**Machine Learning PRACTICAL BACKUP**

**Name:** Gargi Bisht

**College Roll no.:** 2019316

**Examination Roll no.:** 19066570014

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

CODE:

x=int(input("Enter number 1: "))

y=int(input("Enter number 2: "))

print("Results after elementary maths operations:------")

print("Addition: ",x+y)

print("Subtraction: ",x-y)

if(y!=0):

print("Division: ",x/y)

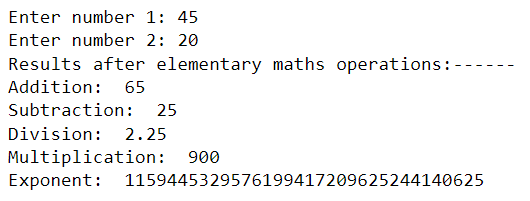
else:

print("Division not possible!!")

print("Multiplication: ",x\*y)

print("Exponent: ",x\*\*y)

OUTPUT:



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

CODE:

x=int(input("Enter number 1: "))

y=int(input("Enter number 2: "))

print("Results after boolean operations:------")

print("AND :",x and y)

print("OR :",x or y)

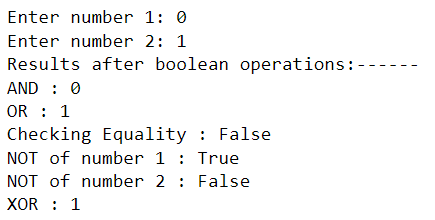
print("Checking Equality :",x==y)

print("NOT of number 1 :",not x)

print("NOT of number 2 :",not y)

print("XOR :",x^y)

OUTPUT:



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

CODE:

x= 2/3

z=16

y= """ I have awesome friends"""

print("String Formatting---------------------")

print(x)

print(y.split())

print(y.splitlines())

print(y.strip())

print(y.upper())

print(y.lower())

print(y.capitalize())

print(y.title())

print("\_".join(y.split()))

print()

print("Number Formatting----------------------")

print(int(x))

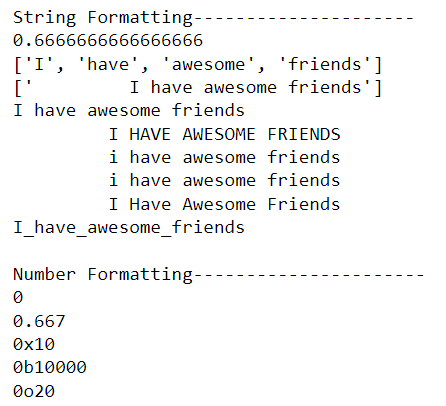
print("%0.3f"%(x))

print(hex(z))

print(bin(z))

print(oct(z))

OUTPUT:



**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.**

CODE:

import numpy as np

print('Single Dimensional Array :')

print('---------------------------------------------------------------------------')

print('With all zeroes')

print(np.zeros(5,int))

print()

print('With all ones')

print(np.ones(5,int))

print()

print('With all random integers')

print(np.random.randint(1,10,5))

print()

print('With values within a range')

print(np.arange(1,10))

print()

print('Multi Dimensional Array :')

print('---------------------------------------------------------------------------')

print('With all zeroes')

print(np.zeros((4,3),int))

print()

print('With all ones')

print(np.ones((4,3),int))

print()

print('With all random integers')

print(np.random.randint(1,10,(4,3)))

print()

print('With values within a range')

print(np.arange(1,10).reshape(3,3))

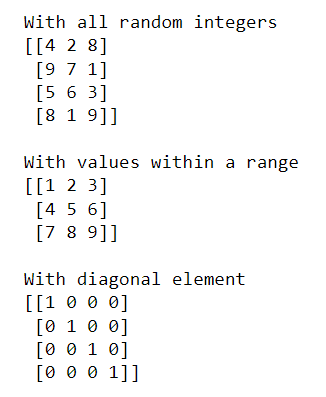
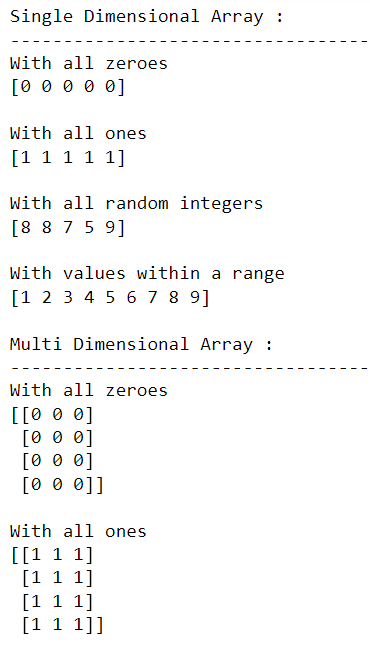
print()

print('With diagonal element')

print(np.eye(4,None,0,int))

print()

OUTPUT:



**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.**

CODE:

import numpy as np

arr=np.random.randn(4,3)

print('Array:')

print('---------------------------------------------------------------------------')

print(arr)

print()

print('Size of Array:')

print('---------------------------------------------------------------------------')

print(arr.size)

print()

print('Size of particular row (row 2):')

print('---------------------------------------------------------------------------')

print(len(arr[:,2]))

print()

print('Size of particular column (column 2):')

print('---------------------------------------------------------------------------')

print(len(arr[2,:]))

print()

print('Taking input from file: ')

print('---------------------------------------------------------------------------')

with open('numpyfile.txt', 'r') as f:

l =np.loadtxt(f)

print(l)

print()

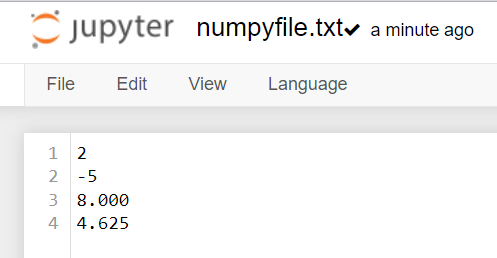
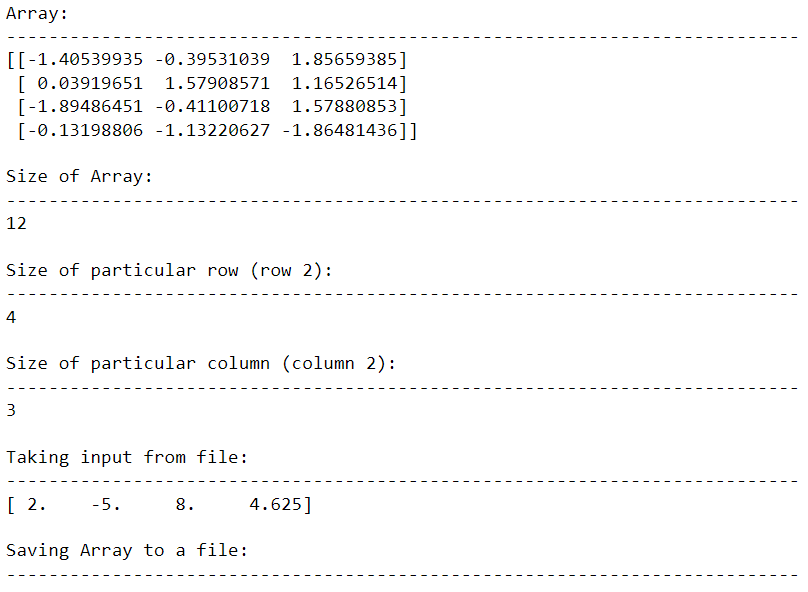
print('Saving Array to a file: ')

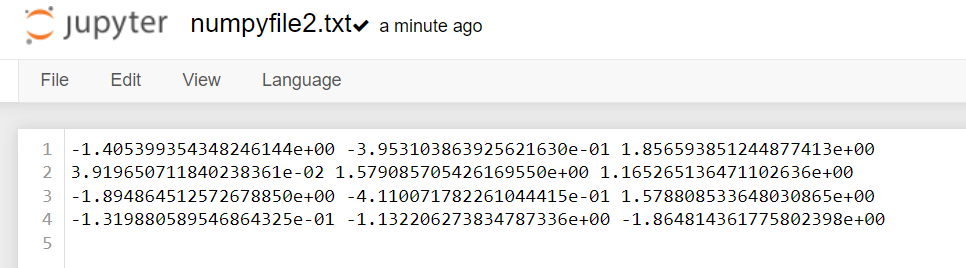
print('---------------------------------------------------------------------------')

with open('numpyfile2.txt', 'wb') as f:

np.savetxt(f,arr)

OUTPUT:





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

CODE:

import numpy as np

mat1= np.random.randint(1,10,(3,3))

mat2= np.random.randint(1,10,(3,3))

print("Matrix 1:")

print(mat1)

print("Matrix 2:")

print(mat2)

print()

print('Addition of Two matrices')

print('---------------------------------------------------------------------------')

mat3=mat1+mat2

print(mat3)

print()

print('Subtraction of Two matrices')

print('---------------------------------------------------------------------------')

mat3=mat1+mat2

print(mat3)

print()

print('Multiplication of Two matrices')

print('---------------------------------------------------------------------------')

mat3=mat1\*mat2

print(mat3)

print()

mat1= np.random.randint(1,10,(4,3))

mat2= np.random.randint(1,10,(3,2))

print("Matrix 1:")

print(mat1)

print("Matrix 2:")

print(mat2)

print()

print('Element-by-Element Multiplication of Two matrices')

print('---------------------------------------------------------------------------')

mat3=mat1.dot(mat2)

print(mat3)

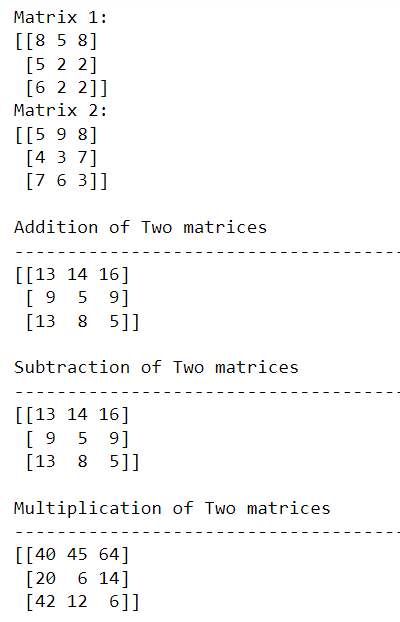
print()

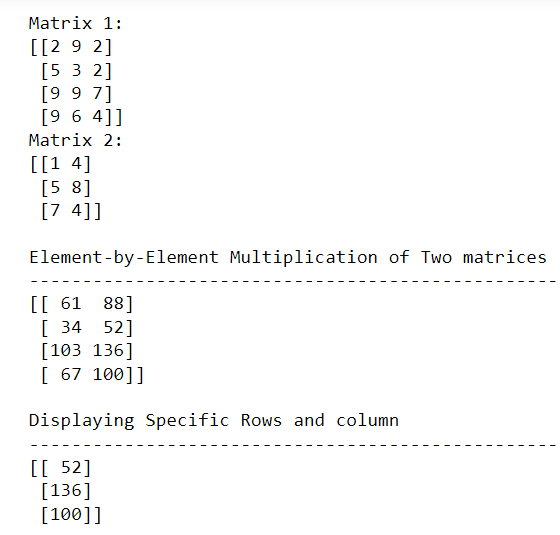
print('Displaying Specific Rows and column')

print('---------------------------------------------------------------------------')

print(mat3[1:,1:])

OUTPUT:





**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.**

CODE:

import numpy as np

arr=np.random.randn(4,3)

print('Array:')

print('---------------------------------------------------------------------------')

print(arr)

print()

print('Absolute Value of the Array')

print('---------------------------------------------------------------------------')

print(np.absolute(arr))

print()

print('Negative Value of the Array')

print('---------------------------------------------------------------------------')

print(np.negative(arr))

print()

print('Array after deleting row 2')

print('---------------------------------------------------------------------------')

print(np.delete(arr,2,0))

print()

print('Array after deleting column 2')

print('---------------------------------------------------------------------------')

print(np.delete(arr,2,1))

print()

print('Maximum of Array')

print('---------------------------------------------------------------------------')

print(np.max(arr))

print()

print('Minimum of Array')

print('---------------------------------------------------------------------------')

print(np.min(arr))

print()

print('Sum of all elements of Array')

print('---------------------------------------------------------------------------')

print(np.sum(arr))

print()

print('Sum of all elements of Array row-wise')

print('---------------------------------------------------------------------------')

print(np.sum(arr,0))

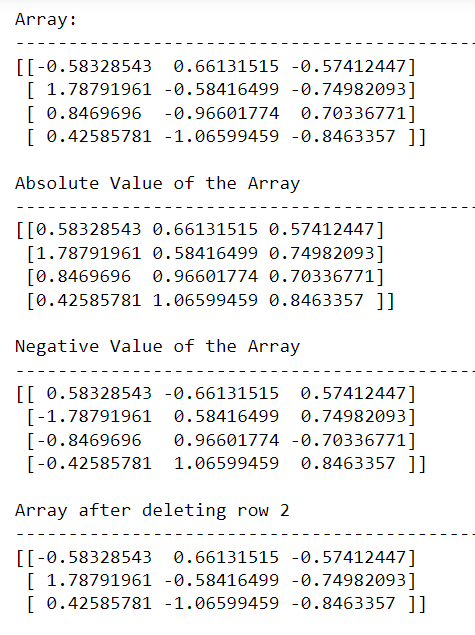
print()

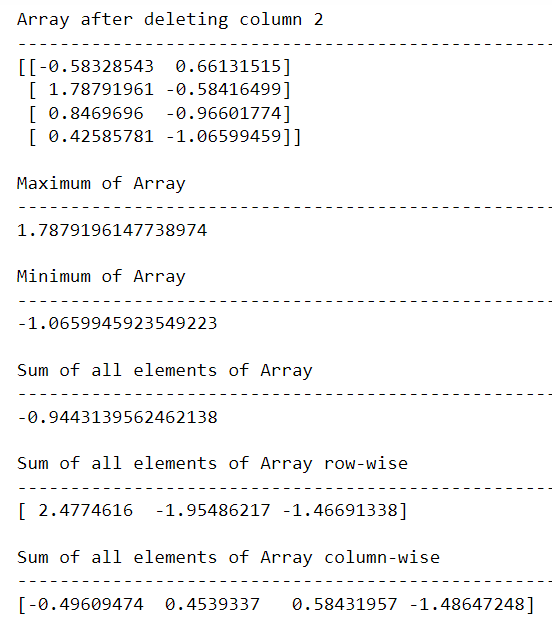
print('Sum of all elements of Array column-wise')

print('---------------------------------------------------------------------------')

print(np.sum(arr,1))

OUTPUT:





**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.**

CODE:

import matplotlib.pyplot as plt

import numpy as np

x=np.arange(40)

x

plt.plot(x)

plt.show()

y=np.sin(x)

plt.plot(x)

plt.plot(y)

plt.show()

y=np.cos(x)

plt.plot(x,color='green')

plt.plot(y)

plt.show()

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

y=x\*\*2

plt.scatter(x,y,color="orange", label="squares of number", marker="o") #color, label, marker are optional arguments

plt.legend()

plt.xlabel("numbers")

plt.ylabel("squares")

plt.show()

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

plt.plot(x,x\*\*2,label="squares of number", marker="o")

plt.plot(x,x\*\*3,label="cubes of number", marker="o")

plt.legend()

plt.xlabel("numbers")

plt.title("Squares and Cubes")

plt.show()

numbers=np.random.randint(0,10,5)

indices=np.arange(5)

indices=indices+2015

numbers2=np.random.randint(0,10,5)

indices2=np.arange(5)

indices2=indices2+2015

print(numbers)

print(numbers2)

plt.bar(indices,numbers,0.25,color="red",label="rainfall")

plt.bar(indices2+0.25,numbers2,0.25,color="green",label="humidity")

plt.legend()

plt.show()

labels=["english","hindi","maths","science","social science","computers"]

values=[90,80,40,73,78,43]

plt.pie(values,labels=labels,radius=1)

plt.show()

plt.pie(values,labels=labels,radius=1,explode=(0,0,1,0,0,0),shadow=True)

plt.show()

u=5

sigma=2

vals=u+sigma\*np.random.randn(1000)

print(vals.shape)

plt.hist(vals,50)

plt.show()

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

plt.subplot(221)

plt.plot(x,x\*\*2, label="2018")

plt.plot(x,x\*\*3, label="2017")

plt.subplot(222)

plt.hist(vals,50)

plt.subplot(223)

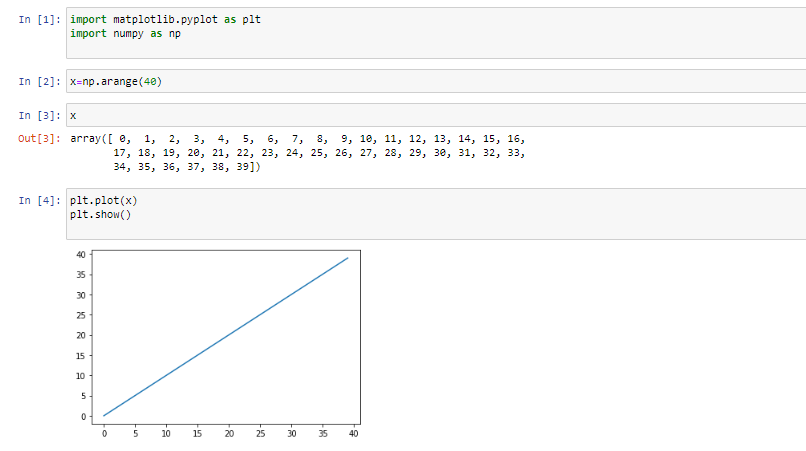
plt.pie(values,labels=labels,radius=1)

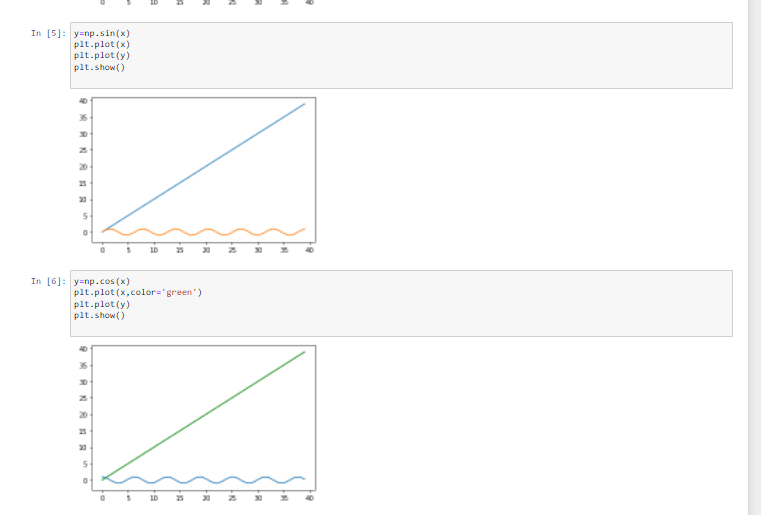
plt.subplot(224)

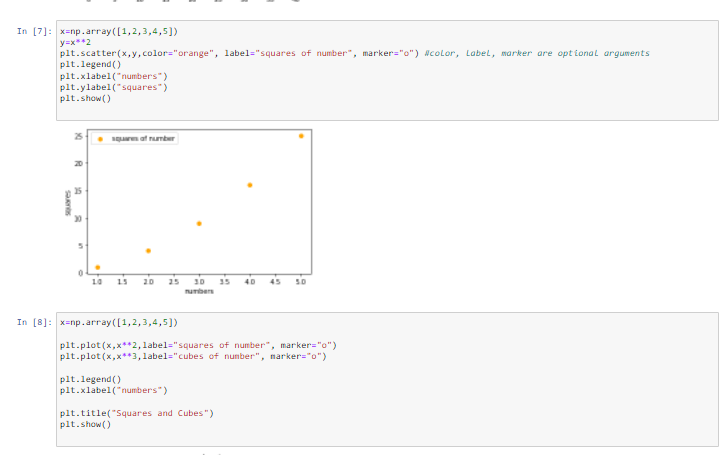
plt.pie(values,labels=labels,radius=1,explode=(0,0,1,1,0,1),shadow=True)

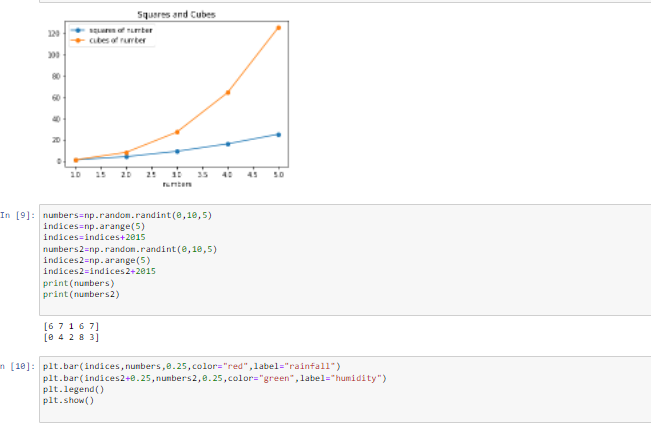
plt.show()

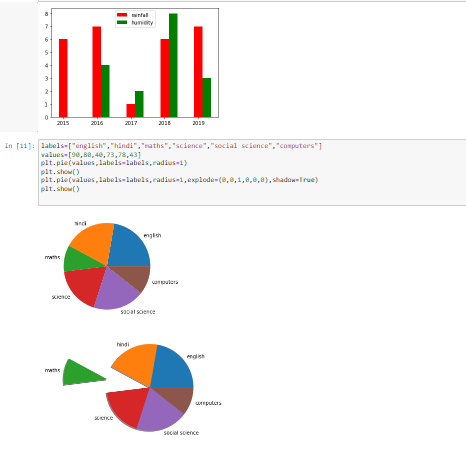
OUTPUT:

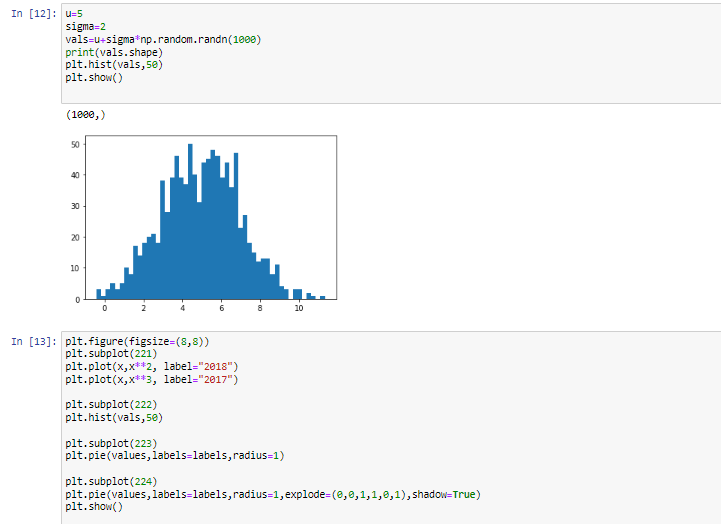












**10. Use conditional statements and different type of loops based on simple examples.**

CODE:

A = [[1, 2, 3],

[4, 5, 6],

[7, 8, 9]]

B = [[10, 11, 12, 13],

[14, 15, 16, 17],

[18, 19, 20, 21]]

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

[0, 0, 0, 0],

[0, 0, 0, 0]]

for i in range(len(A)):

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

for k in range(len(B)):

result[i][j] += A[i][k] \* B[k][j]

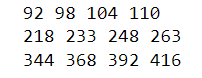
for i in range(len(result)):

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

print(result[i][j],end=" ")

print()

OUTPUT:

****

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

CODE:

1. Plotting a data

import matplotlib.pyplot as plt

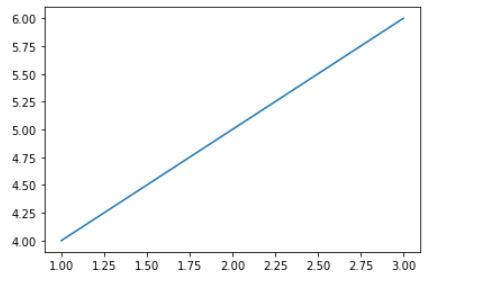
plt.plot([1,2,3],[4,5,6])

#[<matplotlib.lines.Line2D object at 0x00000224330A0790>]

plt.draw()

plt.show()

Output:



1. Vectorization

Code:

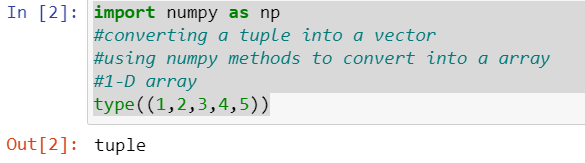
import numpy as np

#converting a tuple into a vector

#using numpy methods to convert into a array

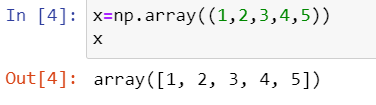
#1-D array

type((1,2,3,4,5))

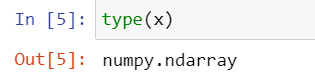


x=np.array((1,2,3,4,5))

x



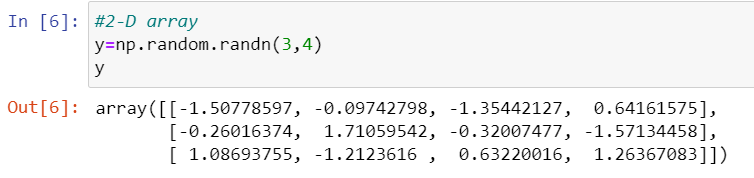
type(x)



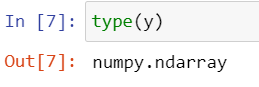
#2-D array

y=np.random.randn(3,4)

y



type(y)



1. Operations on Vectors

Code:

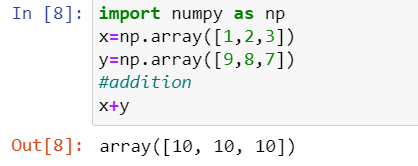
import numpy as np

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

y=np.array([9,8,7])

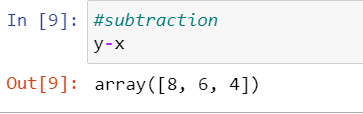
#addition

x+y



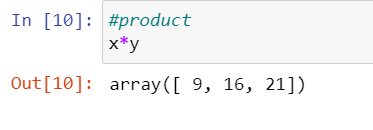
#subtraction

y-x



#product

x\*y



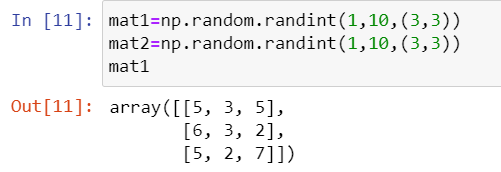
1. Operations on Matrices

Code:

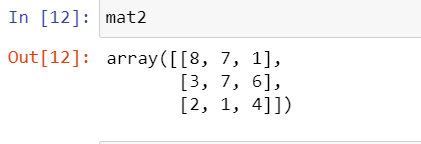
mat1=np.random.randint(1,10,(3,3))

mat2=np.random.randint(1,10,(3,3))

mat1



mat2



#addition

mat1 + mat2

#subtraction

mat1 - mat2

#product

mat1 \* mat2

#matrix multiplication

mat1.dot(mat2)

#transpose

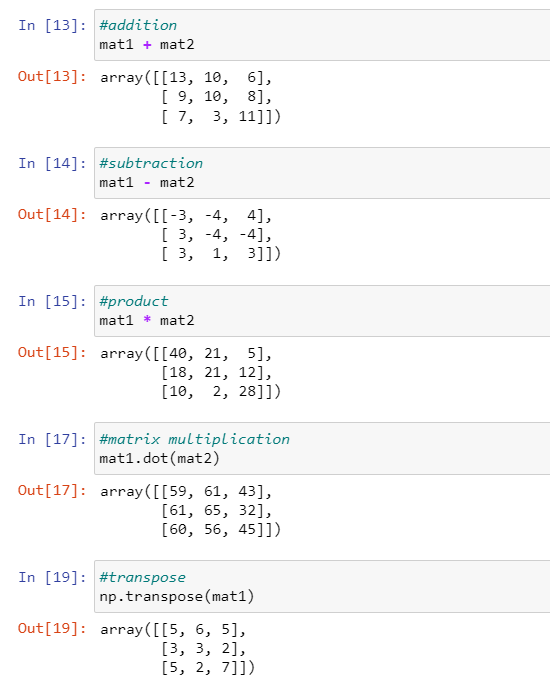
np.transpose(mat1)

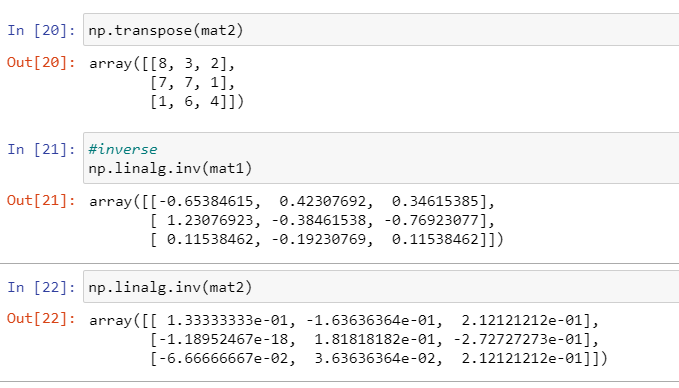
np.transpose(mat2)

#inverse

np.linalg.inv(mat1)

np.linalg.inv(mat2)





OUTPUT:

**Q12A. Implement Simple Linear Regression using analytical method and depict model on scatter data plot.**

**#a) Take x=[1,2,4], y=[2,3,6]**

CODE

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

x=np.array([1,2,4])

y=np.array([2,3,6])

x\_mean=np.mean(x)

print(x\_mean)

y\_mean=np.mean(y)

print(y\_mean)

n=x.size

print(n)

ss\_xy=np.sum(x\*y) - n\*x\_mean\*y\_mean

print(ss\_xy)

ss\_xx=np.sum(x\*x) - n\*x\_mean\*x\_mean

print(ss\_xx)

b1=ss\_xy/ss\_xx

print(b1)

b0=y\_mean-b1\*x\_mean

print(b0)

plt.scatter(x,y)

y\_pred=b0+b1\*x

print(y\_pred)

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

plt.xlabel("x")

plt.ylabel("y")

plt.title("Linear Regression[x~y]")

plt.show()

#b) Regress Sales~Radio from Advertisingdata.csv

col\_list = ["TV", "radio","newspaper","sales"]

data = pd.read\_csv("Advertising.csv", usecols=col\_list)

data["sales"] #x-axis

data["radio"] #y-axis

x\_mean=np.mean(data["sales"])

print(x\_mean)

y\_mean=np.mean(data["radio"])

print(y\_mean)

n=data["sales"].size

print(n)

ss\_xy=np.sum(data["sales"]\*data["radio"]) - n\*x\_mean\*y\_mean

print(ss\_xy)

ss\_xx=np.sum(data["sales"]\*data["sales"]) - n\*x\_mean\*x\_mean

print(ss\_xx)

b1=ss\_xy/ss\_xx

print(b1)

b0=y\_mean-b1\*x\_mean

print(b0)

plt.scatter(data["sales"],data["radio"])

y\_pred=b0+b1\*data["sales"]

print(y\_pred)

plt.plot(data["sales"],y\_pred,color='red')

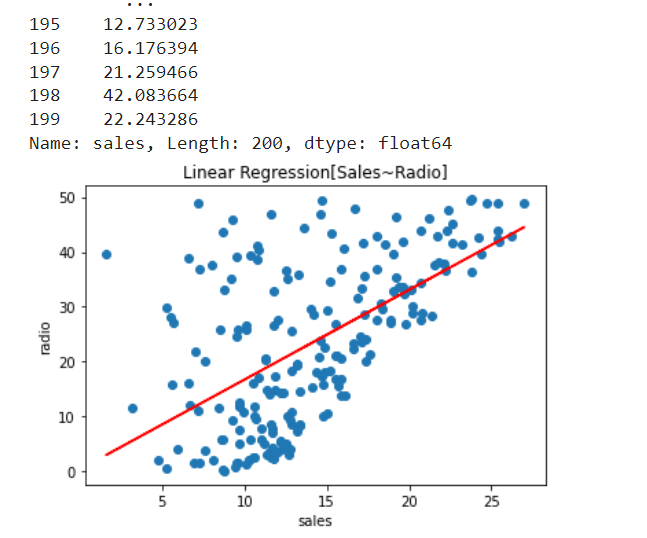
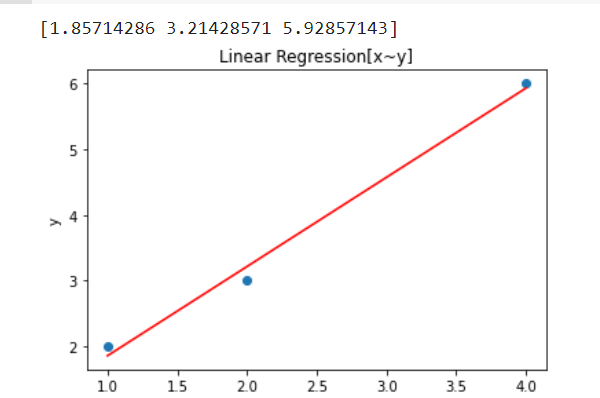
plt.xlabel("sales")

plt.ylabel("radio")

plt.title("Linear Regression[Sales~Radio]")

plt.show()

OUTPUT



**12. Implement Linear Regression problem. For example, based on a dataset comprising of existing set of prices and area/size of the houses, predict the estimated price of a given house.**

CODE:

#!/usr/bin/env python

# coding: utf-8

# In[2]:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

import statsmodels.api as sm

# In[3]:

data = pd.read\_csv(r"Advertising.csv")

data

data.columns

# In[18]:

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

plt.scatter(

data['TV'],

data['sales']

)

plt.xlabel("TV ")

plt.ylabel("Sales ")

plt.show()

# In[19]:

X = data['TV'].values.reshape(-1,1)

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

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

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

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

reg = LinearRegression()

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

# In[20]:

print(reg.coef\_[0][0])

print(reg.intercept\_[0])

print("The linear model is: Y = {:.5} + {:.5}X".format(reg.intercept\_[0], reg.coef\_[0][0]))

# In[24]:

predictions = reg.predict(X\_test)

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

plt.scatter(

data['TV'],

data['sales']

)

plt.plot(

X\_test,

predictions,

linewidth=2,

color='red'

)

plt.xlabel("TV ")

plt.ylabel("Sales ")

plt.show()

# In[13]:

X=X\_train

y=y\_train

X2 = sm.add\_constant(X)

est = sm.OLS(y, X2)

est2 = est.fit()

print(est2.summary())

# In[14]:

print('Train Score :', reg.score(X\_train,y\_train))

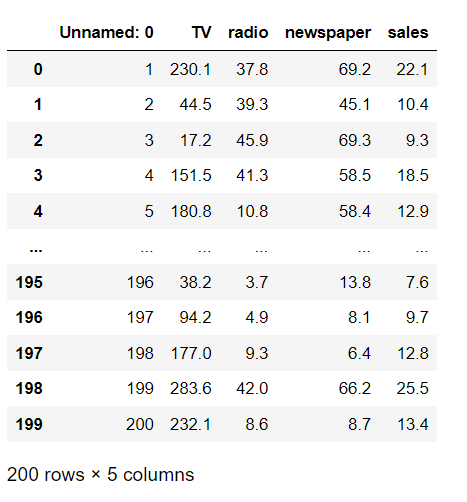
print('Test Score:', reg.score(X\_test,y\_test))

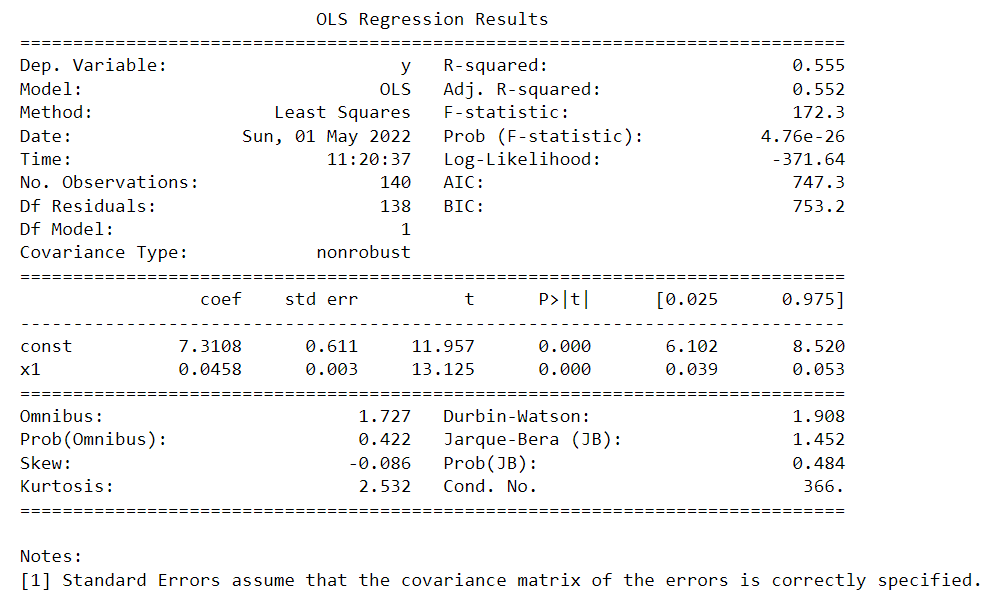
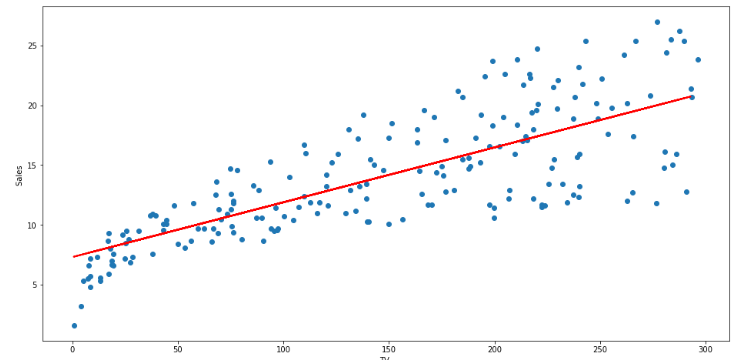
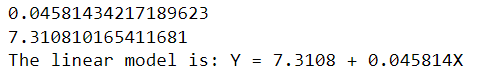
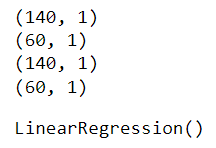
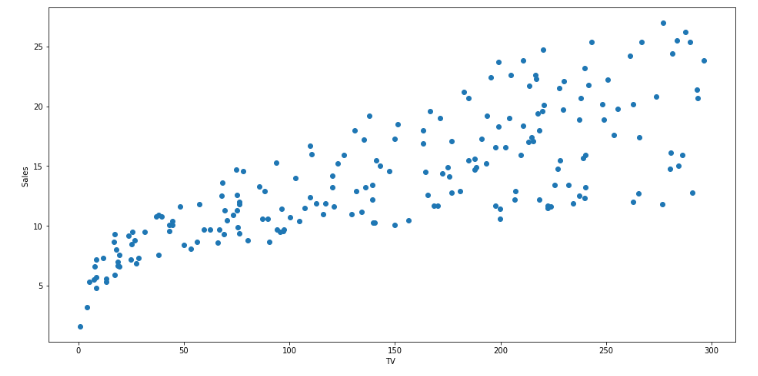
from sklearn import metrics

print('MSE :', metrics.mean\_squared\_error(y\_test,predictions))

print('RMSE :', np.sqrt(metrics.mean\_squared\_error(y\_test,predictions)))

OUTPUT:





**13. 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:

#!/usr/bin/env python

# coding: utf-8

# In[2]:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

import statsmodels.api as sm

# In[6]:

data = pd.read\_csv(r"Advertising.csv")

# In[7]:

Xs = data.drop(['sales', 'Unnamed: 0'], axis=1)

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

reg = LinearRegression()

reg.fit(Xs, y)

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

# In[8]:

X = np.column\_stack((data['TV'], data['radio'], data['newspaper']))

y = data['sales']

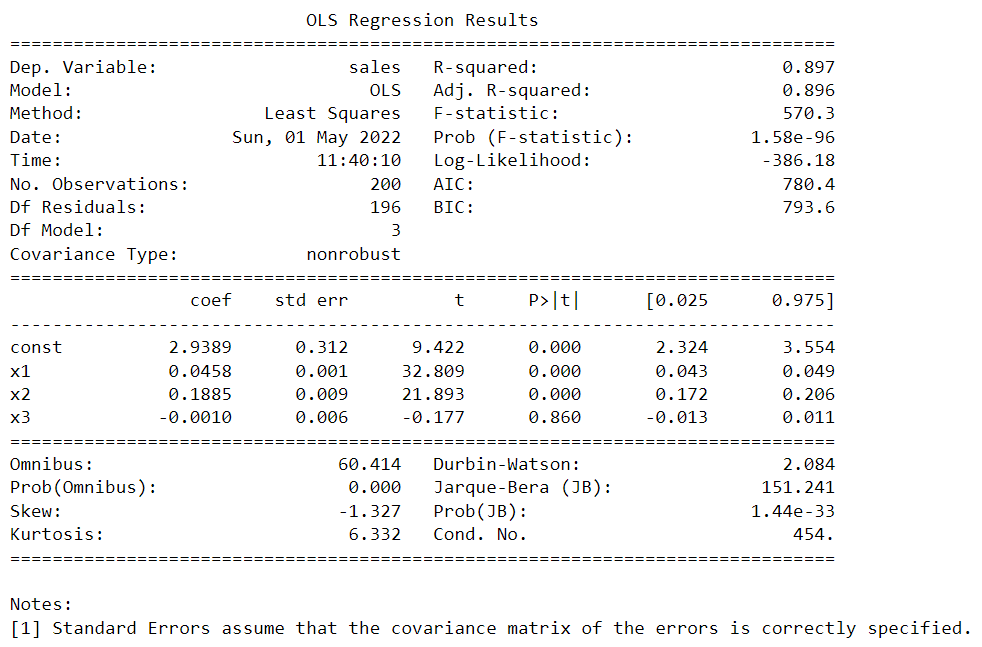
X2 = sm.add\_constant(X)

est = sm.OLS(y, X2)

est2 = est.fit()

print(est2.summary())

OUTPUT:



**14. Implement a classification/ logistic regression problem. For example based on different features of students data, classify, whether a student is suitable for a particular activity. Based on the available dataset, a student can also implement another classification problem like checking whether an email is spam or not.**

CODE:

#!/usr/bin/env python

# coding: utf-8

# In[45]:

import pandas as pd

from matplotlib import pyplot as plt

from sklearn.datasets import load\_breast\_cancer

# In[46]:

df = load\_breast\_cancer()

#df

# In[47]:

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

# In[48]:

#X\_train, X\_test, y\_train, y\_test = train\_test\_split(df[['age']],df.bought\_insurance,train\_size=0.9)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(df.data,df.target,test\_size=0.2, stratify=df.target,random\_state=42)

# In[49]:

X\_test.shape

# In[50]:

Y\_test.shape

# In[51]:

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(C=0.1)

# In[53]:

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

# In[54]:

#X\_test

# In[41]:

y\_predicted = model.predict(X\_test)

# In[42]:

print('Accuracy on the training subset: {:.3f}'.format(model.score(X\_train, Y\_train)))

print('Accuracy on the test subset: {:.3f}'.format(model.score(X\_test, Y\_test)))

# In[43]:

#X\_test

# In[44]:

from sklearn.metrics import confusion\_matrix

confusion\_matrix(y\_test,y\_predicted)

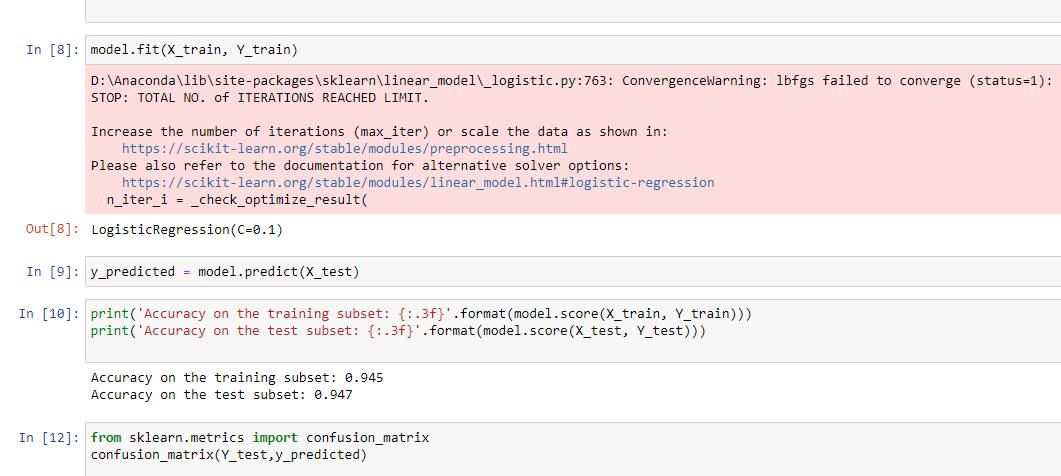
# In[57]:

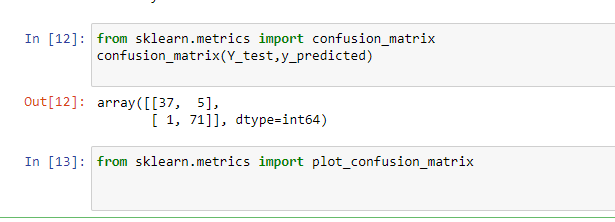
from sklearn.metrics import plot\_confusion\_matrix

# In[ ]:

**OUTPUT:**







OUTPUT:

**15. Use some function for regularization of dataset based on problem 14.**

CODE:

#!/usr/bin/env python

# coding: utf-8

# In[1]:

import pandas as pd

from matplotlib import pyplot as plt

get\_ipython().run\_line\_magic('matplotlib', 'inline')

from sklearn.datasets import load\_breast\_cancer

# In[2]:

df = load\_breast\_cancer()

#df

# In[3]:

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

# In[4]:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.data,df.target,test\_size=0.2, stratify=df.target,random\_state=42)

# In[5]:

X\_test.shape

# In[6]:

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(C=100)

# In[7]:

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

# In[8]:

model.coef\_

# In[9]:

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(C=1)

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

model.coef\_

# In[10]:

### Coefficent's Value (decrease) if C value (decrease)

### Coefficent's Value (increase) if C value (increase)

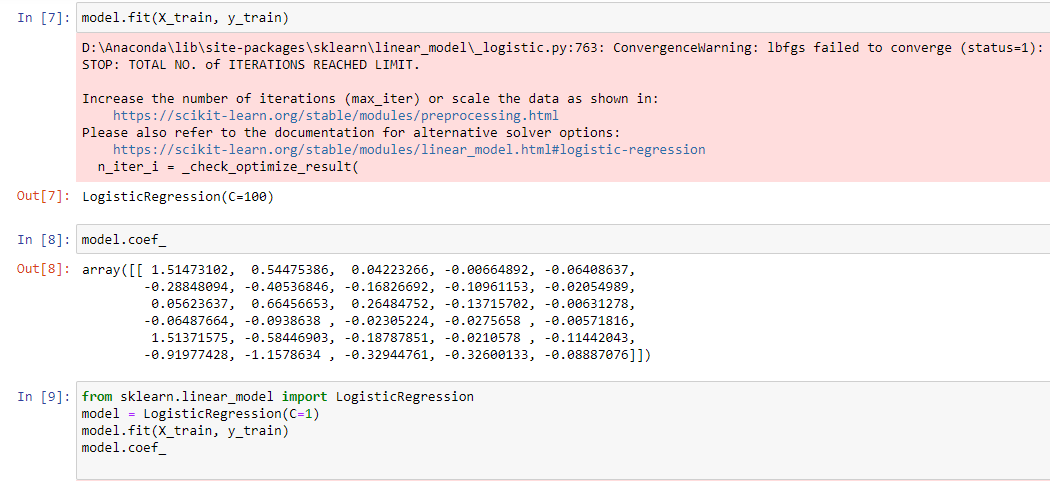
# In[11]:

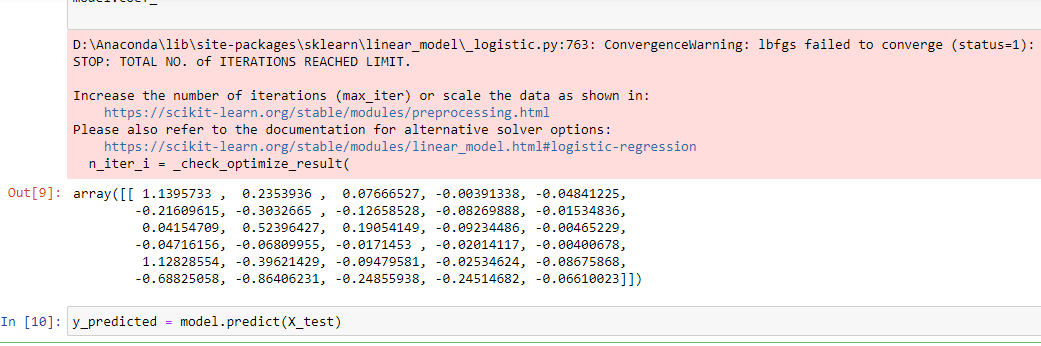
y\_predicted = model.predict(X\_test)

# In[ ]:

OUTPUT:







OUTPUT:

**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.**

CODE:

#!/usr/bin/env python

# coding: utf-8

# In[1]:

from numpy import\*

import pandas as pd

from sklearn.datasets import load\_breast\_cancer

from sklearn.neural\_network import MLPClassifier

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

# In[2]:

df = load\_breast\_cancer()

# In[3]:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.data,df.target,test\_size=0.2,random\_state=4)

y\_test

# In[4]:

nn=MLPClassifier(activation='logistic',solver='sgd',hidden\_layer\_sizes=(10,15),random\_state=1)

# In[5]:

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

# In[6]:

pred=nn.predict(X\_test)

pred

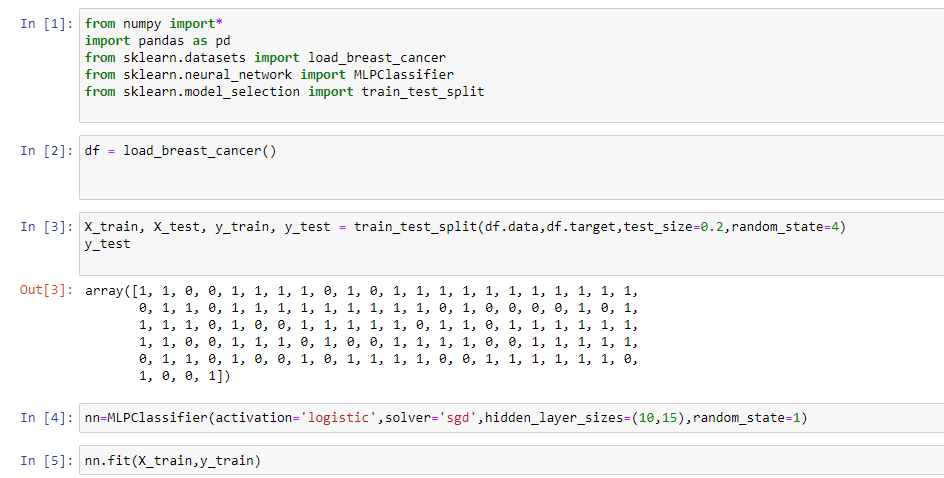
# In[7]:

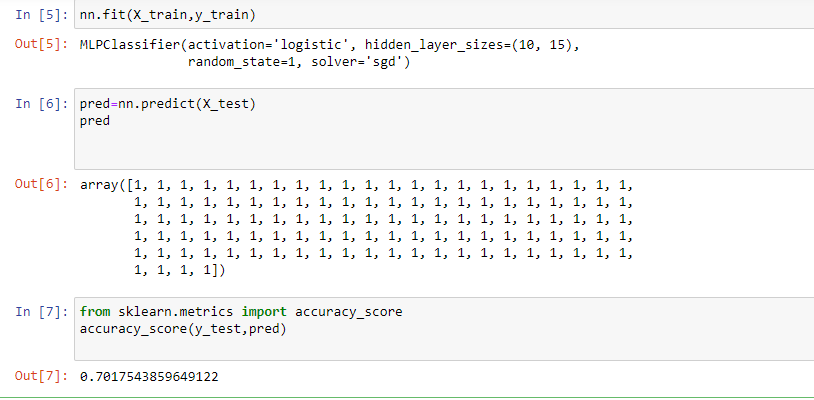
from sklearn.metrics import accuracy\_score

accuracy\_score(y\_test,pred)

# In[ ]:

OUTPUT:





OUTPUT:

**EXTRA QUESTIONS:**

**Que-17:Simple Linear Regression Using Least Square method.**

CODE:

# Making imports

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (12.0, 9.0)

# Preprocessing Input data

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

X = data.iloc[:, 0]

Y = data.iloc[:, 1]

plt.scatter(X, Y)

plt.show()

# Building the model

X\_mean = np.mean(X)

Y\_mean = np.mean(Y)

num = 0

den = 0

for i in range(len(X)):

num += (X[i] - X\_mean)\*(Y[i] - Y\_mean)

den += (X[i] - X\_mean)\*\*2

m = num / den

c = Y\_mean - m\*X\_mean

print (m, c)

# Making predictions

Y\_pred = m\*X + c

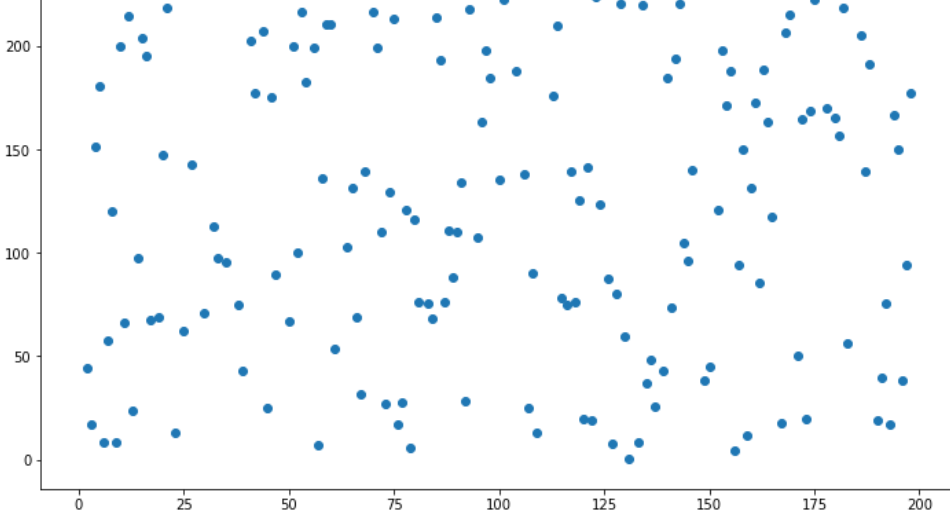
plt.scatter(X, Y) # actual

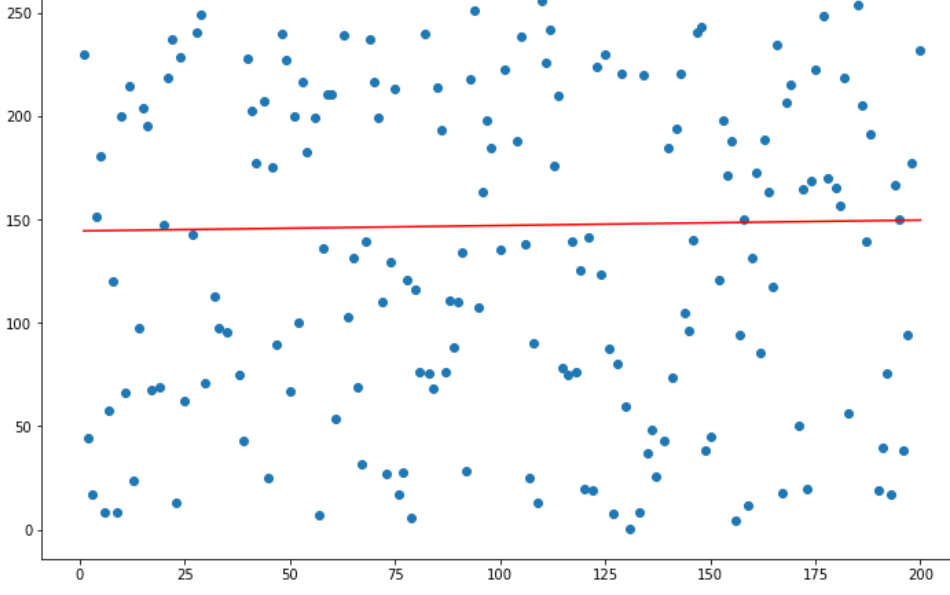
# plt.scatter(X, Y\_pred, color='red')

plt.plot([min(X), max(X)], [min(Y\_pred), max(Y\_pred)], color='red') # predicted

plt.show()

OUTPUT:

****

****

**Que-18:Multiple Linear Regression using Normal Equation Method.**

CODE:

# Multiple linear regression using Normal Method

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

col\_list = ["TV", "radio","newspaper","sales"]

data = pd.read\_csv("Advertising.csv", usecols=col\_list)

data.head()

x1 = data["TV"]

x2 = data["radio"]

x3 = data["newspaper"]

y = data["sales"]

print(x1.shape)

print(x2.shape)

print(x3.shape)

#plot Tv vs Sale

plt.scatter(x1,y)

plt.xlabel("TV")

plt.ylabel("Sale")

#plot Radio vs Sale

plt.scatter(x2,y)

plt.xlabel("radio")

plt.ylabel("Sale")

#plot Radio vs Sale

plt.scatter(x3,y)

plt.xlabel("newspaper")

plt.ylabel("Sale")

x1 = np.array(x1)

x2 = np.array(x2)

x3 = np.array(x3)

y = np.array(y)

n = len(x1)

n

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

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

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

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

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

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

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

x\_new

x\_new\_transpose = np.transpose(x\_new)

x\_new\_transpose\_dot\_x\_new = x\_new\_transpose.dot(x\_new)

temp\_1 = np.linalg.inv(x\_new\_transpose\_dot\_x\_new)

temp\_2 = x\_new\_transpose.dot(y)

theta = temp\_1.dot(temp\_2)

theta

beta\_0 = theta[0]

beta\_1 = theta[1]

beta\_2 = theta[2]

beta\_3 = theta[3]

print(beta\_0)

print(beta\_1)

print(beta\_2)

print(beta\_3)

def predict\_values(beta\_0,beta\_1,beta\_2,beta\_3,tv,radio,newspaper):

predicted\_value = beta\_0 + tv\*beta\_1 + radio\*beta\_2 + newspaper\* beta\_3

return predicted\_value

tv = 10

radio = 20

newspaper = 30

print(predict\_values(beta\_0,beta\_1,beta\_2,beta\_3,tv,radio,newspaper))

OUTPUT:

