2=GaussianMixture(n\_components=3,random\_state=3425)

#Training of the model

model2.fit(X)

#Predicting classes for our data

Y\_predict2= model2.predict(X)

#Accuracy of EM Model

from sklearn.metrics import confusion\_matrix

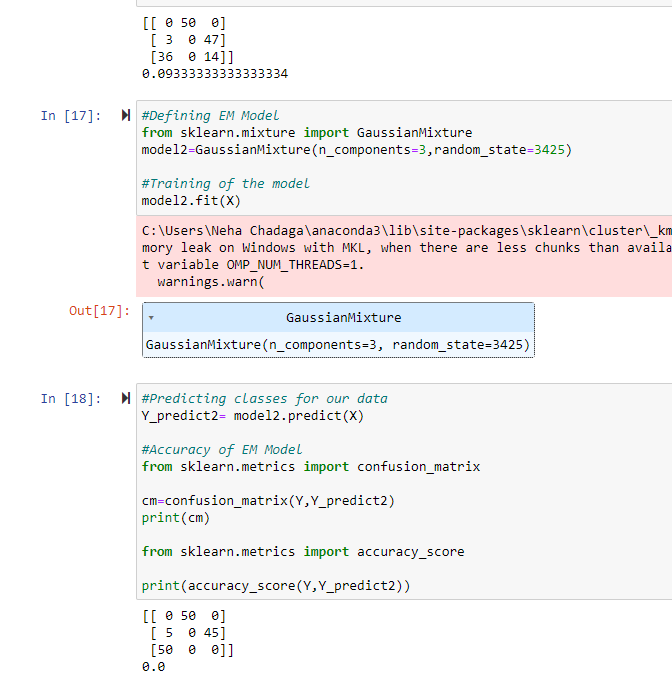
cm=confusion\_matrix(Y,Y\_predict2)

print(cm)

from sklearn.metrics import accuracy\_score

print(accuracy\_score(Y,Y\_predict2))

**OUTPUT:**

****

## Experiment 9

9

**CODE:**

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn import datasets

iris = datasets.load\_iris()

X = iris.data

Y = iris.target

print('sepal-length','sepal-width','petal-length','petal-width')

print(X)

print('target')

print(Y)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X,Y,test\_size=0.3)

classier = KNeighborsClassifier(n\_neighbors=5)

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

y\_pred=classier.predict(x\_test)

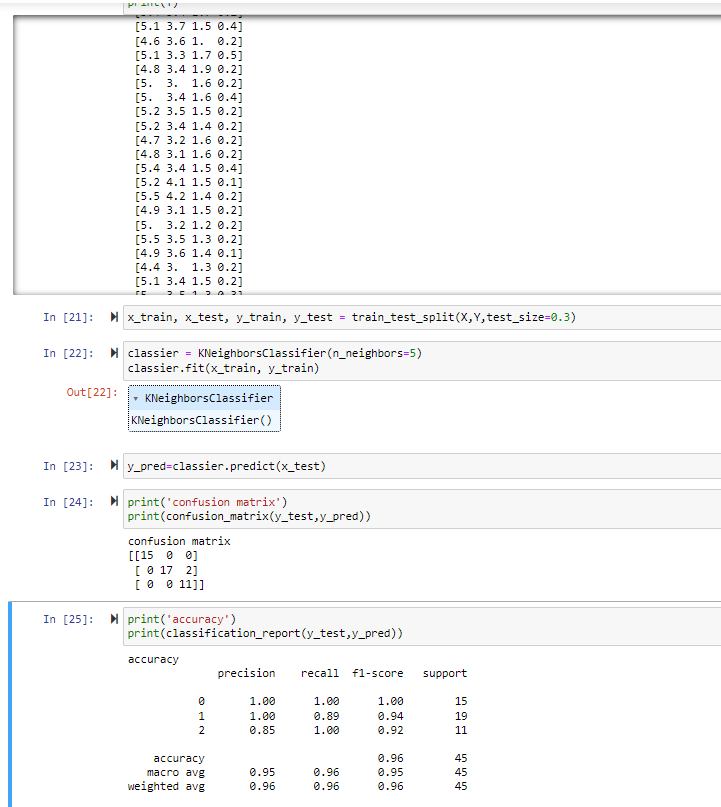
print('confusion matrix')

print(confusion\_matrix(y\_test,y\_pred))

print('accuracy')

print(classification\_report(y\_test,y\_pred))

**OUTPUT:**



## Experiment 10

10

**CODE:**

from numpy import \*

from os import listdir

import matplotlib

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np1

import numpy.linalg as np

from scipy.stats.stats import pearsonr

def kernel(point,xmat, k):

m,n = np1.shape(xmat)

weights = np1.mat(np1.eye((m)))

for j in range(m):

diff = point - X[j]

weights[j,j] = np1.exp(diff\*diff.T/(-2.0\*k\*\*2))

return weights

def localWeight(point,xmat,ymat,k):

wei = kernel(point,xmat,k)

W = (X.T\*(wei\*X)).I\*(X.T\*(wei\*ymat.T))

return W

def localWeightRegression(xmat,ymat,k):

m,n = np1.shape(xmat)

ypred = np1.zeros(m)

for i in range(m):

ypred[i] = xmat[i]\*localWeight(xmat[i],xmat,ymat,k)

return ypred

# load data points

data = pd.read\_csv('tips.csv')

bill = np1.array(data.total\_bill)

tip = np1.array(data.tip)

#preparing and add 1 in bill

mbill = np1.mat(bill)

mtip = np1.mat(tip) # mat is used to convert to n dimesiona to 2 dimensional array form

m= np1.shape(mbill)[1]

# print(m) 244 data is stored in m

one = np1.mat(np1.ones(m))

X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE

#print(X)

#set k here

ypred = localWeightRegression(X,mtip,2)

SortIndex = X[:,1].argsort(0)

xsort = X[SortIndex][:,0]

fig = plt.figure()

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

ax.scatter(bill,tip, color='blue')

ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)

plt.xlabel('Total bill')

plt.ylabel('Tip')

plt.show()

import numpy as np

from bokeh.plotting import figure, show, output\_notebook

from bokeh.layouts import gridplot

from bokeh.io import push\_notebook

def local\_regression(x0, X, Y, tau):# add bias term

x0 = np.r\_[1, x0] # Add one to avoid the loss in information

X = np.c\_[np.ones(len(X)), X]

# fit model: normal equations with kernel

xw = X.T \* radial\_kernel(x0, X, tau) # XTranspose \* W

beta = np.linalg.pinv(xw @ X) @ xw @ Y #@ Matrix Multiplication or Dot Product

# predict value

return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction

def radial\_kernel(x0, X, tau):

return np.exp(np.sum((X - x0) \*\* 2, axis=1) / (-2 \* tau \* tau))

# Weight or Radial Kernal Bias Function

n = 1000

# generate dataset

X = np.linspace(-3, 3, num=n)

print("The Data Set ( 10 Samples) X :\n",X[1:10])

Y = np.log(np.abs(X \*\* 2 - 1) + .5)

print("The Fitting Curve Data Set (10 Samples) Y :\n",Y[1:10])

# jitter X

X += np.random.normal(scale=.1, size=n)

print("Normalised (10 Samples) X :\n",X[1:10])

domain = np.linspace(-3, 3, num=300)

print(" Xo Domain Space(10 Samples) :\n",domain[1:10])

def plot\_lwr(tau):

# prediction through regression

prediction = [local\_regression(x0, X, Y, tau) for x0 in domain]

plot = figure(plot\_width=400, plot\_height=400)

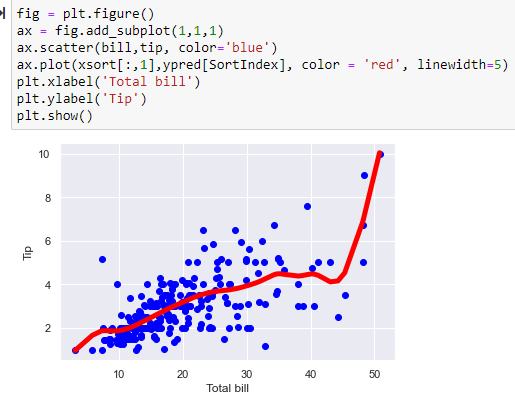
plot.title.text='tau=%g' % tau

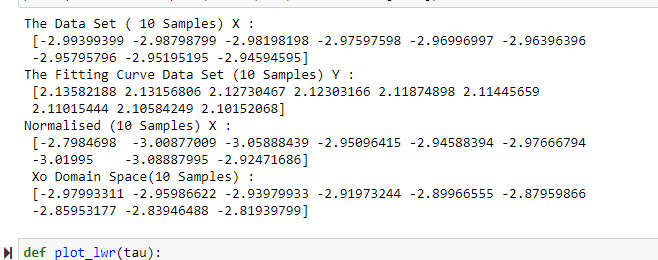
plot.scatter(X, Y, alpha=.3)

plot.line(domain, prediction, line\_width=2, color='red')

return plot

**OUTPUT:**

****

****