**LIST OF 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, additing/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.

Q1) Perform elementary mathematical operations in Octave/MATLAB/R like addition, multiplication, division and exponentiation.

**CODE-**

def main():

ans="Y"

while(ans=="y" or ans=="Y"):

print("\n\t-\_-\_-\_-\_-\_-\_-\_-\_-\_-MENU-\_-\_-\_-\_-\_-\_-\_-\_-\_-\n")

print("1)Addition\n2)Subtraction\n3)Multiplication\n4)Division\n5)Exponentiation")

ch=int(input("\n\tEnter an option: "))

if(ch==1):

x=float(input("Enter Operand 1: "))

y=float(input("Enter Operand 2: "))

print("Sum =",x+y)

elif(ch==2):

x=float(input("Enter Operand 1: "))

y=float(input("Enter Operand 2: "))

print("Difference =",x-y)

elif(ch==3):

x=float(input("Enter Operand 1: "))

y=float(input("Enter Operand 2: "))

print("Product =",x\*y)

elif(ch==4):

x=float(input("Enter Dividend: "))

y=float(input("Enter Divisor: "))

while(y==0):

print("Error: Divisor can't be zero! Re-enter value")

y=float(input("Enter Divisor: "))

print("Quotient =",x//y,"Remainder =",x%y)

elif(ch==5):

x=float(input("Enter base: "))

y=float(input("Enter power: "))

print("Result =",x\*\*y)

else:

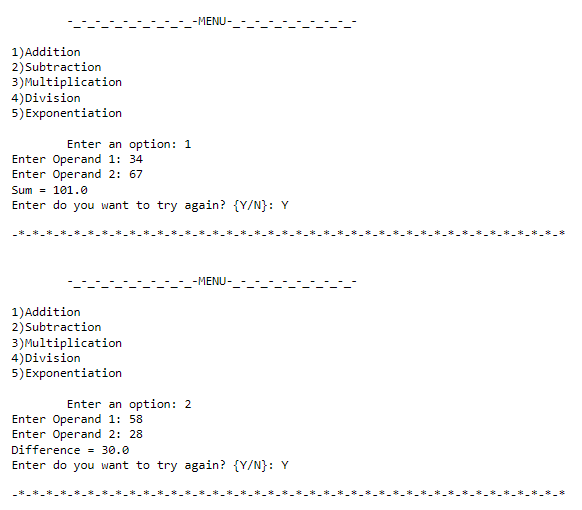
print("Enter valid option")

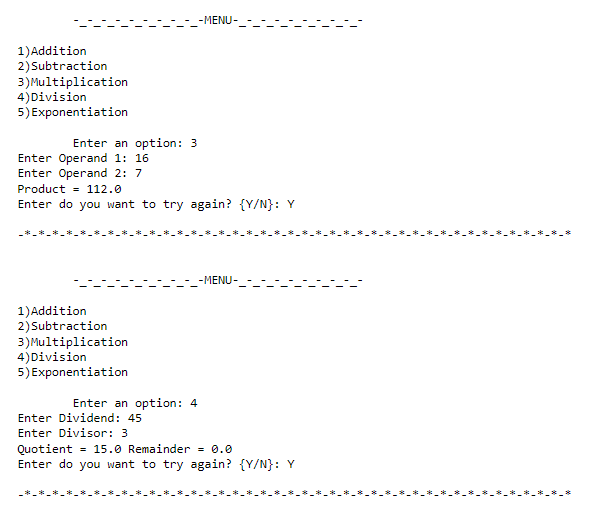
ans=input("Enter do you want to try again? {Y/N}: ")

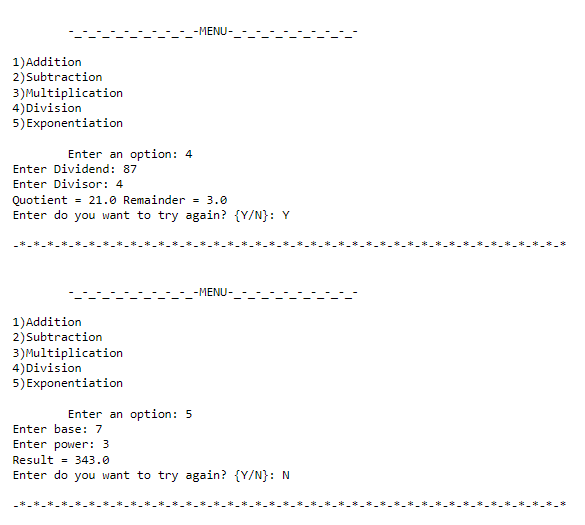
print("\n-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*\n")

main()

**OUTPUT-**







Q2) Perform elementary logical operations in Octave/MATLAB/R (like OR, AND, Checking for Equality, NOT, XOR).

**CODE-**

def main():

ans="Y"

while(ans=="y" or ans=="Y"):

print("\n\t-\_-\_-\_-\_-\_-\_-\_-\_-\_-MENU-\_-\_-\_-\_-\_-\_-\_-\_-\_-\n")

print("1)OR\n2)AND\n3)Check for Equality\n4)NOT\n5)XOR")

ch=int(input("\n\tEnter an option: "))

x=int(input("\nEnter a number(x): "))

if(ch==1):

print(" x < 4 OR x > 10: ",(x<4 or x>10))

elif(ch==2):

print(" x > 3 AND x < 10: ",(x>3 and x<10))

elif(ch==3):

print("x==5: ",x==5)

elif(ch==4):

print("NOT x: ",not x)

elif(ch==5):

print("2 XOR x: ",2^x)

else:

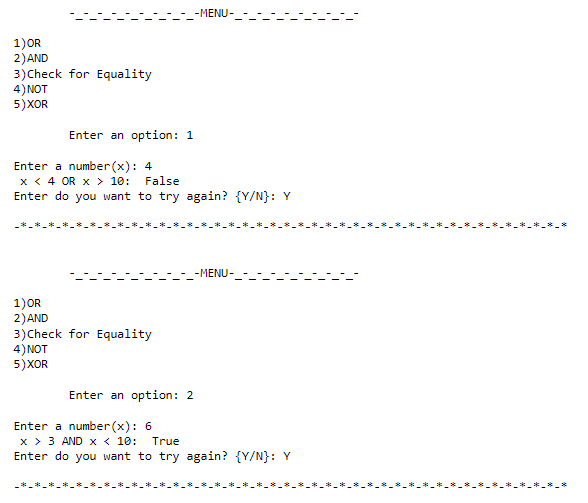
print("Enter valid option")

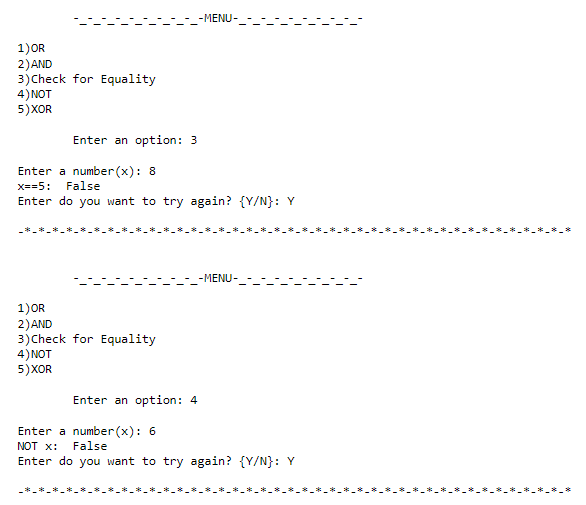
ans=input("Enter do you want to try again? {Y/N}: ")

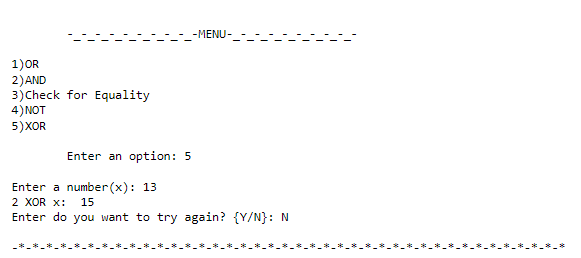
print("\n-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*\n")

main()

**OUTPUT-**







Q3) Create, initialize and display simple variables and simple strings and use simple formatting for variable.

**CODE-**

x="206"

y="28"

z="8"

print("\nx: ",x,"\ny: ",y,"\nz: ",z)

print("\n-\*-\*-\*-Using Formatters-\*-\*-\*-\n")

print("There are {} bones in the body.".format(x))

print("India has {} states and {} union territories in total.".format(y,z))

x= 4/7

y=15

z= """ The sun rises in the EAST.

And sets in the WEST."""

print("\nx: ",x,"\ny: ",y,"\nz: ",z)

print("\n-\*-\*-\*-String Formatting-\*-\*-\*-\n")

print("Applying split:\n",z.split())

print("\nApplying splitlines:\n",z.splitlines())

print("\nApplying strip:\n",z.strip())

print("\nApplying uppercase:\n",z.upper())

print("\nApplying lowercase:\n",z.lower())

print("\nApplying Capitalization:\n",z.capitalize())

print("\nApplying title() function:\n",z.title())

print("\nApplying join:\n","\_".join(z.split()))

print("\n-\*-\*-\*-Number Formatting-\*-\*-\*-\n")

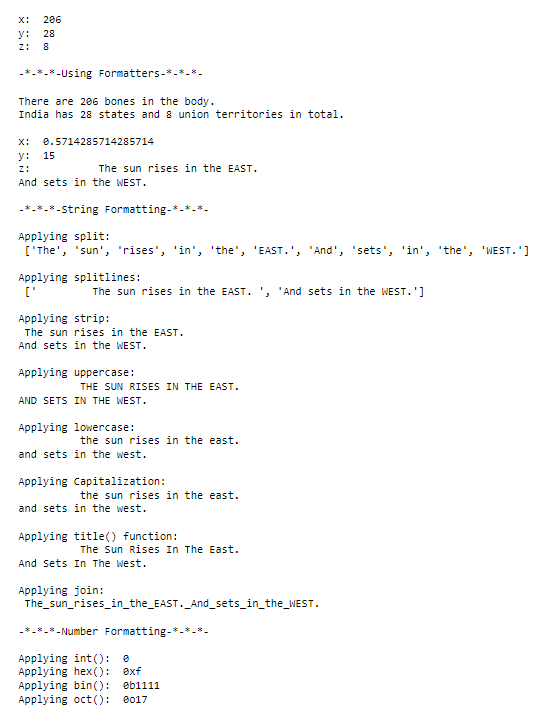
print("Applying int(): ",int(x))

print("Applying hex(): ",hex(y))

print("Applying bin(): ",bin(y))

print("Applying oct(): ",oct(y))

**OUTPUT-**



Q4) 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

def main():

print("\n\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*")

print("\n\t1D ARRAY OF ALL 1s\n")

s1=int(input("Enter size for 1D Array of all 1s: "))

a1= np.array([])

for i in range(s1):

a1=np.append(a1,1)

print("\nOUTPUT:\n",a1)

print("\n\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*")

print("\n\tZERO MATRIX\n")

s2=int(input("\nEnter number of rows & columns for matrix: "))

a2= np.zeros(shape=(s2,s2))

print("\nOUTPUT:\n",a2)

print("\n\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*")

print("\n\tDIAGONAL MATRIX\n")

s3=int(input("\nEnter number of rows & columns for matrix: "))

p=int(input("Enter an integer: "))

a3= np.zeros(shape=(s3,s3))

for i in range(s3):

for j in range(s3):

if(i==j):

a3[i][j]=p

print("\nOUTPUT:\n",a3)

print("\n\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*")

print("\n\t3D ARRAY WITH USER INPUT VALUES\n")

print("Enter 3 integers for size of 3D array:")

i1=int(input("\nEnter i1: "))

i2=int(input("\nEnter i2: "))

i3=int(input("\nEnter i3: "))

a4= np.zeros(shape=(i1,i2,i3))

print()

for i in range(i1):

for j in range(i2):

for k in range(i3):

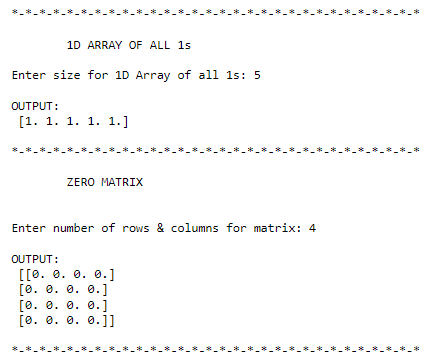
a4[i][j][k]=int(input("Enter element: "))

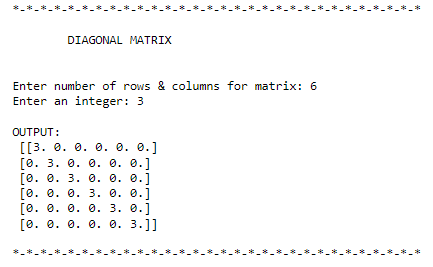
print("\nOUTPUT:\n",a4)

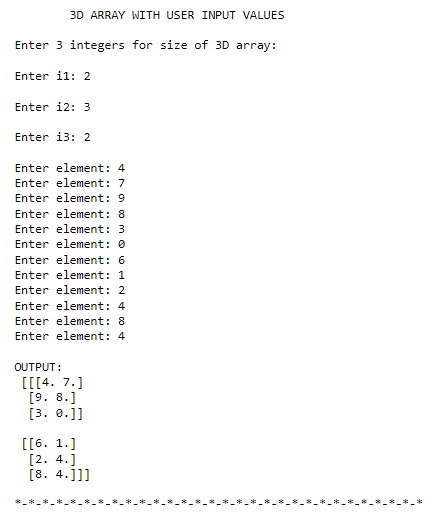
print("\n\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*")

main()

**OUTPUT-**







Q5) 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

def main():

a=np.loadtxt("input.txt")

print("\nMatrix loaded from file 'input.txt':\n",a)

print("\nSize of matrix:", np.size(a))

print("\nDimension of matrix:", np.ndim(a))

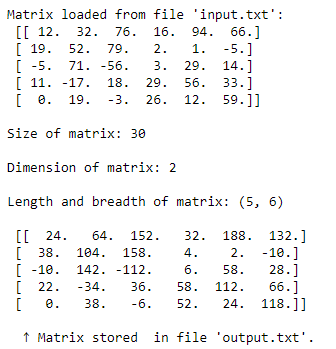
print("\nLength and breadth of matrix:", np.shape(a))

np.savetxt("output.txt", a\*2)

print("\n",a\*2,"\n\n ↑ Matrix stored in file 'output.txt'.")

main()

**OUTPUT-**



Q6) Perform basic operations on matrices (like addition, subtraction, multiplication) and display specific rows or columns of the matrix.

**CODE-**

import numpy as np

def main():

print("\n\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*")

print("\n\tADDITION, SUBTRACTION AND MULTIPLICATION OF MATRICES\n")

row=int(input("Enter number of rows in both matrix: "))

col=int(input("Enter number of columns in both matrix: "))

a=np.zeros((row,col))

b=np.zeros((row,col))

print("Enter elements of 1st matrix: ")

for i in range (row):

for j in range (col):

a[i,j]=int(input("Enter element: "))

print("Enter elements of 2nd matrix: ")

for i in range (row):

for j in range (col):

b[i,j]=int(input("Enter element: "))

print("Matrix 1:\n",a)

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

print("\nSum:\n",a+b)

print("\nDifference:\n",a-b)

print("\n Product of corresponding elements:\n",a\*b)

print("\n\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*")

print("\n\tMULTIPLICATION OF MATRICES\n")

r1=int(input("Enter number of rows in 1st matrix: "))

c1=int(input("Enter number of columns in 1st matrix and rows in 2nd matrix: "))

r2=c1

c2=int(input("Enter number of columns in 2nd matrix: "))

a=np.zeros((r1,c1))

b=np.zeros((r2,c2))

p=np.zeros((r1,c2))

print("Enter elements of 1st matrix: ")

for i in range (r1):

for j in range (c1):

a[i,j]=int(input("Enter element: "))

print("Enter elements of 2nd matrix: ")

for i in range (r2):

for j in range (c2):

b[i,j]=int(input("Enter element: "))

for i in range(r1):

for j in range(c2):

for k in range(c1):

p[i,j] += a[i,k]\*b[k,j]

print("Matrix 1:\n",a)

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

print("\nProduct:\n",p)

print("\n\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*")

print("\n\tDISPLAY CERTAIN ROWS OR COLUMNS OF GIVEN MATRIX\n")

given=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16],[17,18,19,20],[21,22,23,24]])

print("Given matrix:\n",given)

p=int(input("\nEnter row to be displayed: "))

print("OUTPUT:",given[p,:])

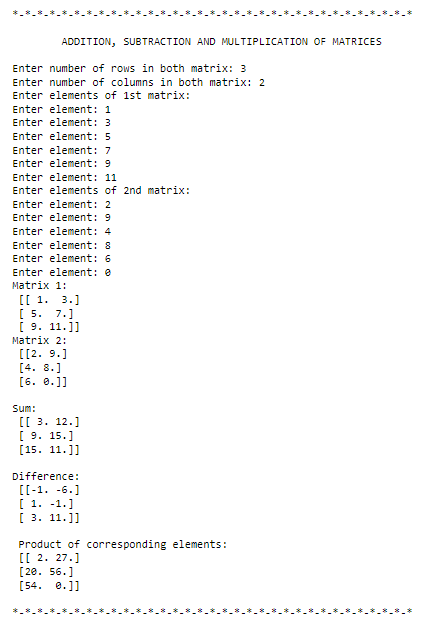
p=int(input("\nEnter column to be displayed: "))

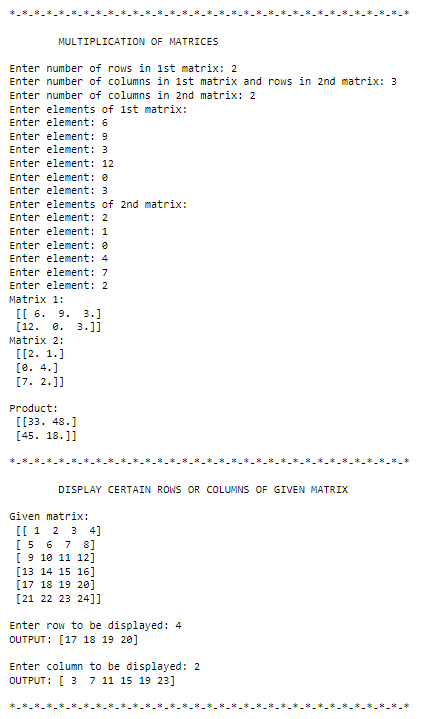
print("OUTPUT:",given[:,p])

print("\n\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*")

main()

**OUTPUT-**





Q7) 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

def main():

print("\n\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*")

print("\n\tOPERATIONS ON MATRICES\n")

row=int(input("Enter number of rows in matrix: "))

col=int(input("Enter number of columns in matrix: "))

a=np.zeros((row,col))

print("Enter elements of matrix: ")

for i in range (row):

for j in range (col):

a[i,j]=int(input("Enter element: "))

print("Matrix:\n",a)

print("\nAbsolute Matrix:\n",np.abs(a))

print("\nNegative Matrix:\n",np.negative(a))

print("\nElement of greatest value in matrix: ",np.max(a))

print("Element of least value in matrix: ",np.min(a))

print("Greatest element in each row: ",np.max(a,axis=1))

print("Least element in each row: ",np.min(a,axis=1))

print("Greatest element in each column: ",np.max(a,axis=0))

print("Least element in each column: ",np.min(a,axis=0))

print("\nSum of elements in each row: ",np.sum(a,axis=1))

print("Sum of elements in each column: ",np.sum(a,axis=0))

print("\nSum of all elements: ",np.sum(a))

print("\n\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*")

del\_row=int(input("\nEnter row index to be deleted: "))

print("OUTPUT:\n",np.delete(a,del\_row,0))

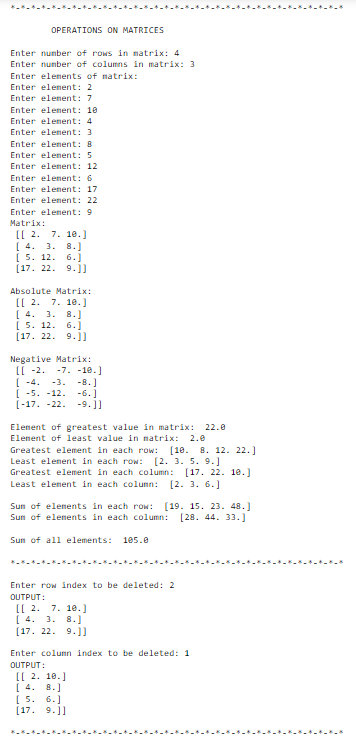
del\_col=int(input("\nEnter column index to be deleted: "))

print("OUTPUT:\n",np.delete(a,del\_col,1))

print("\n\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*")

main()

**OUTPUT-**



Q8) 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

// PLOTTING SINE AND COSINE FUNCTION

rng=int((4\*np.pi-1)/0.1)+1

data=np.zeros(shape=(3,rng))

for i in range(rng):

data[0][i]=i\*0.1

data[1][i]=np.sin(data[0][i])

data[2][i]=np.cos(data[0][i])

plt.plot(data[0],data[1],data[0],data[2])

plt.xlabel('x values from 0 to 4pi')

plt.ylabel('sin(x) and cos(x)')

plt.title('Plot of sin and cos from 0 to 4pi')

plt.legend(['sin(x)', 'cos(x)'])

plt.show()

// PLOTTING A HISTOGRAM

x = np.random.normal(50,10, 600)

plt.hist(x)

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.title('Histogram plot')

plt.show()

// PLOTTING A PIE CHART WITH NEWSPAPER DATA

newspapers=[["Dainik Bhaskar","Dainik Jagran","The Times of India","Malayala Manorama","Amar Ujala","Hindustan Dainik","Rajasthan Patrika", "Eenadu","Dina Thanthi","The Hindu"],[4579051,3614162,2880144,2308612,2261990,2221566,1788420,1614105,1472948,1415792]]

plt.pie(newspapers[1],labels=newspapers[0],radius=0.8)

plt.title('Top 10 newspapers in India based on sale in 2019')

plt.show()

plt.pie(newspapers[1],labels=newspapers[0],radius=0.8,explode=(0,0,1,0,0,0,0,0,0,1),shadow=True)

plt.title('Top 10 newspapers in India based on sale in 2019 (Highlighting English newspapers)')

plt.show()

// PLOTTING A BAR GRAPH WITH MARKS OF 10 STUDENTS

marks=[np.arange(10),np.random.randint(0,100,10),np.random.randint(0,100,10),np.random.randint(0,100,10)]

plt.bar(marks[0]+0.75,marks[1],0.25,color="orange",label="Theory of Computation")

plt.bar(marks[0]+1,marks[2],0.25,color="blue",label="Operating System")

plt.bar(marks[0]+1.25,marks[3],0.25,color="green",label="Microprocessors")

plt.yticks(range(0,160,20))

plt.title("Marks Semester VIII")

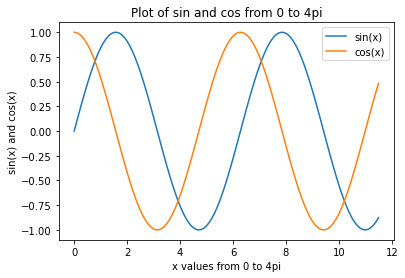
plt.xlabel('Roll Number')

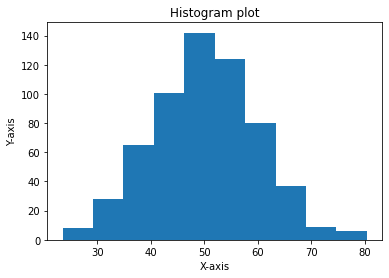
plt.ylabel('Marks')

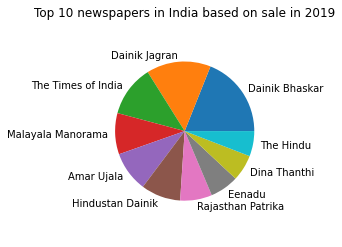
plt.legend()

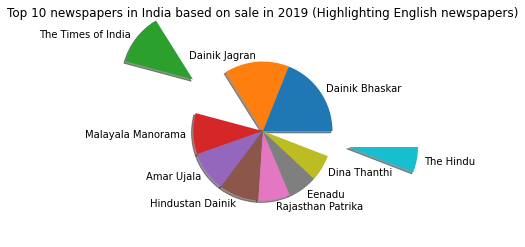
plt.show()

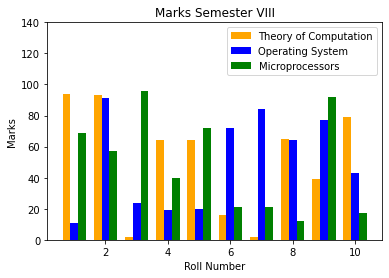
**OUTPUT-**

****

****

****

****

****

Q9) Generate different subplots from a given plot and color plot data.

**CODE-**

import matplotlib.pyplot as plt

import numpy as np

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

plt.subplot(221)

rng=int((10\*np.pi-1)/0.1)+1

data=np.zeros(shape=(2,rng))

for i in range(rng):

data[0][i]=i\*0.1

data[1][i]=np.tan(data[0][i])

plt.plot(data[0],data[1])

plt.xlabel('x values from 0 to 10pi')

plt.ylabel('tan(x)')

plt.title('Plot of tan from 0 to 10pi')

plt.legend(['tan(x)'])

plt.subplot(222)

x = np.random.normal(25,5,100)

plt.hist(x,color="orange")

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.title('Histogram plot')

plt.subplot(223)

newspapers=[["JavaScript","HTML/CSS","Python","SQL","Java","TypeScript","C#"],[64.96,56.07,48.24,47.08,35.35,30.19,27.86]]

plt.pie(newspapers[1],labels=newspapers[0],radius=1)

plt.title('Most used programming languages among developers worldwide, as of 2021')

plt.subplot(224)

medals=[['United States','China','Japan','Great Britain','ROC','Australia'],[113,88,58,65,71,46]]

plt.bar(medals[0],medals[1],0.25,color="pink",label="Medals")

plt.yticks(range(0,160,20))

plt.title("Summer Olympics 2020 Medals tally")

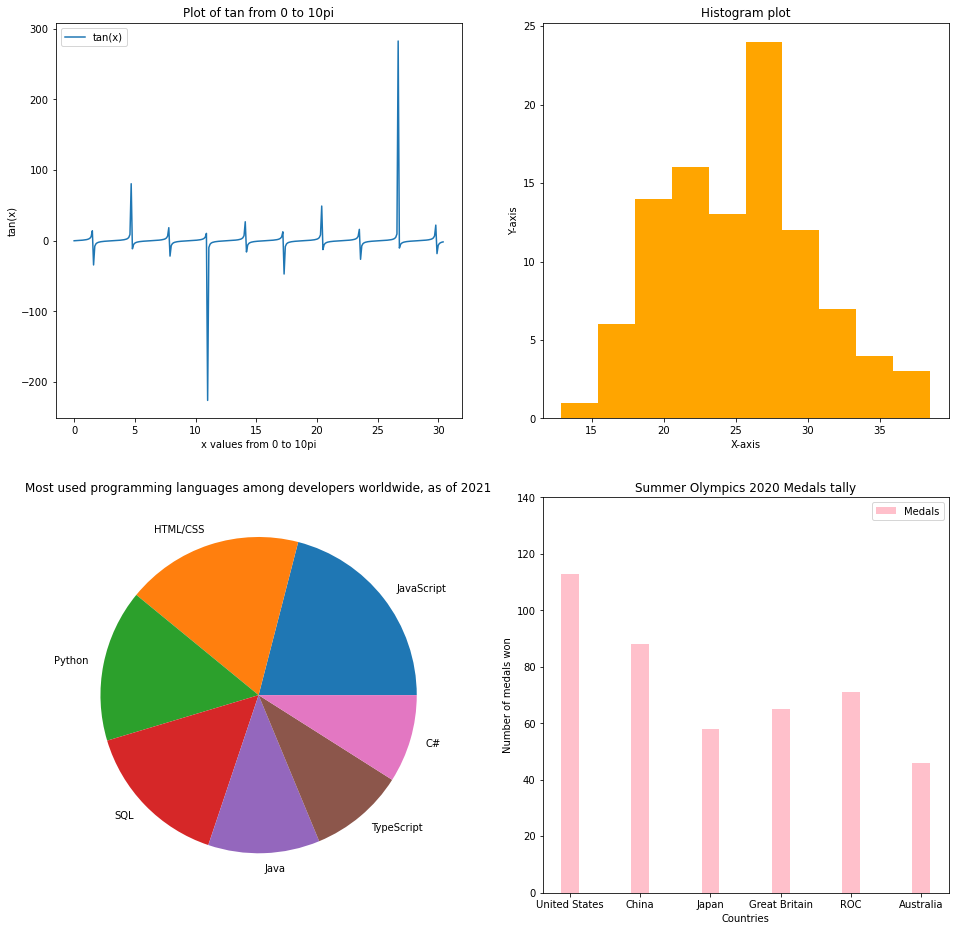
plt.xlabel('Countries')

plt.ylabel('Number of medals won')

plt.legend()

plt.show()

**OUTPUT-**



Q10) Use conditional statements and different type of loops based on simple example/s.

**CODE-**

def main():

print("\n-\*-\*-\*-\*-\*-\*-\*-\*-\*- IMPLEMENTATION OF LOOPS AND CONDITIONAL STATEMENTS TO PERFORM MATRIX MULTIPLICATION -\*-\*-\*-\*-\*-\*-\*-\*-\*-\n")

r1=int(input("Enter the number of rows in 1st matrix: "))

c1=int(input("Enter the number of columns in 1st matrix: "))

r2=int(input("Enter the number of rows in 2nd matrix: "))

c2=int(input("Enter the number of columns in 2nd matrix: "))

a=[[0 for x in range(c1)] for y in range(r1)]

b=[[0 for x in range(c2)] for y in range(r2)]

p=[[0 for x in range(c2)] for y in range(r1)]

if((r1 and r2 and c1 and c2) > 0):

if(c1==r2):

print("Enter elements in 1st matrix:")

for i in range(r1):

print("Row",i+1,":",sep="")

for j in range(c1):

print("Column",j+1,":",sep="",end=" ")

a[i][j]=int(input())

print("Enter elements in 2nd matrix:")

for i in range(r2):

print("Row",i+1,":",sep="")

for j in range(c2):

print("Column",j+1,":",sep="",end=" ")

b[i][j]=int(input())

for i in range(len(a)):

for j in range(len(b[0])):

for k in range(len(b)):

p[i][j] += a[i][k]\*b[k][j]

print("Product: ")

for i in range(len(p)):

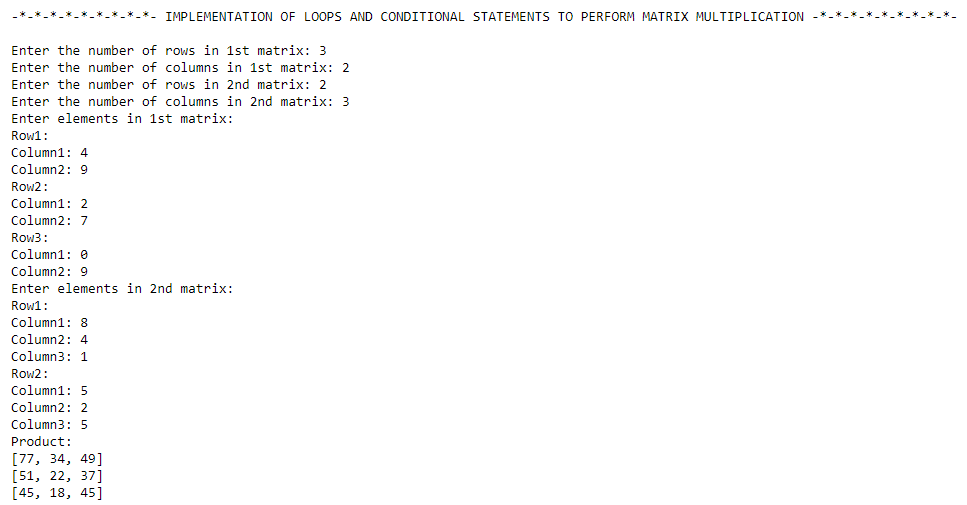
print(p[i])

else:

print("Matrices not compatible for multiplication!")

main()

**OUTPUT-**



Q11) 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

def main():

v1= np.array([[1,3,0],[9,1,8],[4,7,1]])

v2= np.array([[2,8,5],[3,5,0],[9,1,5]])

print("Matrix 1:\n",v1)

print("Matrix 2:\n",v2)

sum= np.add(v1,v2)

print("Sum: \n",sum)

result = [[0, 0, 0],[0, 0, 0],[0, 0, 0]]

for i in range(len(v1)):

for j in range(len(v2[0])):

for k in range(len(v2)):

result[i][j] += v1[i][k] \* v2[k][j]

print("\n Product:\n")

for r in result:

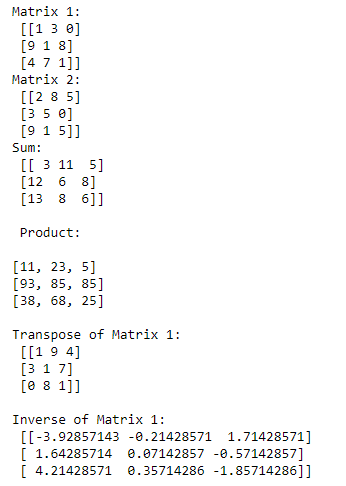
print(r)

print("\nTranspose of Matrix 1:\n",v1.T)

print("\nInverse of Matrix 1:\n",np.linalg.inv(v1))

main()

**OUTPUT-**



Q12) Implement Linear Regression problem. For example, based onthe “Advertising” dataset comprising of budget of TV, Radio etc. and the sales data, predict the estimated sales for TV budget.

**CODE AND CORRESPONDING OUTPUTS-**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

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

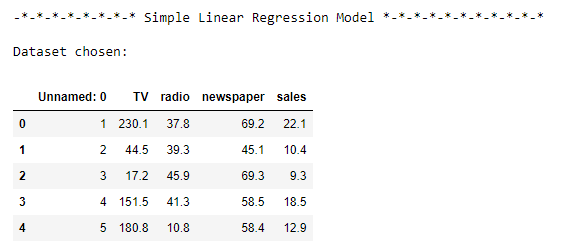
from sklearn import metrics

print("\n-\*-\*-\*-\*-\*-\*-\*-\* Simple Linear Regression Model \*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*\n")

df = pd.read\_csv('Advertising.csv')

print("Dataset chosen:")

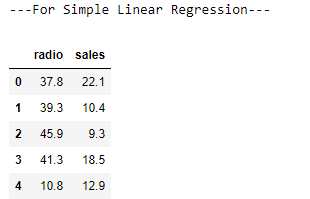
df.head()



print("---For Simple Linear Regression---")

df\_binary = df[['radio', 'sales']]

df\_binary.head()



df\_binary.isnull().sum()

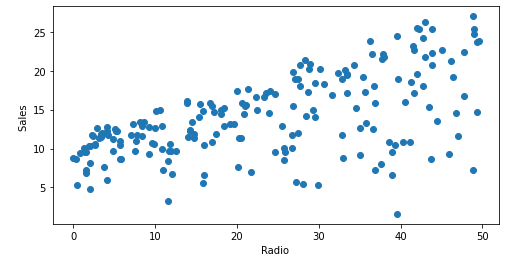
plt.figure(figsize=(8, 4))

plt.scatter(df\_binary['radio'], df\_binary['sales'])

plt.xlabel("Radio ")

plt.ylabel("Sales ")

plt.show()



x = np.array(df\_binary['radio']).reshape(-1, 1)

y = np.array(df\_binary['sales']).reshape(-1, 1)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.3,random\_state=4)

SLR = LinearRegression()

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

print('Intercept: ',round(SLR.intercept\_[0],4))

print('Coefficient: ',round(SLR.coef\_[0][0],5))

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

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

y=SLR.predict(np.array([1000]).reshape(1,-1))

print("\nPredicted Value for the sale of radios: ", y)

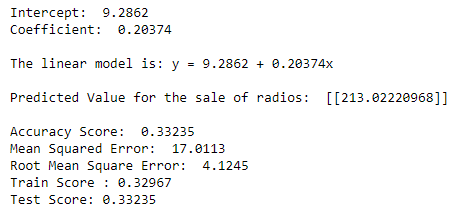
print("\nAccuracy Score: ", round(SLR.score(x\_test,y\_test),5))

print('Mean Squared Error: ', round(metrics.mean\_squared\_error(y\_test,y\_pred),4))

print('Root Mean Square Error: ', round(np.sqrt(metrics.mean\_squared\_error(y\_test,y\_pred)),4))

print('Train Score :', round(SLR.score(x\_train,y\_train),5))

print('Test Score:', round(SLR.score(x\_test,y\_test),5))

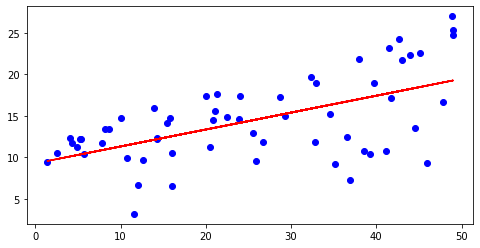


plt.figure(figsize=(8, 4))

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

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

plt.show()



errors = list()

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

E1 = (y\_test[i] - y\_pred[i])\*\*2

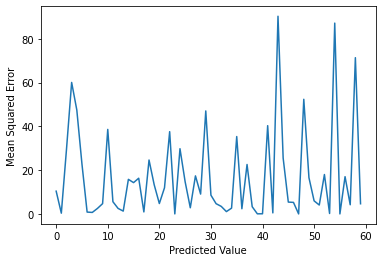
errors.append(E1)

plt.plot(errors)

plt.xlabel('Predicted Value')

plt.ylabel('Mean Squared Error')

plt.show()



Q13) Based on multiple features/variables perform Linear Regression. For example, based on a number of additional features like number of bedrooms, servant room, number of balconies, number of houses of years a house has been built – predict the price of a house.

**CODE AND CORRESPONDING OUTPUTS-**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

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

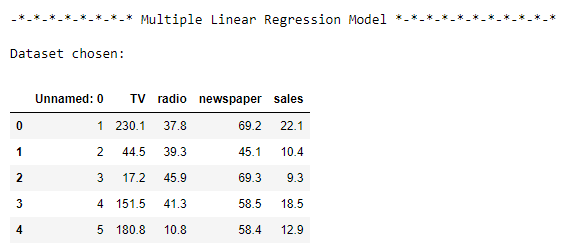
from sklearn import metrics

print("\n-\*-\*-\*-\*-\*-\*-\*-\* Multiple Linear Regression Model \*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*\n")

dataset = pd.read\_csv('Advertising.csv')

print("Dataset chosen:")

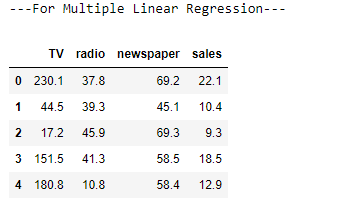
dataset.head()



print("---For Multiple Linear Regression---")

df = dataset[['TV','radio','newspaper', 'sales']]

df.head()



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

y = np.array(df['sales']).reshape(-1, 1)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.3,random\_state=4)

MLR = LinearRegression()

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

print('Intercept: ',round(MLR.intercept\_[0],4))

print('Coefficient: ',round(MLR.coef\_[0][0],5))

print("\nThe linear model is: y = {:.5} + {:.5}\*x1 + {:.5}\*x2 + {:.5}\*x3".format(MLR.intercept\_[0], MLR.coef\_[0][0], MLR.coef\_[0][1], MLR.coef\_[0][2]))

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

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

print("\nPredicted Value for the sale of radios: ", y)

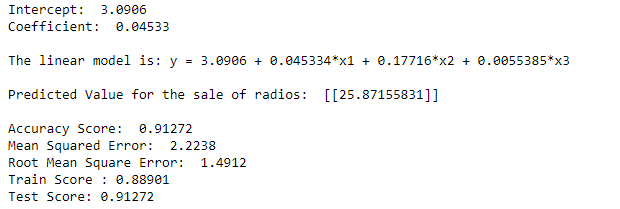
print("\nAccuracy Score: ", round(MLR.score(x\_test,y\_test),5))

print('Mean Squared Error: ', round(metrics.mean\_squared\_error(y\_test,y\_pred),4))

print('Root Mean Square Error: ', round(np.sqrt(metrics.mean\_squared\_error(y\_test,y\_pred)),4))

print('Train Score :', round(MLR.score(x\_train,y\_train),5))

print('Test Score:', round(MLR.score(x\_test,y\_test),5))



errors = list()

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

E1 = (y\_test[i] - y\_pred[i])\*\*2

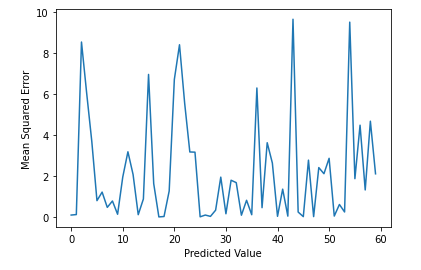
errors.append(E1)

plt.plot(errors)

plt.xlabel('Predicted Value')

plt.ylabel('Mean Squared Error')

plt.show()



Q14) Implement a classification/ logistic regression problem. For example, based on different features of “diabetes” data, classify, whether a woman is diabetic or not.

**CODE AND CORRESPONDING OUTPUTS-**

import pandas as pd

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

from sklearn.linear\_model import LogisticRegression

from sklearn import metrics

import numpy as np

import matplotlib.pyplot as plt

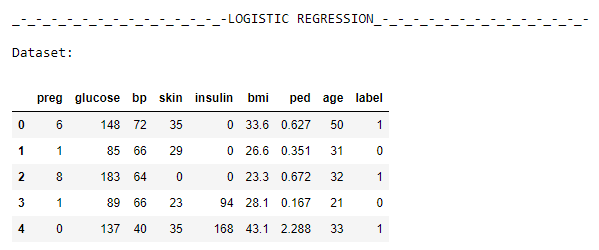
print("\n\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-LOGISTIC REGRESSION\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\n")

dataset = pd.read\_csv("diabetes.csv", header=None, names=['preg', 'glucose', 'bp', 'skin', 'insulin',

'bmi', 'ped', 'age', 'label'])

print("Dataset:")

dataset.head()



x = dataset[['preg', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'ped', 'age']]

y = dataset.label

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.4,random\_state=0)

logreg = LogisticRegression()

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

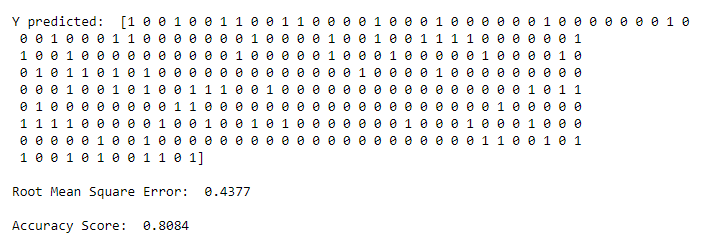
y\_pred=logreg.predict(x\_test)

print("\nY predicted: ", y\_pred)

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

print("\nRoot Mean Square Error: ",round(np.sqrt(mse),4))

print("\nAccuracy Score: ", round(logreg.score(x\_test,y\_test),4))



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)

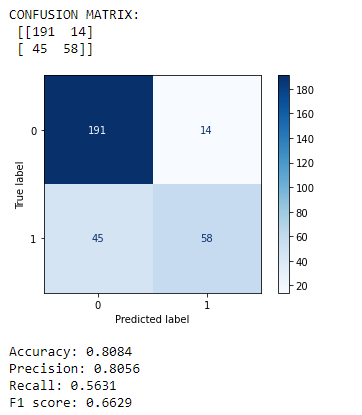
plt.show()

print("Accuracy:",round(metrics.accuracy\_score(y\_test, y\_pred),4))

print("Precision:",round(metrics.precision\_score(y\_test, y\_pred),4))

print("Recall:",round(metrics.recall\_score(y\_test, y\_pred),4))

print("F1 score:",round(metrics.f1\_score(y\_test, y\_pred),4))



Q15) Use some function for regularization of “BOSTON” dataset available in ‘sklearn library’.

**CODE AND CORRESPONDING OUTPUTS-**

from sklearn.datasets import load\_boston

import pandas as pd

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

from sklearn.linear\_model import RidgeCV, LassoCV

from sklearn import metrics

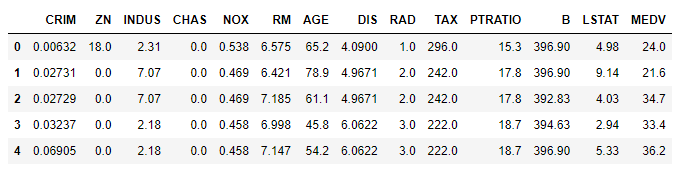
import numpy as np

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')

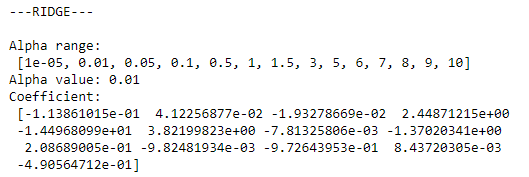
ridgecv.fit(x\_train,y\_train)

print("---RIDGE---\n")

print("Alpha range:\n",alpha\_range)

print("Alpha value:", ridgecv.alpha\_)

print("Coefficient:\n",ridgecv.coef\_)



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

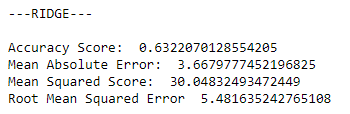
print("---RIDGE---\n")

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

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

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

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



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)

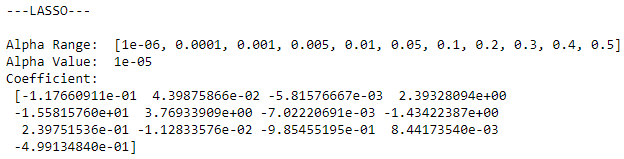
lassocv.fit(x\_train,y\_train)

print("---LASSO---\n")

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

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

print("Coefficient:\n", lassocv.coef\_)



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

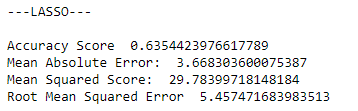
print("---LASSO---\n")

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

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

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

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



Q16) Use some function for neural networks, like Stochastic Gradient Descent or back propagation - algorithm to predict the value of a variable based on the dataset of problem 14.

**CODE AND CORRESPONDING OUTPUTS-**

import pandas as pd

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

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import plot\_confusion\_matrix

from sklearn import metrics

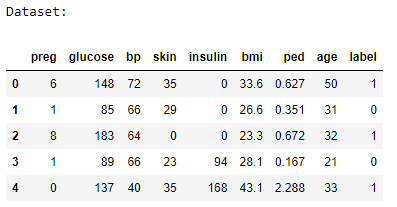
import numpy as np

import matplotlib.pyplot as plt

dataset = pd.read\_csv("diabetes.csv", header=None, names=['preg', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'ped', 'age', 'label'])

print("Dataset:")

dataset.head()



x = dataset[['preg', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'ped', 'age']]

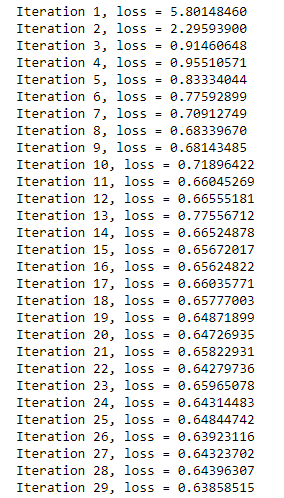
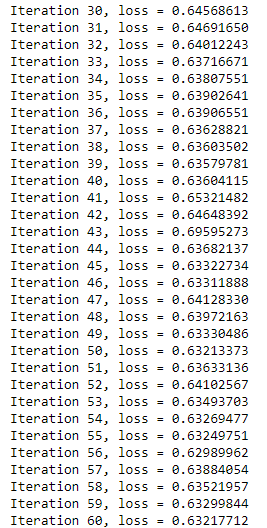
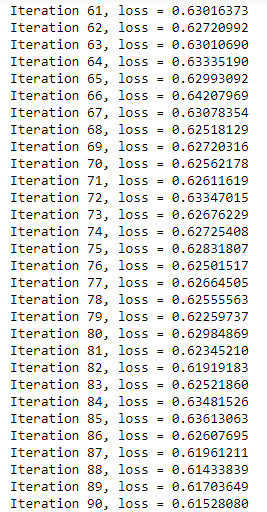
y = dataset.label

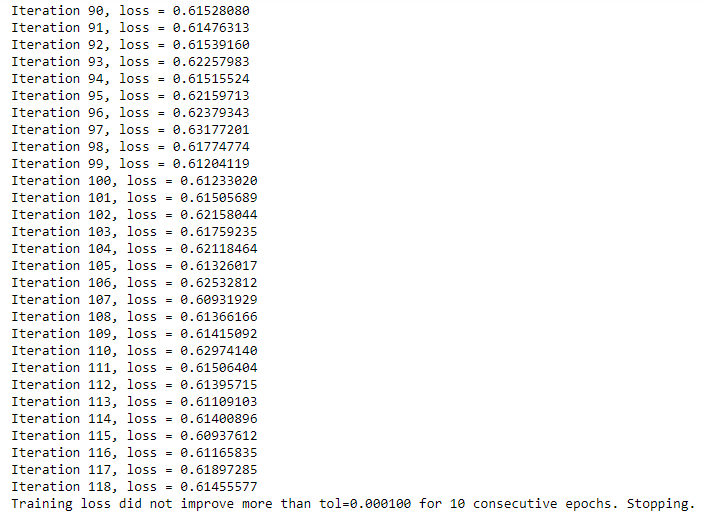
x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.4,random\_state=0)

mlp = MLPClassifier(hidden\_layer\_sizes=(10,10,10),random\_state=5,verbose=True,solver='sgd',learning\_rate\_init=0.001)

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

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



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)

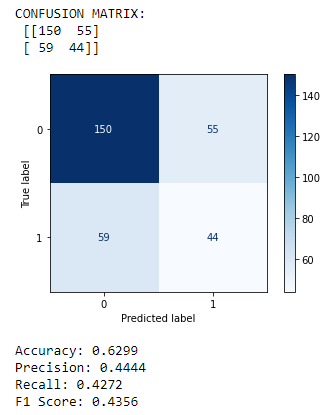
plt.show()

print("Accuracy:",round(metrics.accuracy\_score(y\_test, y\_pred),4))

print("Precision:",round(metrics.precision\_score(y\_test, y\_pred),4))

print("Recall:",round(metrics.recall\_score(y\_test, y\_pred),4))

print("F1 Score:",round(metrics.f1\_score(y\_test,y\_pred),4))



Q17) Implement Simple Linear Regression on “Advertising” dataset using Analytical Method.

**CODE AND CORRESPONDING OUTPUTS-**

import pandas as pd

import numpy as np

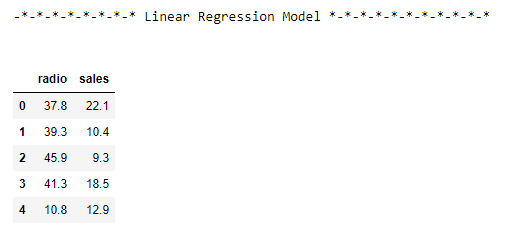
import matplotlib.pyplot as plt

df=pd.read\_csv('Advertising.csv')

print("\n-\*-\*-\*-\*-\*-\*-\*-\* Linear Regression Model \*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*\n")

df=df[['radio','sales']]

df.head()



x\_mean=np.mean(df['radio'])

y\_mean=np.mean(df['sales'])

print("\nx-mean:",round(x\_mean,4),"\ty-mean:",round(y\_mean,4))

df=pd.concat([df,pd.DataFrame((x-x\_mean for x in df['radio']),columns=['x-x\_mean']),pd.DataFrame((y-y\_mean for y in df['sales']),columns=['y-y\_mean'])],axis=1)

df=pd.concat([df,pd.DataFrame((x\*\*2 for x in df['x-x\_mean']),columns=['(x-x\_mean)^2'])],axis=1)

print("\nFor calculating slope (m):\n",df)

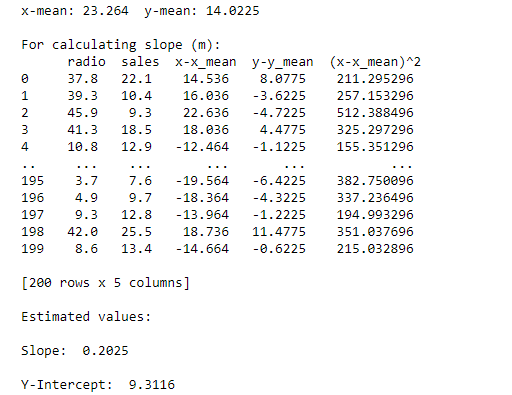
m=np.sum(df['x-x\_mean']\*df['y-y\_mean'])/np.sum(df['(x-x\_mean)^2'])

print("\nEstimated values:")

print("\nSlope: ",round(m,4))

c=y\_mean-m\*x\_mean

print("\nY-Intercept: ",round(c,4))



x=np.array(df['radio'])

y=np.array(df['sales'])

y\_pred=m\*x+c

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

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

accuracy=1-(ssr/sst)

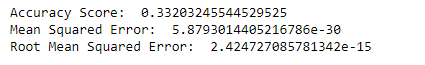
print("Accuracy Score: ", accuracy)

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

MSE=var\*var

print("Mean Squared Error: ", MSE)

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



plt.scatter(df['radio'],df['sales'],color="red", marker="^",label='Dataset points')

plt.scatter(x\_mean,y\_mean,color="green",marker="^",label='Mean point')

x=np.arange(0,50)

plt.plot(x,m\*x+c,color="blue",label='Linear Regression line')

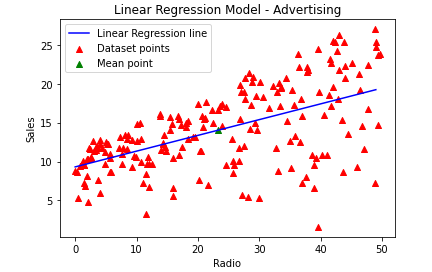
plt.title('Linear Regression Model - Advertising')

plt.xlabel('Radio')

plt.ylabel('Sales')

plt.legend()

plt.show()



Q18) Implement Multiple Linear Regression on “Advertising” dataset using Normal Equation Method.

**CODE AND CORRESPONDING SCREENSHOTS-**

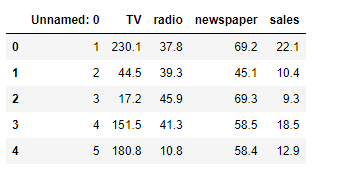
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

dataset.head()



x1=np.array(dataset['TV'])

x2=np.array(dataset['radio'])

x3=np.array(dataset['newspaper'])

y=np.array(dataset['sales'])

n=len(x1)

x0=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(x0,x1\_n,axis=1)

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

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

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]

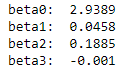
b3=theta[3]

print("\nbeta0: ",round(b0,4))

print("\nbeta1: ",round(b1,4))

print("\nbeta2: ",round(b2,4))

print("\nbeta3: ",round(b3,4))



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

y\_bar=np.mean(y)

ssr=np.sum((y-y\_pred)\*\*2)

sst=np.sum((y-y\_bar)\*\*2)

accuracy=1-(ssr/sst)

print("Accuracy Score: ", round(accuracy,6))

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

MSE=var\*\*2

print("Mean Squared Error: ", MSE)

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

