**Name: SUPRAGYA**

**College Roll No.: 2019348**

**Examination Roll No.: 19066570056**

**Semester: VIth**

**Subject: MACHINE LEARNING**

**UNIQUE PAPER CODE: 32347607**

**CONTENT**

1. Perform elementary mathematical operations in Octave/MATLAB/R like addition, multiplication, division and exponentiation.
2. Perform elementary logical operations in Octave/MATLAB/R (like OR, AND, Checking for Equality, NOT, XOR).
3. Create, initialize and display simple variables and simple strings and use simple formatting for variable.
4. Create/Define single dimension / multi-dimension arrays, and arrays with specific values like array of all ones, all zeros, array with random values within a range, or a diagonal matrix.
5. Use command to compute the size of a matrix, size/length of a particular row/column, load data from a text file, store matrix data to a text file, finding out variables and their features in the current scope.
6. Perform basic operations on matrices (like addition, subtraction, multiplication) and display specific rows or columns of the matrix.
7. Perform other matrix operations like converting matrix data to absolute values, taking the negative of matrix values, adding/ removing rows/columns from a matrix, finding the maximum or minimum values in a matrix or in a row/column, and finding the sum of some/all elements in a matrix.
8. Create various type of plots/charts like histograms, plot based on sine/cosine function based on data from a matrix. Further label different axes in a plot and data in a plot.
9. Generate different subplots from a given plot and color plot data.
10. Use conditional statements and different type of loops based on simple example/s.
11. Perform vectorized implementation of simple matrix operation like finding the transpose of a matrix, adding, subtracting or multiplying two matrices.
12. Implement Linear Regression problem. For example, based on the “Advertising” dataset comprising of budget of TV, Radio etc. and the sales data, predict the estimated sales for TV budget.
13. Based on multiple features/variables perform Linear Regression on “Advertising” dataset. For example, based on the budget of TV, Radio and Newspaper, predict the overall sales.
14. Implement a classification/ logistic regression problem. For example, based on different features of “diabetes” data, classify, whether a woman is diabetic or not.
15. Use some function for regularization of “BOSTON” dataset available in ‘sklearn library’.
16. Use some function for neural networks, like Stochastic Gradient Descent or backpropagation - algorithm to predict the value of a variable based on the dataset of problem 14.
17. Implement Simple Linear Regression on “Advertising” dataset using Analytical Method.
18. Implement Multiple Linear Regression on “Advertising” dataset using Normal Equation Method.
19. **Perform elementary mathematical operations in Octave/MATLAB/R like addition, multiplication, division and exponentiation.**

**CODE:**

n1=int(input("ENTER THE FIRST NUMBER: "))

n2=int(input("ENTER THE SECOND NUMBER: "))

print("SELECT ANY ONE OPTION:\n\t1. ADDITION\n\t2. SUBTRACTION\n\t3. MULTIPLICATION\n\t4. DIVISION\n\t5. EXPONENTIATION")

option='y'

while(option=='y' or option=='Y'):

choice=int(input("ENTER YOUR CHOICE:"))

if choice==1:

s=n1+n2

print("Addition:",s)

elif choice==2:

d=n1-n2

print("Subtraction:",d)

elif choice==3:

m=n1\*n2

print("Multiplication:",m)

elif choice==4:

div=n1/n2

print("Division:",div)

elif choice==5:

e=n1\*\*n2

print("Exponentiation:",e)

else:

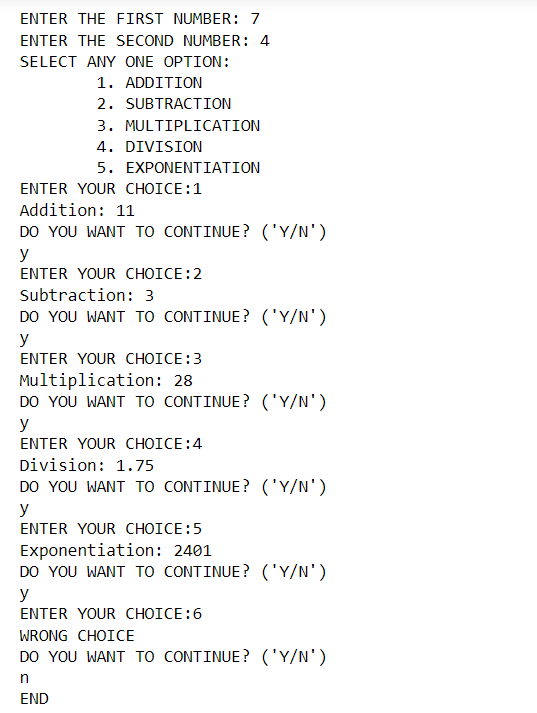
print("WRONG CHOICE")

print("DO YOU WANT TO CONTINUE? ('Y/N')")

option=input()

print("END")

**OUTPUT:**



1. **Perform elementary logical operations in Octave/MATLAB/R (like OR, AND, Checking for Equality, NOT, XOR).**

**CODE:**

X=True

Y=False

print("X AND Y is: ", X and Y)

print("X OR Y is: ", X or Y)

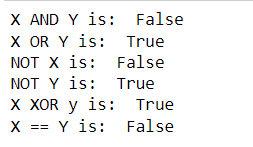
print("NOT X is: ", not X)

print("NOT Y is: ", not Y)

print("X XOR y is: ", X ^ Y)

print("X == Y is: ", X==Y)

**OUTPUT:**



1. **Create, initialize and display simple variables and simple strings and use simple formatting for variable.**

**CODE:**

x=45

y=25

s="Hello"

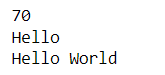
z=x+y

print(z)

print(s)

print(s+" World")

**OUTPUT:**



1. **Create/Define single dimension / multi-dimension arrays, and arrays with specific values like array of all ones, all zeros, array with random values within a range, or a diagonal matrix.**

**CODE:**

import numpy as np

m=np.array([1,2,3])

print("Matrix 1: ", m)

n=np.array([[4,5,6],[1,3,7]])

print("Matrix 2:\n", n,'\n')

b=np.ones((2,3)).astype('int32')

print("Ones Matrix:\n", b,'\n')

a=np.zeros((2,2)).astype('int32')

print("Zeroes Matrix:\n", a,'\n')

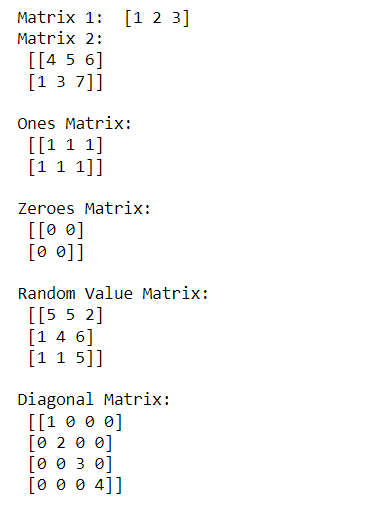
c=np.random.randint(1,7,size=(3,3))

print("Random Value Matrix:\n", c,'\n')

d=np.diag([1,2,3,4])

print("Diagonal Matrix:\n",d)

**OUTPUT:**



1. **Use command to compute the size of a matrix, size/length of a particular row/column, load data from a text file, store matrix data to a text file, finding out variables and their features in the current scope.**

**CODE:**

import numpy as np

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

print("SHAPE: ", x.shape,'\n')

print("SIZE: ", x.size,'\n')

print("ITEM SIZE: ", x.itemsize,'\n')

file=np.random.randint(1,20,size=(3,3))

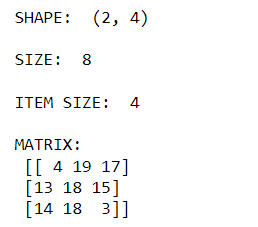
np.savetxt('data1.txt',file)

np.genfromtxt('data1.txt',delimiter=' ')

file=file.astype('int32')

print("MATRIX:\n", file)

**OUTPUT:**



1. **Perform basic operations on matrices (like addition, subtraction, multiplication) and display specific rows or columns of the matrix.**

**CODE:**

import numpy as np

a=np.array([[1,2,3],[4,5,6]])

b=np.array([[4,5,6],[7,8,9]])

s=a+b

print("Matrix after ADDITION:\n", s,'\n')

print("3rd Column of Sum Matrix:\n", s[:,2],'\n')

d=b-a

print("Matrix after SUBTRACTION:\n", d,'\n')

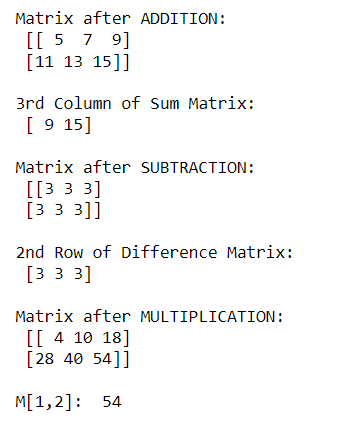
print("2nd Row of Difference Matrix:\n", d[1,:],'\n')

m=a\*b

print("Matrix after MULTIPLICATION:\n", m,'\n')

print("M[1,2]: ", m[1,2],'\n')

**OUTPUT:**



1. **Perform other matrix operations like converting matrix data to absolute values, taking the negative of matrix values, adding/ removing rows/columns from a matrix, finding the maximum or minimum values in a matrix or in a row/column, and finding the sum of some/all elements in a matrix.**

**CODE:**

import numpy as np

c=np.array([[1,2,4],[5,6,8]])

print("MATRIX:\n", c, '\n')

v=np.sin(c)

print("SINE MATRIX:\n", v,'\n')

print("ABSOLUTE MATRIX:\n", np.abs(v),'\n')

d=np.append(c,np.array([[9,10,12]]), axis=0)

print("APPENDED MATRIX:\n", d,'\n')

print("2nd ROW DELETED:\n", np.delete(d,1,0),'\n')

print("2nd COLUMN DELETED:\n", np.delete(d,1,1),'\n')

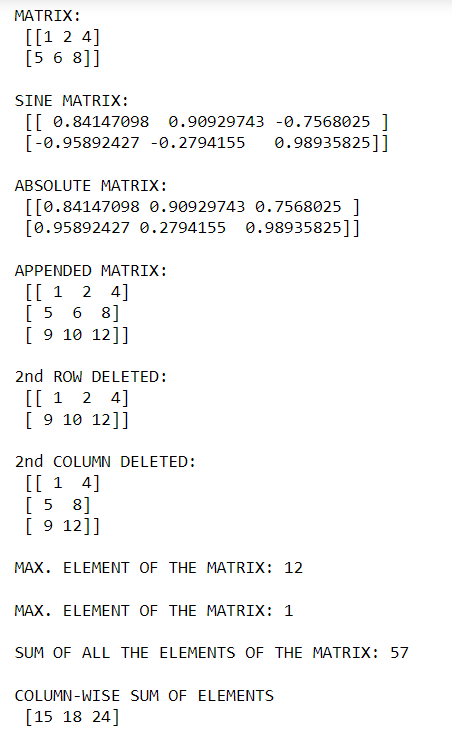
print("MAX. ELEMENT OF THE MATRIX:", np.max(d),'\n')

print("MAX. ELEMENT OF THE MATRIX:", np.min(d),'\n')

print("SUM OF ALL THE ELEMENTS OF THE MATRIX:",np.sum(d),'\n')

print("COLUMN-WISE SUM OF ELEMENTS\n", np.sum(d,axis=0))

**OUTPUT:**



1. **Create various type of plots/charts like histograms, plot based on sine/cosine function based on data from a matrix. Further label different axes in a plot and data in a plot.**

**CODE:**

import matplotlib.pyplot as plt

import numpy as np

# x-coordinates of left sides of bars

left = [1, 2, 3, 4, 5]

# heights of bars

height = [10, 24, 36, 40, 5]

# labels for bars

tick\_label = ['one', 'two', 'three', 'four', 'five']

# plotting a bar chart

plt.bar(left, height, tick\_label = tick\_label, width = 0.8, color = ['blue', 'pink','green','yellow','red'])

# naming the x-axis

plt.xlabel('x - axis')

# naming the y-axis

plt.ylabel('y - axis')

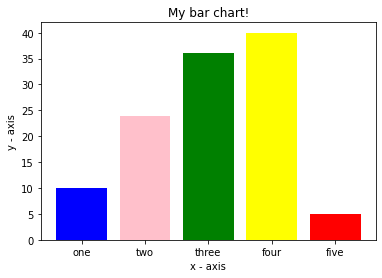
# plot title

plt.title('My bar chart!')

# function to show the plot

plt.show()

**OUTPUT:**



**CODE:**

# defining labels

activities = ['eat', 'sleep', 'work', 'play']

# portion covered by each label

slices = [3, 7, 8, 6]

# color for each label

colors = ['red', 'pink', 'green', 'orange']

# plotting the pie chart

plt.pie(slices, labels = activities, colors=colors, startangle=90, radius = 1.2, autopct = '%1.1f%%')

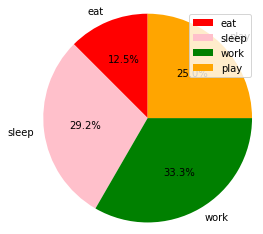
# plotting legend

plt.legend()

# showing the plot

plt.show()

**OUTPUT:**



**CODE:**

x = np.arange(0,4\*np.pi,0.1) # start,stop,step

y = np.sin(x)

z = np.cos(x)

plt.plot(x,y,x,z)

# naming the x-axis

plt.xlabel('x - axis')

# naming the y-axis

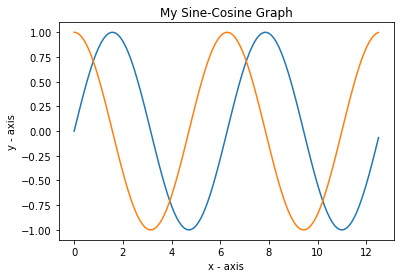
plt.ylabel('y - axis')

# plot title

plt.title('My Sine-Cosine Graph')

plt.show()

**OUTPUT:**



1. **Generate different subplots from a given plot and color plot data.**

**CODE:**

import matplotlib.pyplot as plt

import numpy as np

x = np.linspace(0, 2 \* np.pi, 400)

y = np.sin(x \*\* 2)

fig, axs = plt.subplots(2, 2)

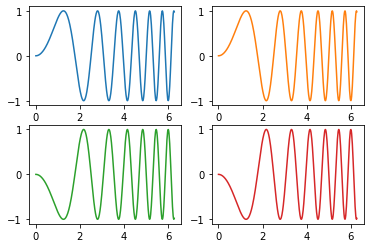
axs[0, 0].plot(x, y)

axs[0, 1].plot(x, y, 'tab:orange')

axs[1, 0].plot(x, -y, 'tab:green')

axs[1, 1].plot(x, -y, 'tab:red')

**OUTPUT:**



1. **Use conditional statements and different type of loops based on simple example/s.**

**CODE:**

s=input("Enter a String: ")

s1=""

for i in s:

s1=i+s1

if(s1==s):

print(s, "is Palindrome")

else:

print(s, "is not Palindrome")

**OUTPUT:**





1. **Perform vectorized implementation of simple matrix operation like finding the transpose of a matrix, adding, subtracting or multiplying two matrices.**

**CODE:**

import numpy as np

a=np.array([[1,2,3],[4,5,6],[1,1,1]])

b=np.array([[6,7,8],[9,10,11],[2,2,2]])

print("MATRIX 1:\n", a,'\n')

print("MATRIX 2:\n", b,'\n')

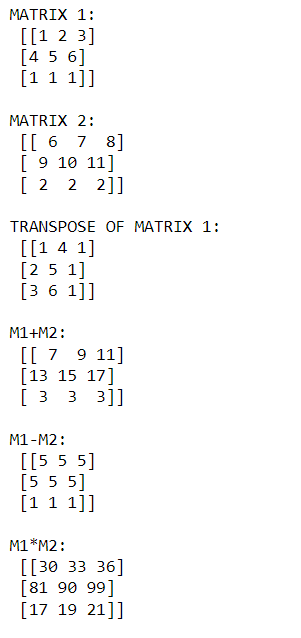
print("TRANSPOSE OF MATRIX 1:\n", np.transpose(a),'\n')

print("M1+M2:\n", a+b,'\n')

print("M1-M2:\n", b-a,'\n')

print("M1\*M2:\n", np.matmul(a,b),'\n')

**OUTPUT:**



1. **Implement Linear Regression problem. For example, based on the “Advertising” dataset comprising of budget of TV, Radio etc. and the sales data, predict the estimated sales for TV budget.**

**CODE:**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn.linear\_model import LinearRegression

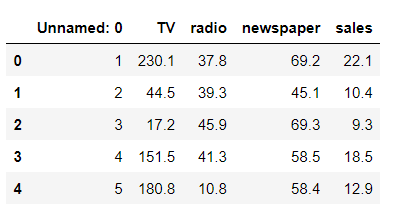
from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

adv = pd.read\_csv("Advertising.csv")

adv.head()

**OUTPUT:**



**CODE:**

#check for nulls in the data

adv.isnull().sum()

plt.figure(figsize=(16, 8))

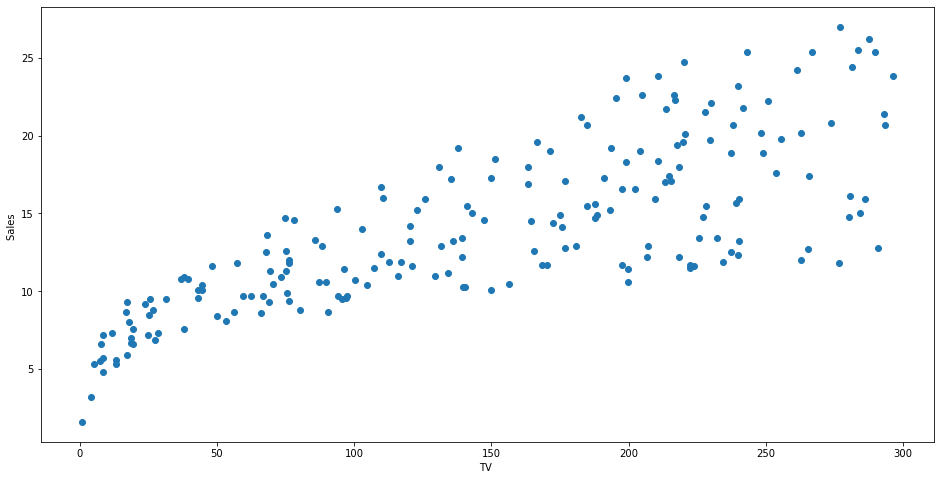
plt.scatter(adv['TV'], adv['sales'])

plt.xlabel("TV ")

plt.ylabel("Sales ")

plt.show()

**OUTPUT:**



**CODE:**

x = adv['TV'].values.reshape(-1,1)

y = adv['sales'].values.reshape(-1,1)

# split data into train and test

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.3,random\_state=0)

#fit the model using Linear Regression

linreg = LinearRegression()

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

print("INTERCEPT: ", linreg.intercept\_[0]) #Intercept

print("\nCOEFFICIENT: ", linreg.coef\_[0][0]) #Coefficient

print("\nThe linear model is: y = {:.5} + {:.5}TV".format(linreg.intercept\_[0], linreg.coef\_[0][0]))

# Make predictions using the testing set

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

y=linreg.predict(np.array([1000]).reshape(1,-1)) #Prediction

print("\nPredicted Value for the SALES of TV: ", y)

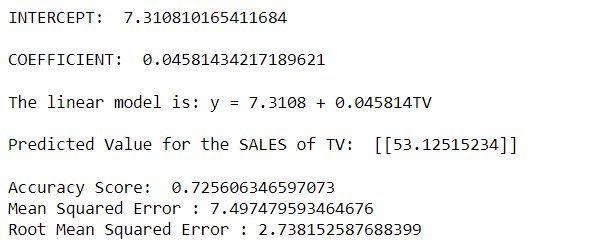
#Accuracy Score

print("\nAccuracy Score: ", linreg.score(x\_test,y\_test))

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:**



**CODE:**

print('Train Score :', linreg.score(x\_train,y\_train))

print('Test Score:', linreg.score(x\_test,y\_test))

**OUTPUT:**



**CODE:**

plt.figure(figsize=(16, 8))

plt.scatter(x\_test, y\_test, color="red")

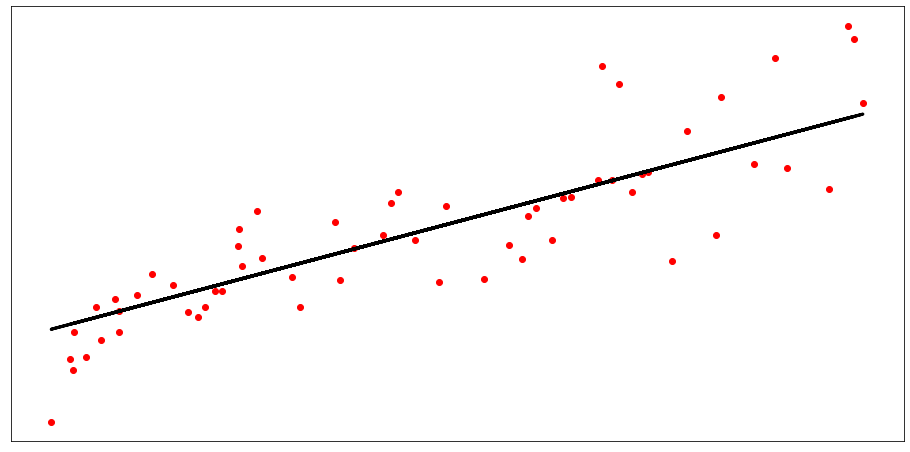
plt.plot(x\_test, y\_pred, color="black", linewidth=3)

plt.xticks(())

plt.yticks(())

plt.show()

**OUTPUT:**



**CODE:**

errors = list()

for i in range(len(y\_test)):

# calculate error

err = (y\_test[i] - y\_pred[i])\*\*2

# store error

errors.append(err)

# plot errors

plt.plot(errors)

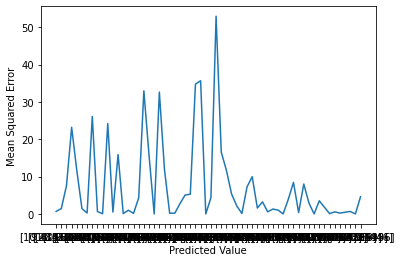
plt.xticks(ticks=[i for i in range(len(errors))], labels=y\_pred)

plt.xlabel('Predicted Value')

plt.ylabel('Mean Squared Error')

plt.show()

**OUTPUT:**



1. **Based on multiple features/variables perform Linear Regression on “Advertising” dataset. For example, based on the budget of TV, Radio and Newspaper, predict the overall sales.**

**CODE:**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn.linear\_model import LinearRegression

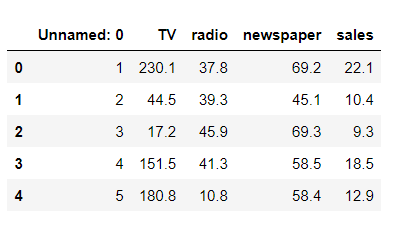
from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

adv = pd.read\_csv("Advertising.csv")

adv.head()

**OUTPUT:**



**CODE:**

x = adv.drop(['sales', 'Unnamed: 0'], axis=1)

y = adv['sales'].values.reshape(-1,1)

# split data into train and test

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.3,random\_state=0)

#fit the model using Linear Regression

linreg = LinearRegression()

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

print("INTERCEPT: ", linreg.intercept\_[0]) #Intercept

print("\nCOEFFICIENT: ", linreg.coef\_) #Coefficient

print("The linear model is: Y = {:.5} + {:.5}\*TV + {:.5}\*radio + {:.5}\*newspaper".format(linreg.intercept\_[0], linreg.coef\_[0][0], linreg.coef\_[0][1], linreg.coef\_[0][2]))

# Make predictions using the testing set

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

y=linreg.predict(np.array([275,55.7,80.6]).reshape(1,-1)) #Prediction

print("\nPredicted Value for the SALES for given instance: ", y)

#Accuracy Score

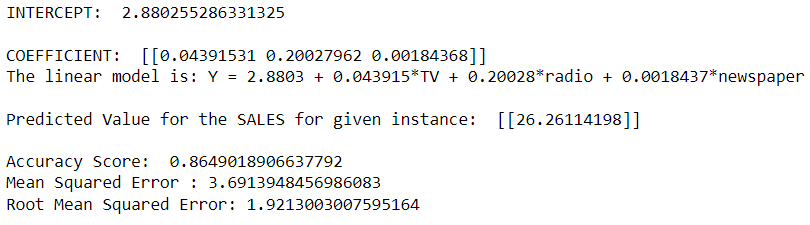
print("\nAccuracy Score: ", linreg.score(x\_test,y\_test))

predictions = linreg.predict(x\_test)

print('Mean Squared Error :', metrics.mean\_squared\_error(y\_test,predictions))

print('Root Mean Squared Error:', np.sqrt(metrics.mean\_squared\_error(y\_test,predictions)))

**OUTPUT:**



1. **Implement a classification/ logistic regression problem. For example, based on different features of “diabetes” data, classify, whether a woman is diabetic or not.**

**CODE:**

import pandas as pd

# split X and y into training and testing sets

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

# import the class

from sklearn.linear\_model import LogisticRegression

from sklearn import metrics

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

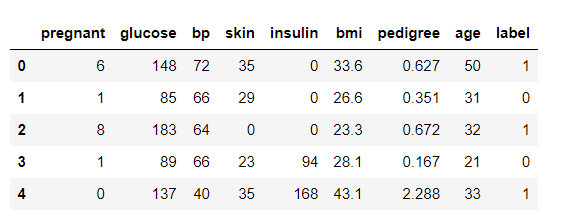
col\_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']

# load dataset

pima = pd.read\_csv("diabetes.csv", header= None, names=col\_names)

pima.head()

**OUTPUT:**



**CODE:**

#split dataset in features and target variable

feature\_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']

x = pima[feature\_cols] # Features

y = pima.label # Target variable

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

print(x\_train)

logreg = LogisticRegression()

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

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

print("\ny\_pred: ", y\_pred)

#Accuracy Score

print("\nAccuracy Score: ", logreg.score(x\_test,y\_test))

mse = metrics.mean\_squared\_error(y\_test, logreg.predict(x\_test))

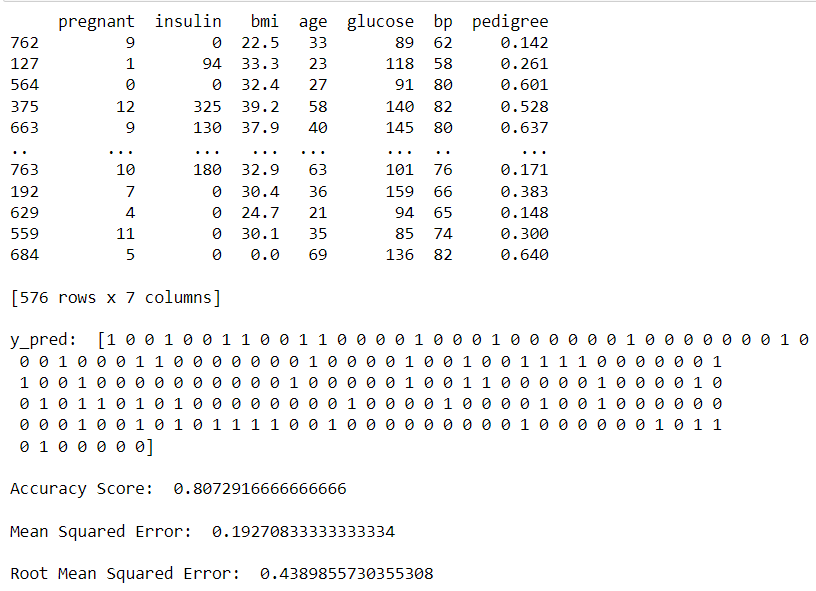
#Mean Square Error

print("\nMean Squared Error: ",mse)

#Root Mean Square Error

print("\nRoot Mean Squared Error: ",np.sqrt(mse))

**OUTPUT:**



**CODE:**

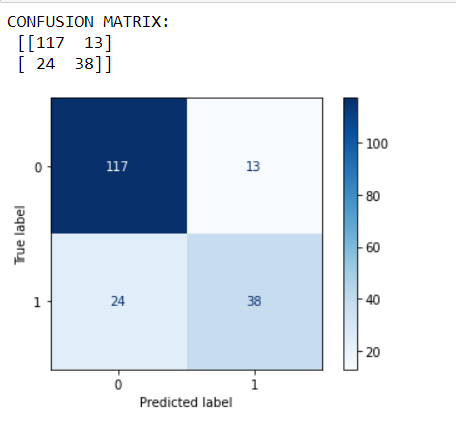
cnf\_matrix = metrics.confusion\_matrix(y\_test, y\_pred)

print("CONFUSION MATRIX:\n", cnf\_matrix)

metrics.plot\_confusion\_matrix(logreg, x\_test, y\_test, cmap=plt.cm.Blues)

plt.show()

**OUTPUT:**



**CODE:**

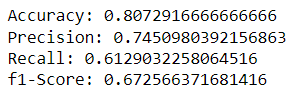
print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))

print("Precision:",metrics.precision\_score(y\_test, y\_pred))

print("Recall:",metrics.recall\_score(y\_test, y\_pred))

print("f1-Score:",metrics.f1\_score(y\_test,y\_pred))

**OUTPUT:**



1. **Use some function for regularization of “BOSTON” dataset available in ‘sklearn library’.**

**CODE:**

from sklearn.datasets import load\_boston

import pandas as pd

# split X and y into training and testing sets

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

# import the class

from sklearn.linear\_model import RidgeCV, LassoCV

from sklearn import metrics

import numpy as np

boston\_dataset = load\_boston()

boston = pd.DataFrame(boston\_dataset.data, columns=boston\_dataset.feature\_names)

boston['MEDV'] = boston\_dataset.target

boston.head()

x = boston.drop(['MEDV'], axis=1)

y = boston['MEDV']

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

alpha\_range=[0.00001, 0.01, 0.05, 0.1, 0.5, 1, 1.5, 3, 5, 6, 7, 8, 9, 10]

ridgecv=RidgeCV(alphas=alpha\_range, normalize=True, scoring='neg\_mean\_squared\_error')

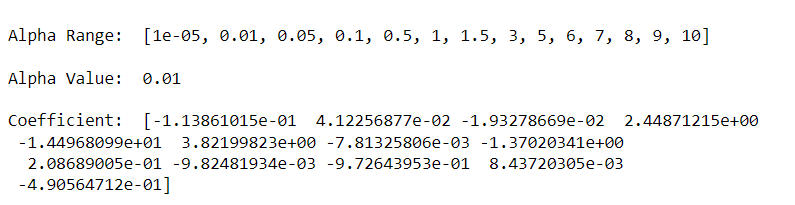
ridgecv.fit(x\_train,y\_train)

print("\nAlpha Range: ", alpha\_range)

print("\nAlpha Value: ", ridgecv.alpha\_)

print("\nCoefficient: ", ridgecv.coef\_)

**OUTPUT:**



**CODE:**

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

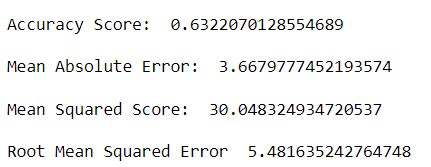
print("\nAccuracy Score: ", ridgecv.score(x\_test,y\_test))

print("\nMean Absolute Error: ", metrics.mean\_absolute\_error(y\_test,y\_pred))

print("\nMean Squared Score: ", metrics.mean\_squared\_error(y\_test,y\_pred))

print("\nRoot Mean Squared Error ", np.sqrt(metrics.mean\_squared\_error(y\_test,y\_pred)))

**OUTPUT:**



**CODE:**

#Lasso

lambda\_values = [0.000001, 0.0001, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5]

lassocv=LassoCV(alphas=alpha\_range, normalize=True)

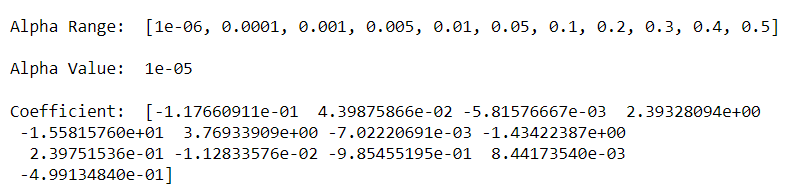
lassocv.fit(x\_train,y\_train)

print("\nAlpha Range: ", lambda\_values)

print("\nAlpha Value: ", lassocv.alpha\_)

print("\nCoefficient: ", lassocv.coef\_)

**OUTPUT:**



**CODE:**

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

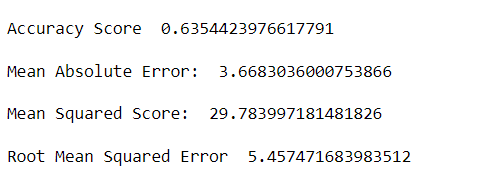
print("\nAccuracy Score ", lassocv.score(x\_test,y\_test))

print("\nMean Absolute Error: ", metrics.mean\_absolute\_error(y\_test,y\_pred))

print("\nMean Squared Score: ", metrics.mean\_squared\_error(y\_test,y\_pred))

print("\nRoot Mean Squared Error ", np.sqrt(metrics.mean\_squared\_error(y\_test,y\_pred)))

**OUTPUT:**



1. **Use some function for neural networks, like Stochastic Gradient Descent or backpropagation - algorithm to predict the value of a variable based on the dataset of problem 14.**

**CODE:**

import pandas as pd

# split X and y into training and testing sets

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

# import the class

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import plot\_confusion\_matrix

from sklearn import metrics

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

col\_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']

# load dataset

pima = pd.read\_csv("diabetes.csv", header= None, names=col\_names)

#split dataset in features and target variable

feature\_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']

x = pima[feature\_cols] # Features

y = pima.label # Target variable

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

# Create model object

mlp = MLPClassifier(hidden\_layer\_sizes=(10,10,10),

random\_state=5,

verbose=True,

solver='sgd',

learning\_rate\_init=0.001)

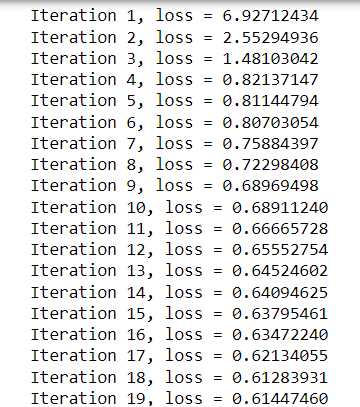
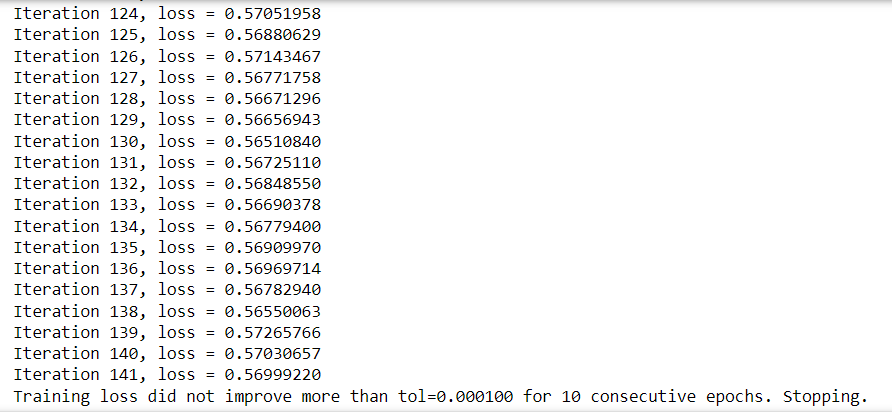
# Fit data onto the model

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

# Make prediction on test dataset

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

**OUTPUT:**

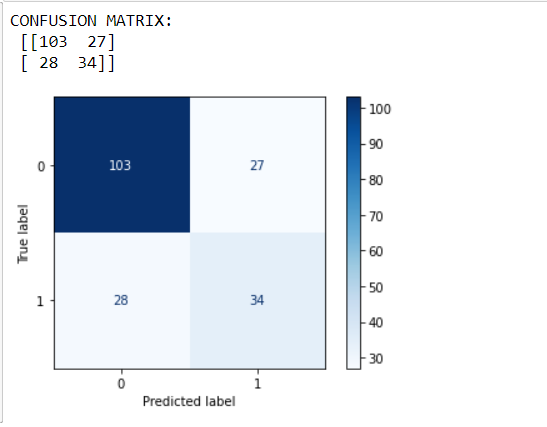
**CODE:**

cnf\_matrix = metrics.confusion\_matrix(y\_test, y\_pred)

print("CONFUSION MATRIX:\n", cnf\_matrix)

plot\_confusion\_matrix(mlp, x\_test, y\_test, cmap=plt.cm.Blues) plt.show()

**OUTPUT:**



**CODE:**

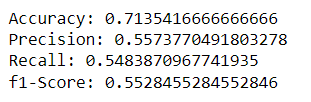
print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))

print("Precision:",metrics.precision\_score(y\_test, y\_pred))

print("Recall:",metrics.recall\_score(y\_test, y\_pred))

print("f1-Score:",metrics.f1\_score(y\_test,y\_pred))

**OUTPUT:**

****

1. **Implement Simple Linear Regression on “Advertising” dataset using Analytical Method.**

**CODE:**

import numpy as np

import pandas as pd

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)

# observations / data

adv = pd.read\_csv("Advertising.csv")

print(adv.head())

x = adv['TV']

y = adv['sales']

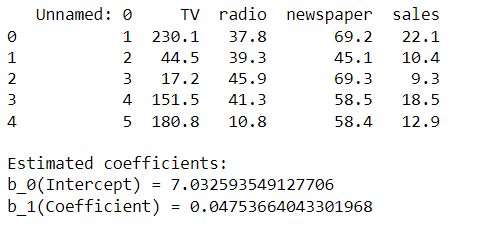
# estimating coefficients

b = estimate\_coef(x, y)

print("\nEstimated coefficients:\nb\_0(Intercept) = {} \

\nb\_1(Coefficient) = {}".format(b[0], b[1]))

**OUTPUT:**



**CODE:**

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

y\_bar=np.mean(y)

ssr=np.sum((y-y\_pred)\*(y-y\_pred))

sst=np.sum((y-y\_bar)\*(y-y\_bar))

acc=1-(ssr/sst)

print("Accuracy Score: ", acc)

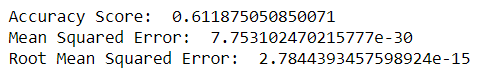
var=np.mean(y-y\_pred)

MSE=var\*var

print("Mean Squared Error: ", MSE)

print("Root Mean Squared Error: ", np.sqrt(MSE))

**OUTPUT:**



**CODE:**

#Plot Regression Line

# plotting the actual points as scatter plot

plt.scatter(x, y, color = "r", s = 30)

# predicted response vector

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

# plotting the regression line

plt.plot(x, y\_pred, color = "black")

# putting labels

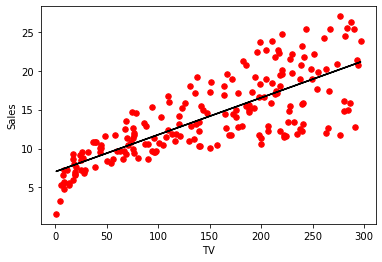
plt.xlabel('TV')

plt.ylabel('Sales')

# function to show plot

plt.show()

**OUTPUT:**



1. **Implement Multiple Linear Regression on “Advertising” dataset using Normal Equation Method.**

**CODE:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

adv = pd.read\_csv("Advertising.csv")

print(adv.head())

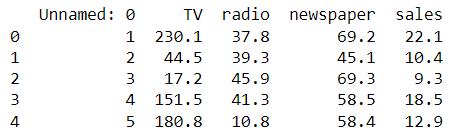
x1=adv['TV']

x2=adv['radio']

x3=adv['newspaper']

y=adv['sales']

**OUTPUT:**



**CODE:**

x1=np.array(x1)

x2=np.array(x2)

x3=np.array(x3)

y=np.array(y)

n=len(x1)

x\_bias=np.ones((n,1))

x1\_n=np.reshape(x1,(n,1))

x2\_n=np.reshape(x2,(n,1))

x3\_n=np.reshape(x3,(n,1))

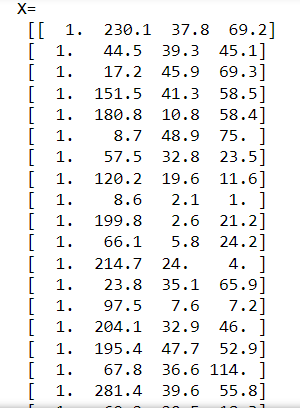
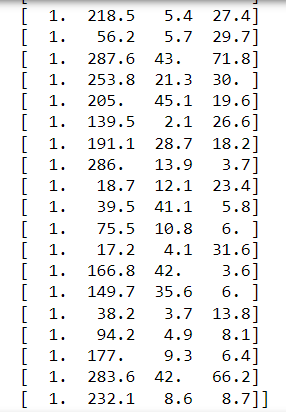
x=np.append(x\_bias,x1\_n,axis=1)

x=np.append(x,x2\_n,axis=1)

x=np.append(x,x3\_n,axis=1)

print('X= \n', x)

**OUTPUT:**

**CODE:**

x\_trans=np.transpose(x)

x\_trans\_dot\_x=x\_trans.dot(x)

temp1=np.linalg.inv(x\_trans\_dot\_x)

temp2=x\_trans.dot(y)

theta=temp1.dot(temp2)

b0=theta[0]

b1=theta[1]

b2=theta[2]

b3=theta[3]

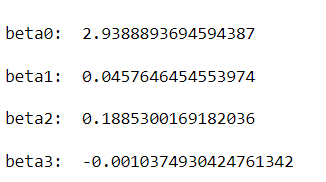
print("\nbeta0: ",b0)

print("\nbeta1: ",b1)

print("\nbeta2: ",b2)

print("\nbeta3: ",b3)

**OUTPUT:**



**CODE:**

y\_pred=b0+x1\*b1+x2\*b2+x3\*b3

y\_bar=np.mean(y)

ssr=np.sum((y-y\_pred)\*(y-y\_pred))

sst=np.sum((y-y\_bar)\*(y-y\_bar))

acc=1-(ssr/sst)

print("Accuracy Score: ", acc)

var=np.mean(y-y\_pred)

MSE=var\*var

print("Mean Squared Error: ", MSE)

print("Root Mean Squared Error: ", np.sqrt(MSE))

**OUTPUT:**

