MACHINE LEARNING PRACTICALS

**NAME: SNEHA VASHISHTHA**

**EXAMINATION ROLL NO: 19066570042**

**COLLEGE ROLL NO: 2019347**

**NAME OF COURSE: BSC. (HONS) COMPUTER SCIENCE**

1. Perform elementary mathematical operations in Python 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):

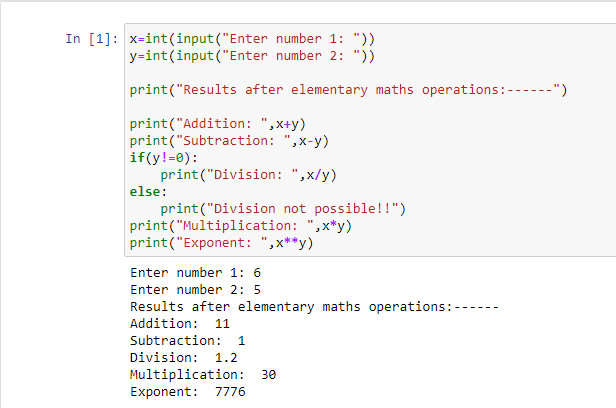
print("Division: ",x/y)

else:

print("Division not possible!!")

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

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



2.Perform elementary logical operations in Python (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)

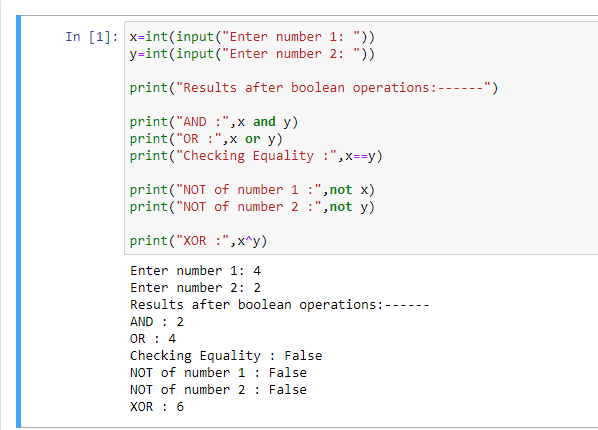
print("OR :",x or y)

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

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

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

print("XOR :",x^y)



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

CODE:

x= 2/3

z=16

y= """WELCOME SUMMER!!"""

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----------------------")

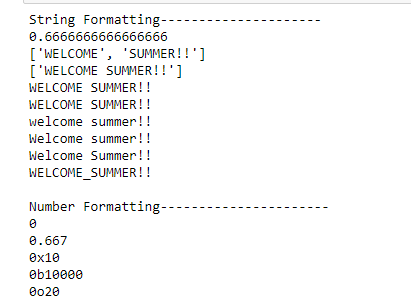
print(int(x))

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

print(hex(z))

print(bin(z))

print(oct(z))



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')

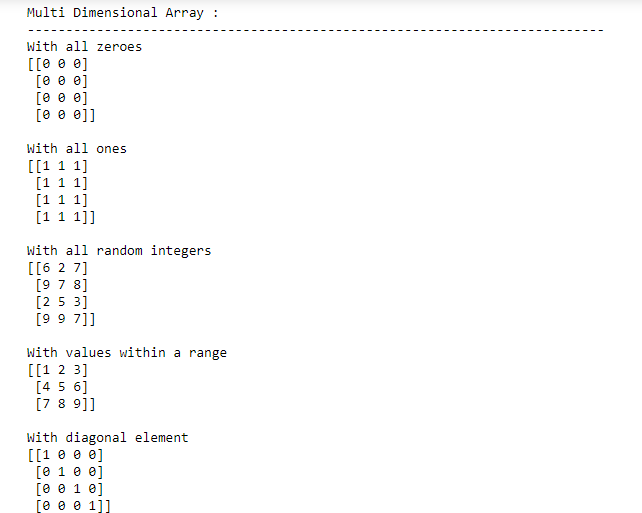
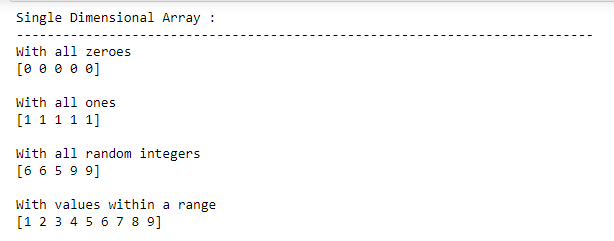
print(np.arange(1,10).reshape(3,3))

print()

print('With diagonal element')

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

print()



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('numpy2text.txt', 'r') as f:

l =np.loadtxt(f)

print(l)

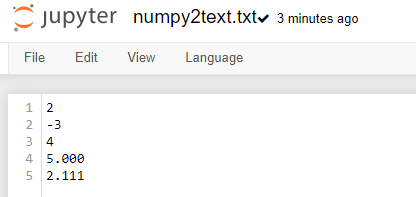
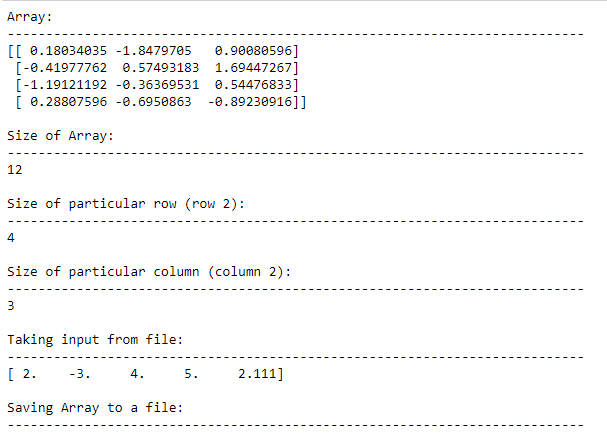
print()

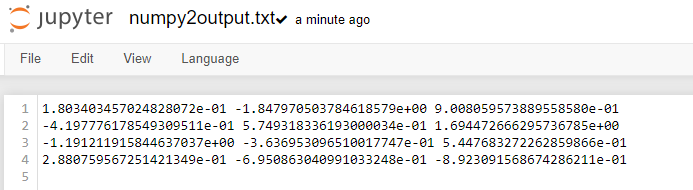
print('Saving Array to a file: ')

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

with open('numpy2output.txt', 'wb') as f:

np.savetxt(f,arr)





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)

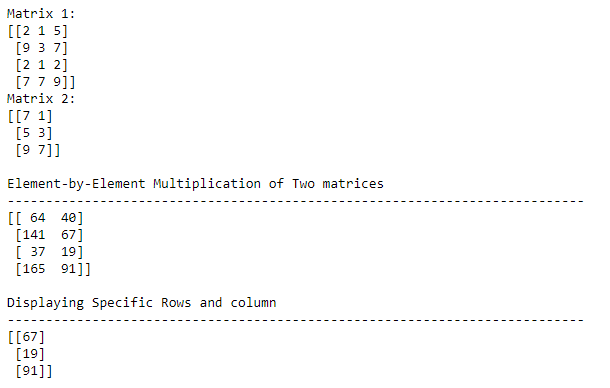
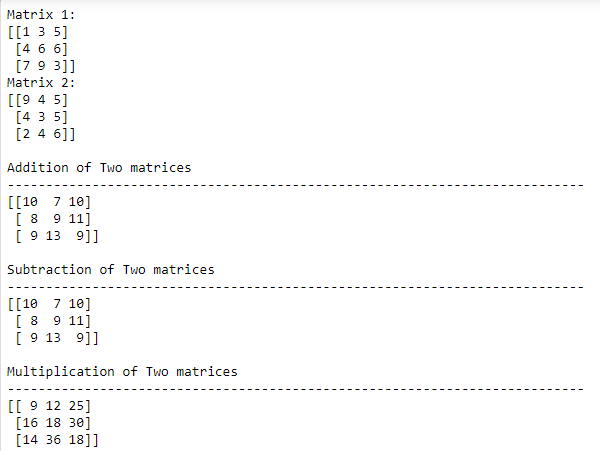
print(mat3)

print()

print('Displaying Specific Rows and column')

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

print(mat3[1:,1:])



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.

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

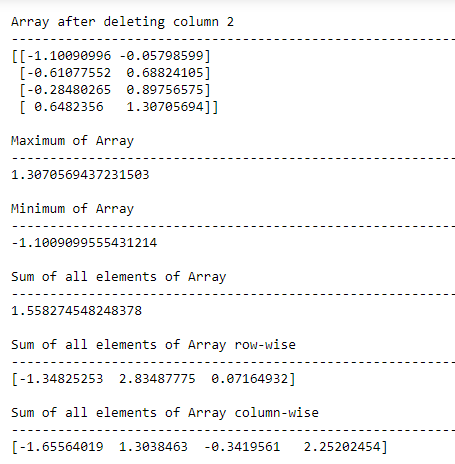
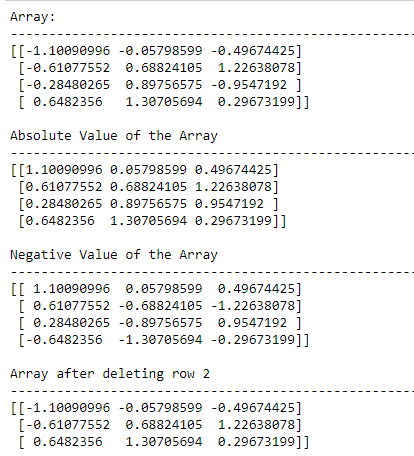
print(np.sum(arr,0))

print()

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

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

print(np.sum(arr,1))



8. Create the various types of plots/charts like histograms, and plots 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=[0,1,2,3,4,5]

y=[0,1,4,9,16,25]

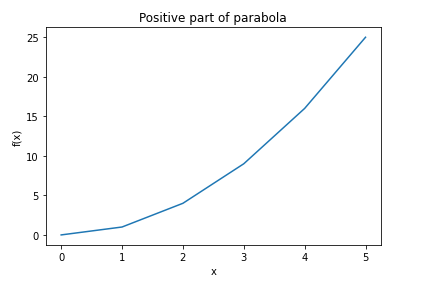
plt.plot(x,y)

plt.title("Positive part of parabola")

plt.xlabel("x")

plt.ylabel("f(x)")

plt.show()



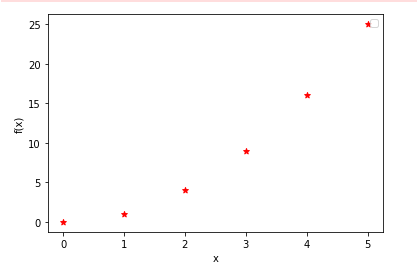
plt.scatter(x,y,color="red", marker="\*")

plt.legend()

plt.xlabel("x")

plt.ylabel("f(x)")

plt.show()



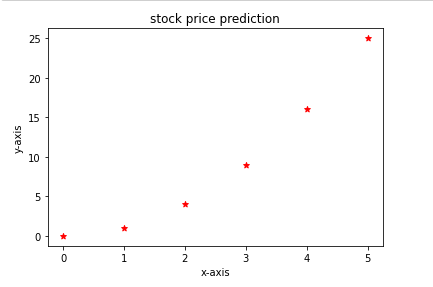
plt.scatter(x,y,color="red", marker="\*")

plt.xlabel("x-axis")

plt.ylabel("y-axis")

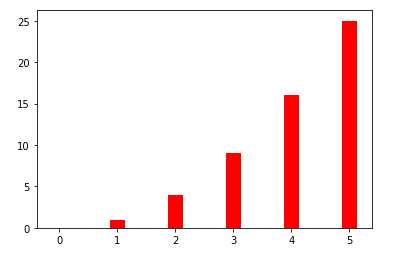
plt.title("stock price prediction")

plt.show()



plt.bar(x,y,0.25,color="red",label="x")

plt.show()



labels=["cloud computing","big data","IOT","ML-AI"]

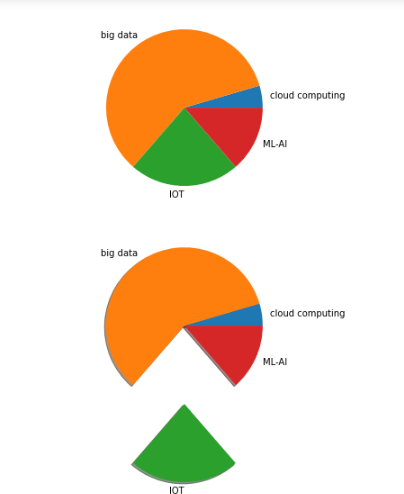
values=[5,65,25,15]

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

plt.show()

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

plt.show()



np.random.seed(23685752)

N\_points=10000

n\_bins=20

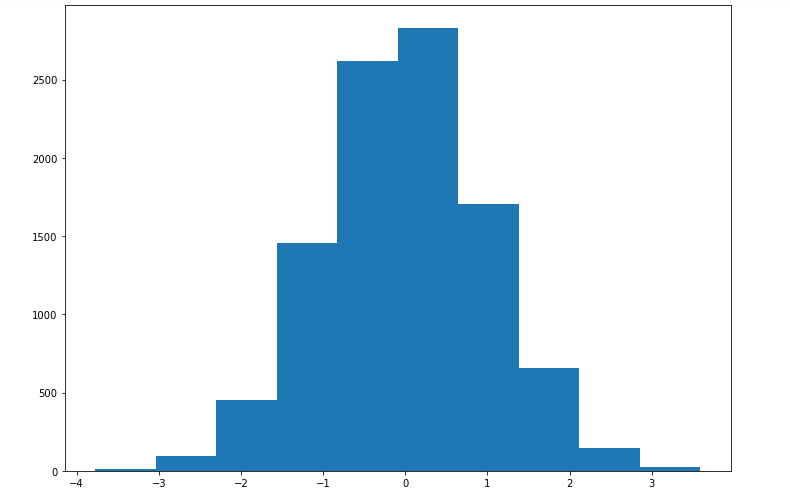
x=np.random.randn(N\_points)

y=.8\*\*x+np.random.randn(10000)+25

fig, axs=plt.subplots(1,1,figsize=(10,7),tight\_layout=True)

axs.hist(x)

plt.show()



x=np.arange(0,4\*np.pi,0.1)

y=np.sin(x)

z=np.cos(x)

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

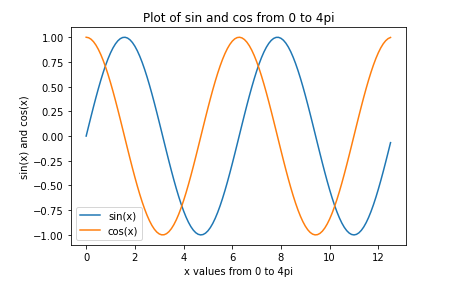
plt.xlabel('x values from 0 to 4pi')

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()



x=np.arange(0,8\*np.pi,0.1)

k=np.tan(x)

plt.plot(x,k)

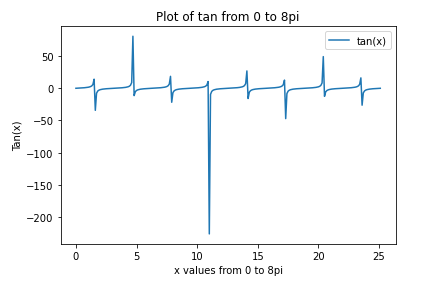
plt.xlabel('x values from 0 to 8pi')

plt.ylabel('Tan(x)')

plt.title('Plot of tan from 0 to 8pi')

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

plt.show()



9. Generate different subplots from a given plot and color plot data.

CODE:

plt.figure(figsize=(10,10))

plt.subplot(221)

plt.plot(x,x\*\*2,label="x squared",color="red")

plt.plot(x,x\*\*3,label="x cubed",color="green")

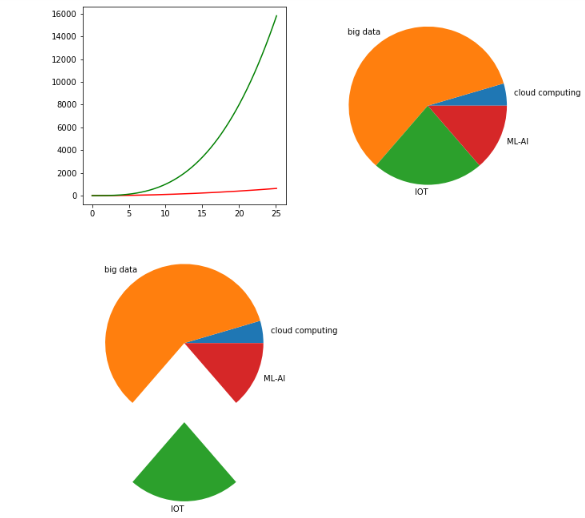
plt.subplot(222)

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

plt.subplot(223)

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

plt.show()



10. Use conditional statements and different types of loops based on simple example/s. ix, adding, subtracting, or multiplying two matrices.

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)):

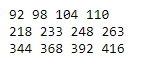
result[i][j] += A[i][k] \* B[k][j]

for i in range(len(result)):

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

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

print()



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

CODE:

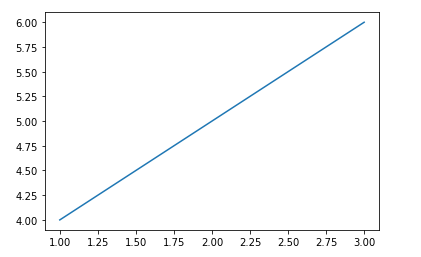
import matplotlib.pyplot as plt

import numpy as np

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

plt.draw()

plt.show()



#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

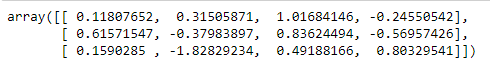
type(x)



#2-D array

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

y



type(y)



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

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

#addition

x+y



#subtraction

y-x



#product

x\*y

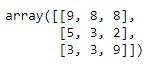


#Operations on Matrices

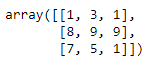
mat1=np.random.randint(1,10,(3,3))

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

mat1



mat2



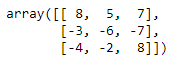
#addition

mat1+mat2



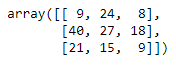
#subtration

mat1-mat2



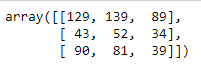
#product

mat1\*mat2



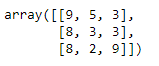
#matrix multipliaction

mat1.dot(mat2)

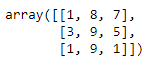


#transpose

np.transpose(mat1)

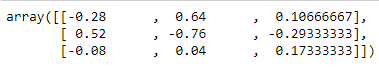


np.transpose(mat2)

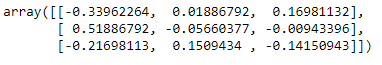


#inverse

np.linalg.inv(mat1)



np.linalg.inv(mat2)



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

CODE:

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

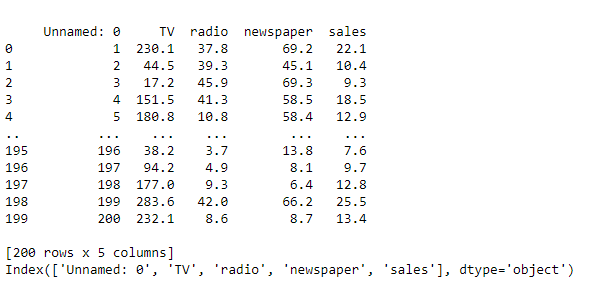
import statsmodels.api as sm

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

print()

print(data)

print(data.columns)



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

plt.scatter(

data['TV'],

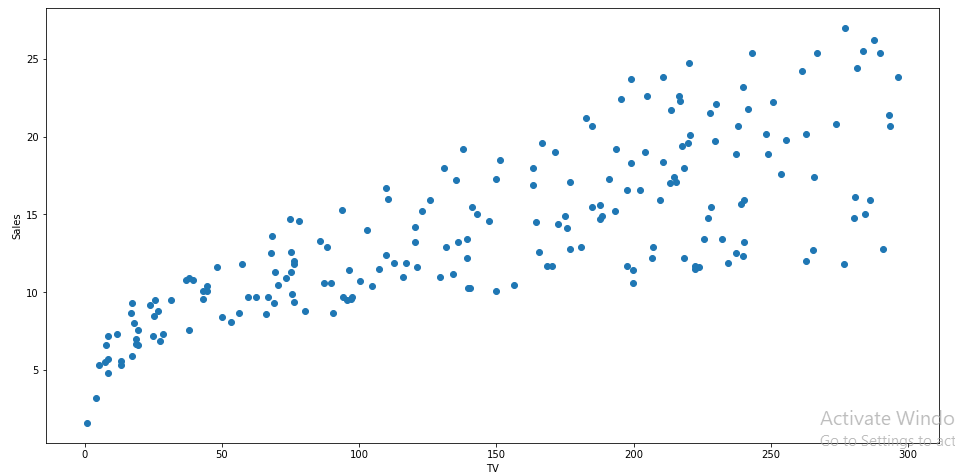
data['sales']

)

plt.xlabel("TV")

plt.ylabel("Sales")

plt.show()



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)

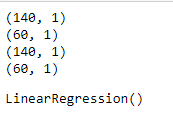
print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

reg = LinearRegression()

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



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]))



predictions = reg.predict(X\_test)

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

plt.scatter(

data['TV'],

data['sales']

)

plt.plot(

X\_test,

predictions,

linewidth=2,

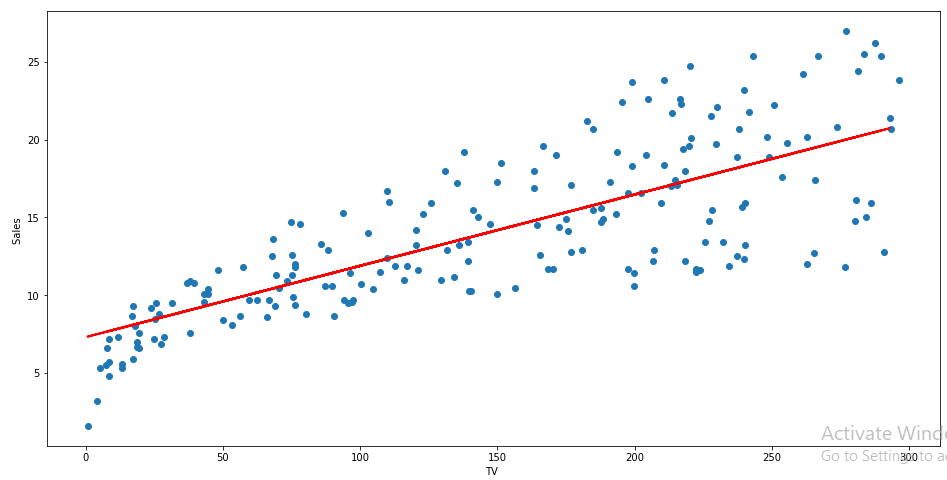
color='red'

)

plt.xlabel("TV ")

plt.ylabel("Sales ")

plt.show()



errors = list()

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

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

# store error

errors.append(err)

# plot errors

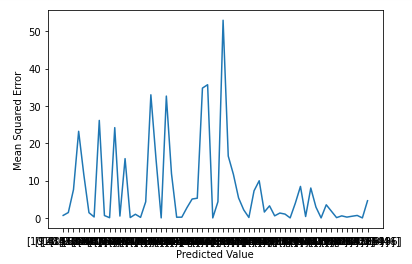
plt.plot(errors)

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

plt.xlabel('Predicted Value')

plt.ylabel('Mean Squared Error')

plt.show()



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

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



rom sklearn import metrics

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

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



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:

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

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

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]))

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

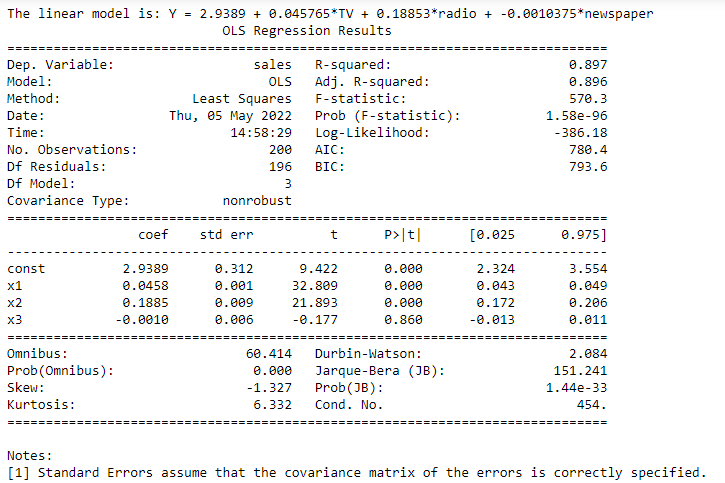
y = data['sales']

X2 = sm.add\_constant(X)

est = sm.OLS(y, X2)

est2 = est.fit()

print(est2.summary())



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 a spam or not.

CODE:

import pandas as pd

from sklearn import metrics

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import math

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

import array as arr

# implement a sigmoid function by hand

def sigmoid(x):

a = []

for item in x:

a.append(1/(1+math.exp(-item)))

return a

# evaluate the sigmoid at some x values

sigm = np.arange(-22, 22, 0.5)

# plot the sigmoid

plt.plot(sigm\*0.2+4.57, np.array(sigmoid(sigm)), color = "red") # manually implemented sigmoid

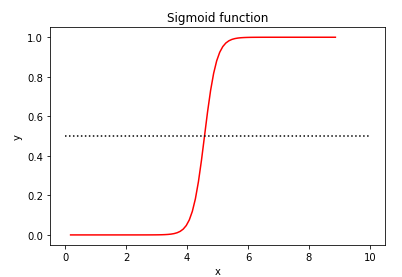
plt.plot([0,10], [0.5, 0.5], linestyle = "dotted", color = "black")

plt.title("Sigmoid function")

plt.xlabel("x")

plt.ylabel("y")

plt.show()



col\_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'outcome'] # load dataset

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

pima.head()

# split data into features/inputs and targets/outputs

feature\_cols = ['pregnant', 'insulin', 'bmi',

'age', 'glucose', 'bp', 'pedigree']

X = pima[feature\_cols] # features

y = pima.outcome # target variable

# split data into training and validation datasets

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

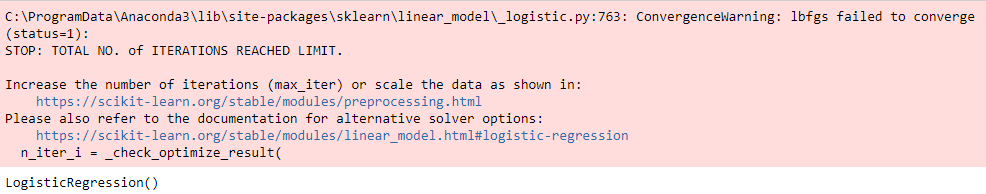
from sklearn.linear\_model import LogisticRegression

# instantiate the model

model = LogisticRegression()

# fitting the model

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

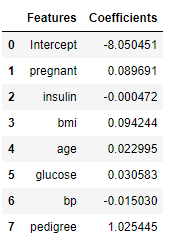


coefficents = {"Features": ["Intercept"] + feature\_cols,

"Coefficients":np.concatenate((model.intercept\_ ,model.coef\_[0]))}

coefficents = pd.DataFrame(coefficents)

coefficents



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

y\_pred[0:5]

#out:

arr.array('i',[1, 0, 0, 1, 0])



# metrics

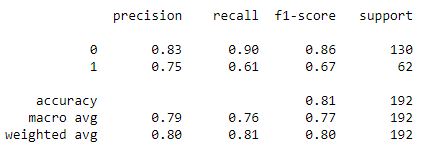
print("Accuracy for test set is {}.".format(round(metrics.accuracy\_score(y\_test, y\_pred), 4)))

print("Precision for test set is {}.".format(round(metrics.precision\_score(y\_test, y\_pred), 4)))

print("Recall for test set is {}.".format(round(metrics.recall\_score(y\_test, y\_pred), 4)))



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



#confusion matrix

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

# plotting the confusion matrix

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

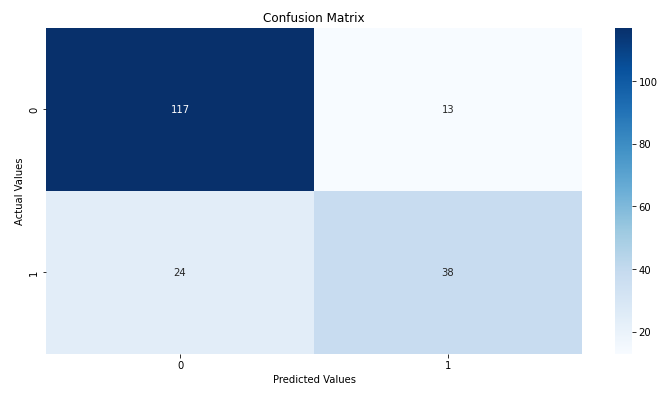
plt.title("Confusion Matrix")

sns.heatmap(conf\_mat, annot=True, fmt='d', cmap='Blues')

plt.ylabel("Actual Values")

plt.xlabel("Predicted Values")

plt.savefig('confusion\_matrix.png')



# ROC curve

y\_pred\_proba = model.predict\_proba(X\_test)[::,1]

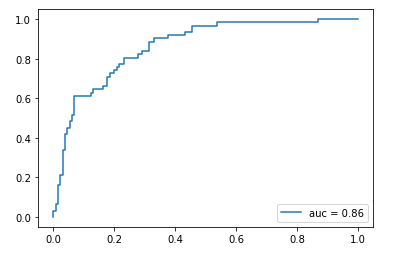
fpr, tpr, \_ = metrics.roc\_curve(y\_test, y\_pred\_proba)

auc = metrics.roc\_auc\_score(y\_test, y\_pred\_proba)

plt.plot(fpr, tpr, label="auc = " + str(round(auc,2)))

plt.legend(loc=4)

plt.show()



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

CODE:

import pandas as pd

from matplotlib import pyplot as plt

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

from sklearn.datasets import load\_breast\_cancer

df = load\_breast\_cancer()

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

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)

X\_test.shape

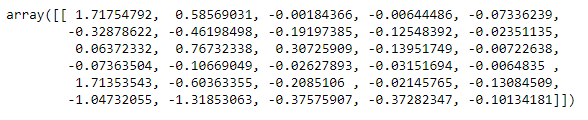


from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(C=100)

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

model.coef\_

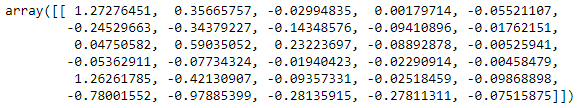


from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(C=1)

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

model.coef\_



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

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:

from numpy import\*

import pandas as pd

from sklearn.datasets import load\_breast\_cancer

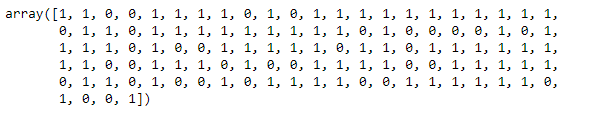
from sklearn.neural\_network import MLPClassifier

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

df = load\_breast\_cancer()

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

y\_test



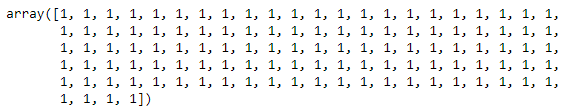
nn=MLPClassifier(activation='logistic',solver='sgd',hidden\_layer\_sizes=(10,15),random\_state=1)

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



pred=nn.predict(X\_test)

pred



from sklearn.metrics import accuracy\_score

accuracy\_score(y\_test,pred)



17. Implement Simple Linear Regression using Least Squared Method.

CODE:

import numpy as np

import pandas as pd

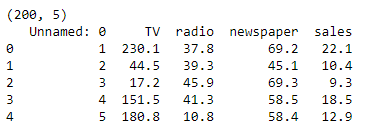
import matplotlib.pyplot as plt

# Reading Data

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

print(data.shape)

print(data.head())



# Computing X and Y

X = data['radio'].values

Y = data['sales'].values

# Mean X and Y

mean\_x = np.mean(X)

mean\_y = np.mean(Y)

# Total number of values

n = len(X)

# Using the formula to calculate 'm' and 'c'

numer = 0

denom = 0

for i in range(n):

numer += (X[i] - mean\_x) \* (Y[i] - mean\_y)

denom += (X[i] - mean\_x) \*\* 2

m = numer / denom

c = mean\_y - (m \* mean\_x)

# Printing coefficients

print("Coefficients")

print(m, c)



# Plotting Values and Regression Line

max\_x = np.max(X) + 100

min\_x = np.min(X) - 100

# Calculating line values x and y

x = np.linspace(min\_x, max\_x, 1000)

y = c + m \* x

# Ploting Line

plt.plot(x, y, color='#58b970', label='Regression Line')

# Ploting Scatter Points

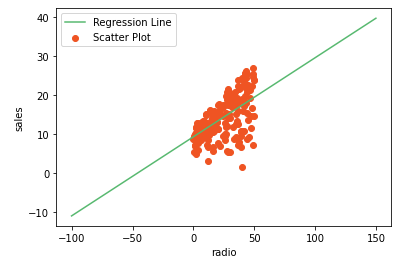
plt.scatter(X, Y, c='#ef5423', label='Scatter Plot')

plt.xlabel('radio')

plt.ylabel('sales')

plt.legend()

plt.show()



# Calculating Root Mean Squares Error

rmse = 0

for i in range(n):

y\_pred = c + m \* X[i]

rmse += (Y[i] - y\_pred) \*\* 2

rmse = np.sqrt(rmse/n)

print("RMSE")

print(rmse)



# Calculating R2 Score

ss\_tot = 0

ss\_res = 0

for i in range(n):

y\_pred = c + m \* X[i]

ss\_tot += (Y[i] - mean\_y) \*\* 2

ss\_res += (Y[i] - y\_pred) \*\* 2

r2 = 1 - (ss\_res/ss\_tot)

print("R2 Score")

print(r2)



18. Multiple Linear Regression using Normal Equation method.

CODE:

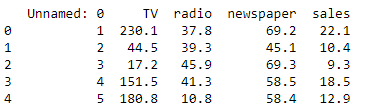
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

print(data.head())



#print column names

print(data.columns)



#read the columns into variables

x1 = data["TV"]

x2 = data["radio"]

x3 = data["newspaper"]

y = data["sales"]

#shape of our variables

print(x1.shape)

print(x2.shape)

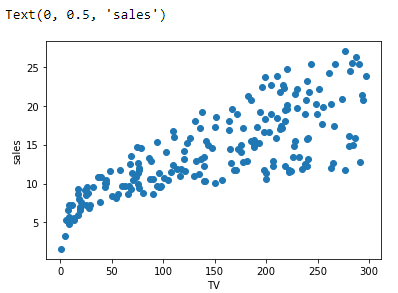
print(x3.shape)



plt.scatter(x1, y)

plt.xlabel("TV")

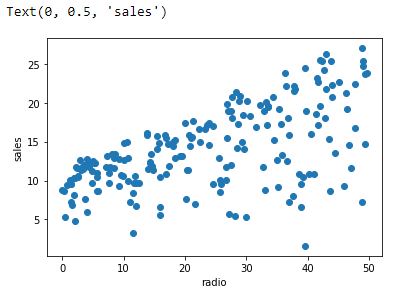
plt.ylabel("sales")



plt.scatter(x2, y)

plt.xlabel("radio")

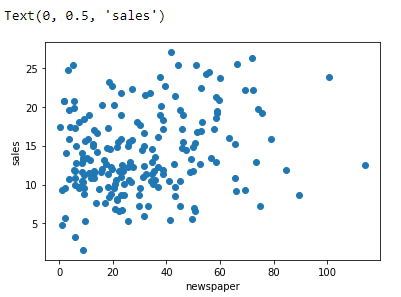
plt.ylabel("sales")



plt.scatter(x3, y)

plt.xlabel("newspaper")

plt.ylabel("sales")



#convert our variables datatype from series to array

x1 = np.array(x1)

x2 = np.array(x2)

x3 = np.array(x3)

#no of rows

n = len(x1)

n



#create a "ones" matrix

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

#reshaping to perform addition and multiplication

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

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

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

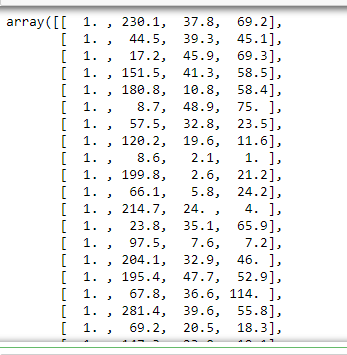
#create major matrix

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



#find transpose

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

#find dot product

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

#find inverse

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

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

#find coeffs

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

theta



#print coeff values

beta\_0 = theta[0]

beta\_1 = theta[1]

beta\_2 = theta[2]

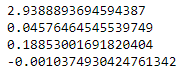
beta\_3 = theta[3]

print(beta\_0)

print(beta\_1)

print(beta\_2)

print(beta\_3)



#prediction function:

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

#predicting a value

TV = 40.2

radio = 33.6

newspaper = 60.5

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

