Practical File
MACHINE LEARNING

PRACHI AGGARWAL

—

University Roll No-21033570042

Roll NO.-21570015

BSc. (H) Computer Science

—

Submitted To

Dr. Nidhi Arora

Q1. Perform elementary mathematical operations in python like addition, multiplication, division and exponentiation.

```
num1 = int(input("Enter first number"))
num2 = int(input("Enter second number"))
print("ADDITION")
print(num1+num2)
print("MULTIPLICATION")
print(num1*num2)
print("DIVISION")
print(num1/num2)
print("EXPONENTIATION")
print(num1**num2)
```

```
Enter first number6
Enter second number7
ADDITION
13
MULTIPLICATION
42
DIVISION
0.8571428571428571
EXPONENTIATION
279936
```

Q2. Perform elementary logical operations in python like OR, AND, Checking for eqality, NOT, XOR

```
num1 = int(input("Enter first number"))
num2 = int(input("Enter second number"))
print("num1 OR num2 = ", num1|num2)
print("num1 AND num2 = ", num1&num2)
print("NOT num1", ~num1)
print("num1 XOR num2 = ", num1^num2)
print("**CHECKING FOR EQALITY**")
print("num1==num2 => ", num1==num2)
print("num1>=num2 => ", num1>=num2)
print("num1<=num2 => ", num1<=num2)
print("num1!=num2 => ", num1!=num2)
```

```
Enter first number6
Enter second number5
num1 OR num2 =  7
num1 AND num2 =  4
NOT num1 -7
num1 XOR num2 =  3
**CHECKING FOR EQALITY**
num1==num2 =>  False
num1>=num2 =>  True
num1<=num2 =>  False
num1!=num2 =>  True
```

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

```
#Initialization and creation of variables and strings
num = 3
str = "Hello 6th semester"
print("First way")
print(f"{num} and {str}")
print("Second way")
print("{0} and {1}".format(num,str))
```

```
First way
3 and Hello 6th semester
```

```
Second way
3 and Hello 6th semester
```

Q4. Create and define single dimension multi-dimenstion arrays, and arrays with specific values like array of all ones,all zeros, array with random values within a range, or a diagonal matrix.

```
import numpy as np
#1D array
arr1 = np.array([1,2,3,4,5])
print("1D array = ", arr1)
#2D array
arr2 = np.array([[1,2,3],[4,5,6]])
print("2D array = ")
print(arr2)
#3D array
num3 = np.array([[[1,2,3],[4,5,6]],[[6,7,8],[1,5,7]]])
print("3D array = ")
print(num3)
#Arrays with all ones
arr1 = np.ones(5)
print("1D array with all ones ",arr1)
arr2 = np.ones((2,3))
print("2D array with all ones ")
print(arr2)
arr3 = np.ones((2,2,3))
print("3D array with all ones ")
print(arr3)
#Arrays with all zeros
arr1 = np.zeros(5)
print("1D array with all zeros ",arr1)
arr2 = np.zeros((2,3))
print("2D array with all zeros ")
print(arr2)
arr3 = np.zeros((2,2,3))
print("3D array with all zeros ")
print(arr3)
#Arrays with random values
arr1 = np.random.randint(5,size=5)
print("1D array with random values: ", arr1)
arr2 = np.random.randint(6,size=(2,3))
print("2D array with random values ")
print(arr2)
arr3 = np.random.randint(12,size=(2,2,3))
print("3D array with random values ")
print(arr3)
#Diagonal matrix
diagonal = np.diag(arr1)
print("Diagonal matrix from array 1 is: ")
print(diagonal)
```

```
1D array =  [1 2 3 4 5]
2D array = 
[[1 2 3]
 [4 5 6]]
3D array = 
[[[1 2 3]
  [4 5 6]]

 [[6 7 8]
  [1 5 7]]]
1D array with all ones  [1. 1. 1. 1. 1.]
2D array with all ones 
[[1. 1. 1.]
 [1. 1. 1.]]
3D array with all ones 
```

```
[[[1. 1. 1.]
  [1. 1. 1.]]

 [[1. 1. 1.]
  [1. 1. 1.]]]
1D array with all zeros  [0. 0. 0. 0. 0.]
2D array with all zeros
[[0. 0. 0.]
 [0. 0. 0.]]
3D array with all zeros
[[[0. 0. 0.]
  [0. 0. 0.]]

 [[0. 0. 0.]
  [0. 0. 0.]]]
1D array with random values:  [4 2 2 4 3]
2D array with random values
[[3 0 1]
 [0 2 5]]
3D array with random values
[[[ 4  6 11]
  [ 6  7  3]]

 [[ 5  6  3]
  [ 7  6  0]]]
Diagonal matrix from array 1 is:
[[4 0 0 0 0]
 [0 2 0 0 0]
 [0 0 2 0 0]
 [0 0 0 4 0]
 [0 0 0 0 3]]
```

Q5. Use command to compute the size of the matrix, size/length of a particular row/column, load data from a text file, store matrix data to a text file, finding out variable and their features in the current scope.

```
import pandas as pd
import numpy as np
#2D array
arr2 = np.array([[1,2,3],[4,5,6]])
print("2D array = ")
print(arr2)
#Size of the matrix
print("Size of the matrix(2D array is): ",arr2.shape)
print("Number of rows: ", arr2.shape[0])
print("Number of columns: ",arr2.shape[1])
#Reading a text file
file = np.loadtxt('/content/matrices.txt')
print(file)
#Storing matrix in text file
arr=np.array(np.random.randint(0,100,size=(2,3)))
#arr = np.array([[1,2,3],[4,5,6]])
np.savetxt('/content/matrices.txt',arr)
#features of local variables
t = 10
print(arr)
print(locals())
```

```
2D array =
[[1 2 3]
 [4 5 6]]
Size of the matrix(2D array is):  (2, 3)
Number of rows:  2
Number of columns:  3
[[72. 20. 40.]
 [14. 32. 53.]]
[[35 78 70]
 [15 27 10]]
```

```
{'__name__': '__main__', '__doc__': 'Automatically created module for IPython interactive environment',
       [4, 5, 6]]), '_i2': 'import pandas as pd\nimport numpy as np\n#2D array\narr2 = np.array([[1,2,3
       [14., 32., 53.]]), 'arr': array([[35, 78, 70],
       [15, 27, 10]]), 't': 10, '_i3': 'import pandas as pd\nimport numpy as np\n#2D array\narr2 = np.a
```

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

```
import numpy as np
arr1 = np.array([[1,2,3],[4,5,6]])
arr2 = np.array([[7,8,9],[10,11,12]])
print("Matrix 1 ")
print(arr1)
print("Matrix2")
print(arr2)
print("Addition of two matrices")
print(arr1 + arr2)
print("Subtraction of two matrices")
print(arr1 - arr2)
print("Multiplication of two matrices")
print(arr1 * arr2)
print('2nd row of matrix 1')
print(arr1[1:,:])
print('3rd column of matrix 2')
print(arr2[:,2:])
```

```
Matrix 1
[[1 2 3]
 [4 5 6]]
Matrix2
[[ 7  8  9]
 [10 11 12]]
Addition of two matrices
[[ 8 10 12]
 [14 16 18]]
Subtraction of two matrices
[[-6 -6 -6]
 [-6 -6 -6]]
Multiplication of two matrices
[[ 7 16 27]
 [40 55 72]]
2nd row of matrix 1
[[4 5 6]]
3rd column of matrix 2
[[ 9]
 [12]]
```

Q7. 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

```
import numpy as np
arr = np.array([[1,2,-3],[4,-5,6]])
print("Original matrix is : ")
print(arr)
print("Absolute values: ")
print(np.absolute(arr))
print("Negative matrix")
print(-arr)
print("After adding new row ")
arr=np.append(arr,[[7,8,9]], axis=0)
print(arr)
print("After adding new column")
arr=np.append(arr,[[11],[12],[13]],axis=1)
print(arr)
print("After deleting last row")
arr = np.delete(arr,2,0)
print(arr)
print("After deleting last column")
for row in arr:
  for item in row[:-1]:
    arr[item]=
# arr = [[item for item in row[:-1]] for row in arr]
print(arr)
```

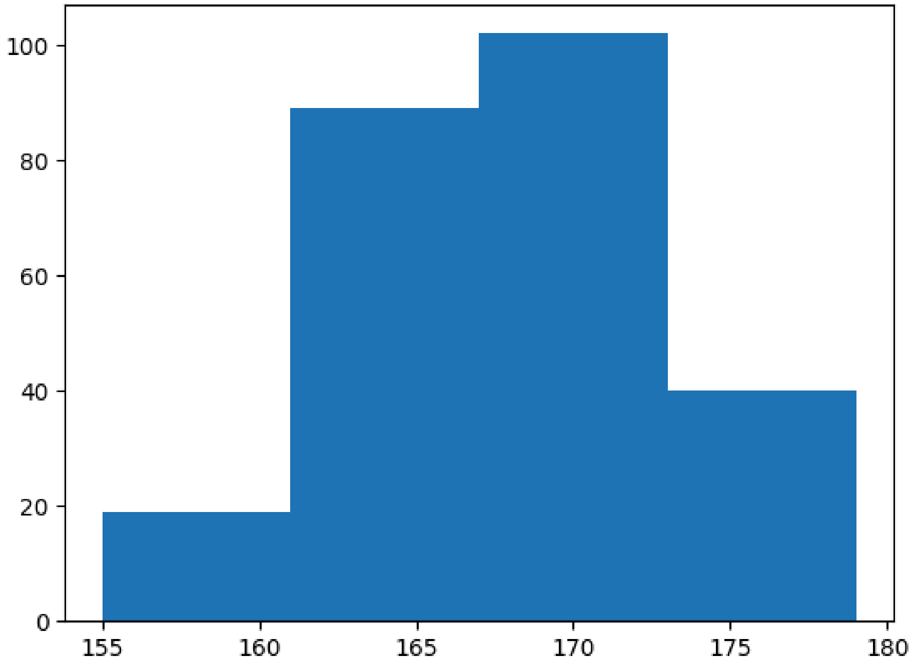
```
Original matrix is :
[[ 1  2 -3]
 [ 4 -5  6]]
Absolute values:
[[1 2 3]
 [4 5 6]]
Negative matrix
[[-1 -2  3]
 [-4  5 -6]]
After adding new row
[[ 1  2 -3]
 [ 4 -5  6]
 [ 7  8  9]]
After adding new column
[[ 1  2 -3 11]
 [ 4 -5  6 12]
 [ 7  8  9 13]]
After deleting last row
[[ 1  2 -3 11]
 [ 4 -5  6 12]]
After deleting last column
[[1, 2, -3], [4, -5, 6]]
```

Q8. Create various type of plots/charts like histograms, plot based on sin/cosin funciton based on data from a matrix. Further label different axes in a plot and data in a plot.

```
import numpy as np
arr = np.array([[1,2,-3],[4,-5,6]])
print("Original matrix is : ")
print(arr)
print("Absolute values: ")
print(np.absolute(arr))
print("Negative matrix")
print(-arr)
print("After adding new row ")
arr=np.append(arr,[[7,8,9]], axis=0)
print(arr)
print("After adding new column")
arr=np.append(arr,[[11],[12],[13]],axis=1)
print(arr)
print("After deleting last row")
arr = np.delete(arr,2,0)
print(arr)
print("After deleting last column")
arr = [[item for item in row[:-1]] for row in arr]
print(arr)
```

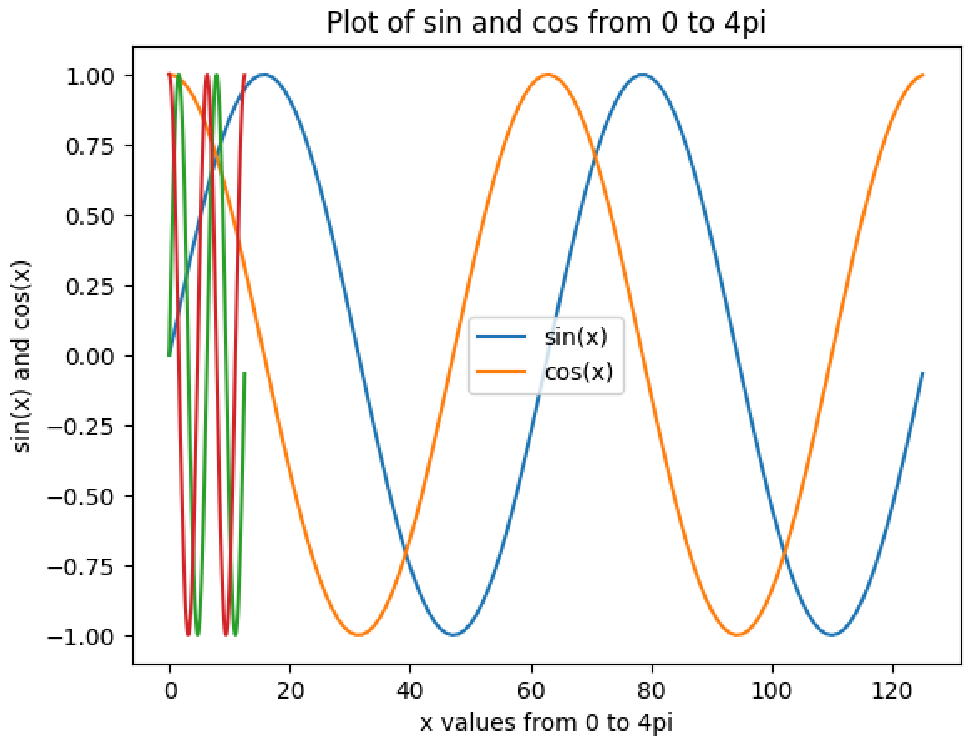
```
Original matrix is :
[[ 1  2 -3]
 [ 4 -5  6]]
Absolute values:
[[1 2 3]
 [4 5 6]]
Negative matrix
[[-1 -2  3]
 [-4  5 -6]]
After adding new row
[[ 1  2 -3]
 [ 4 -5  6]
 [ 7  8  9]]
After adding new column
[[ 1  2 -3 11]
 [ 4 -5  6 12]
 [ 7  8  9 13]]
After deleting last row
[[ 1  2 -3 11]
 [ 4 -5  6 12]]
After deleting last column
[[1, 2, -3], [4, -5, 6]]
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
mu = 168 #mean
sigma = 5 #stddev
sample = 250
np.random.seed(0)
height_f = np.random.normal(mu, sigma, sample).astype(int)

mu = 176 #mean
sigma = 6 #stddev
sample = 250
np.random.seed(1)
height_m = np.random.normal(mu, sigma, sample).astype(int)

plt.hist(height_f,bins=4)
plt.show()
```



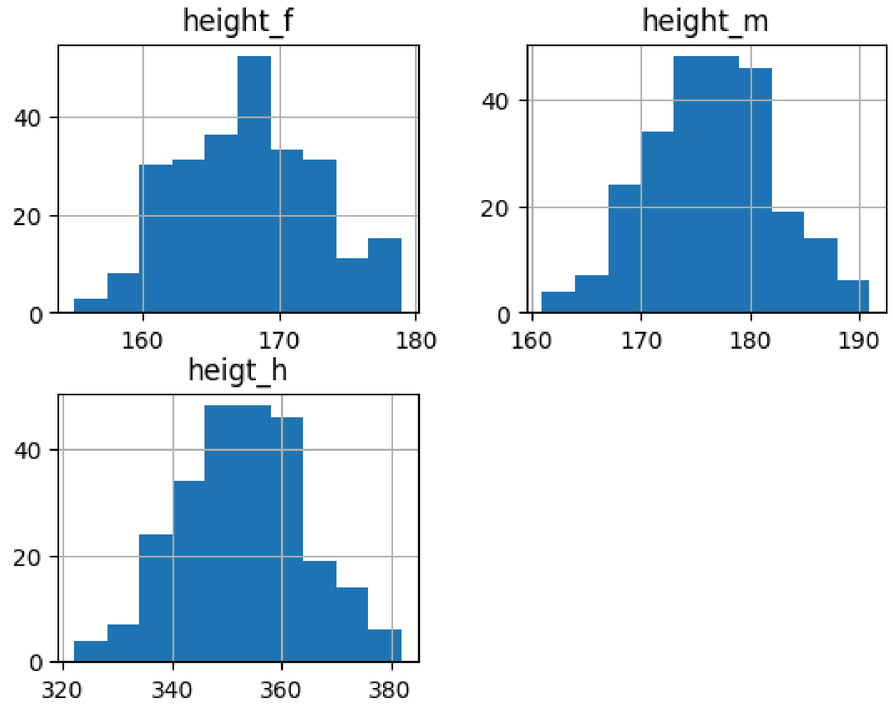
```
x = np.arange(0,4*np.pi,0.1)
y = np.sin(x)
z = np.cos(x)
print (x)
plt.plot(y)
plt.plot(z)
plt.plot(x,y,x,z)
plt.xlabel('x values from 0 to 4pi')  # string must be enclosed with quotes '  '
plt.ylabel('sin(x) and cos(x)')
plt.title('Plot of sin and cos from 0 to 4pi')
plt.legend(['sin(x)', 'cos(x)'])
plt.show()
```

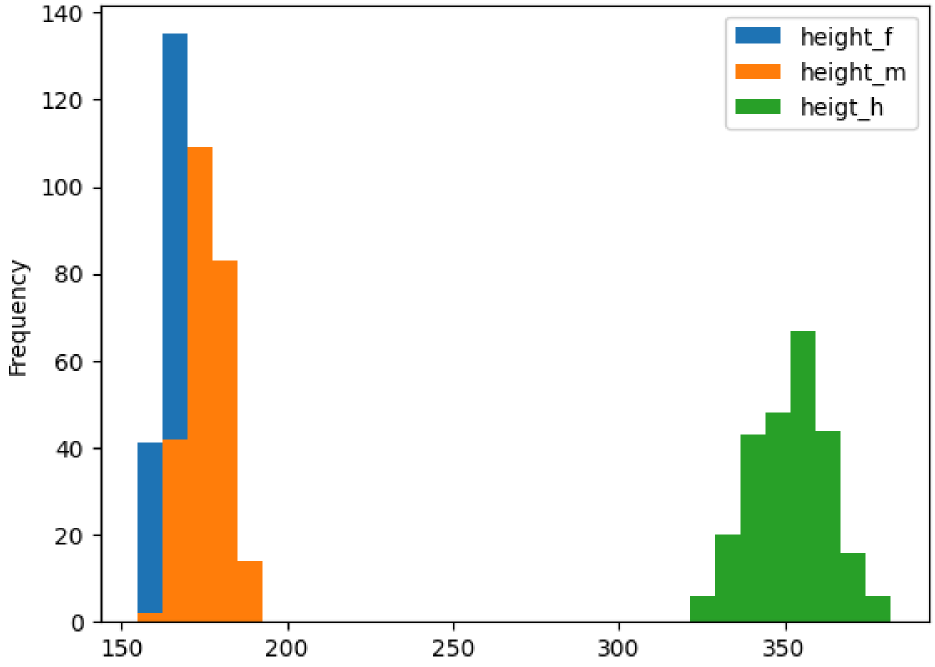
```
[ 0.   0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1.   1.1  1.2  1.3
  1.4  1.5  1.6  1.7  1.8  1.9  2.   2.1  2.2  2.3  2.4  2.5  2.6  2.7
  2.8  2.9  3.   3.1  3.2  3.3  3.4  3.5  3.6  3.7  3.8  3.9  4.   4.1
  4.2  4.3  4.4  4.5  4.6  4.7  4.8  4.9  5.   5.1  5.2  5.3  5.4  5.5
  5.6  5.7  5.8  5.9  6.   6.1  6.2  6.3  6.4  6.5  6.6  6.7  6.8  6.9
  7.   7.1  7.2  7.3  7.4  7.5  7.6  7.7  7.8  7.9  8.   8.1  8.2  8.3
  8.4  8.5  8.6  8.7  8.8  8.9  9.   9.1  9.2  9.3  9.4  9.5  9.6  9.7
  9.8  9.9 10.  10.1 10.2 10.3 10.4 10.5 10.6 10.7 10.8 10.9 11.  11.1
 11.2 11.3 11.4 11.5 11.6 11.7 11.8 11.9 12.  12.1 12.2 12.3 12.4 12.5]
```



```
mu = 168 #mean
sigma = 5 #stddev
sample = 250
np.random.seed(0)
height_f = np.random.normal(mu, sigma, sample).astype(int)

mu = 176 #mean
sigma = 6 #stddev
sample = 250
np.random.seed(1)
height_m = np.random.normal(mu, sigma, sample).astype(int)

gym = pd.DataFrame({'height_f': height_f, 'height_m': height_m,'heigt_h':height_m*2})
print (gym)
gym.hist()
gym.plot.hist(bins=30)
plt.show()
```

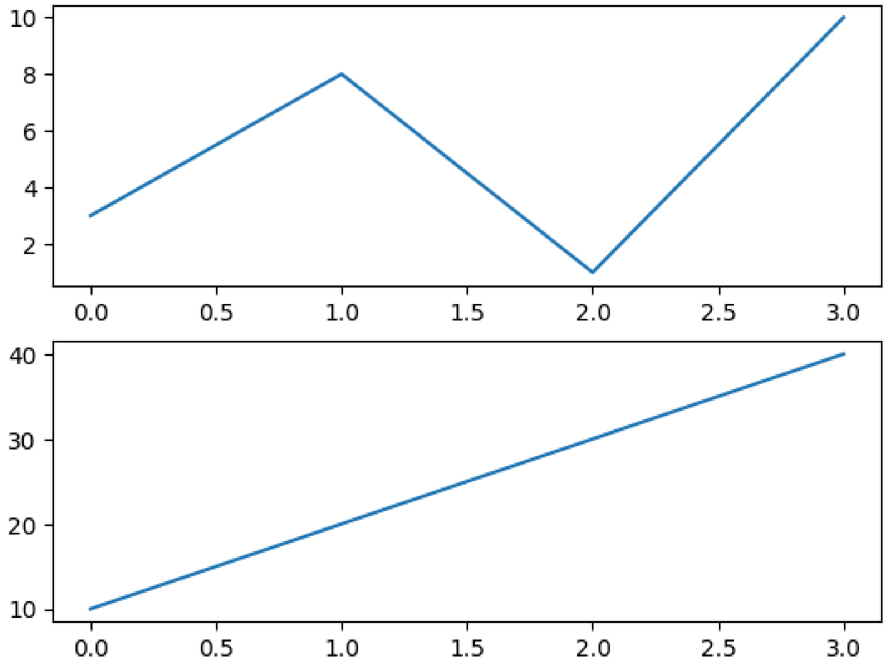
```
     height_f  height_m  heigt_h
0         176       185      370
1         170       172      344
2         172       172      344
3         179       169      338
4         177       181      362
..        ...       ...      ...
245       159       180      360
246       173       179      358
247       173       173      346
248       163       186      372
249       160       168      336

[250 rows x 3 columns]
```



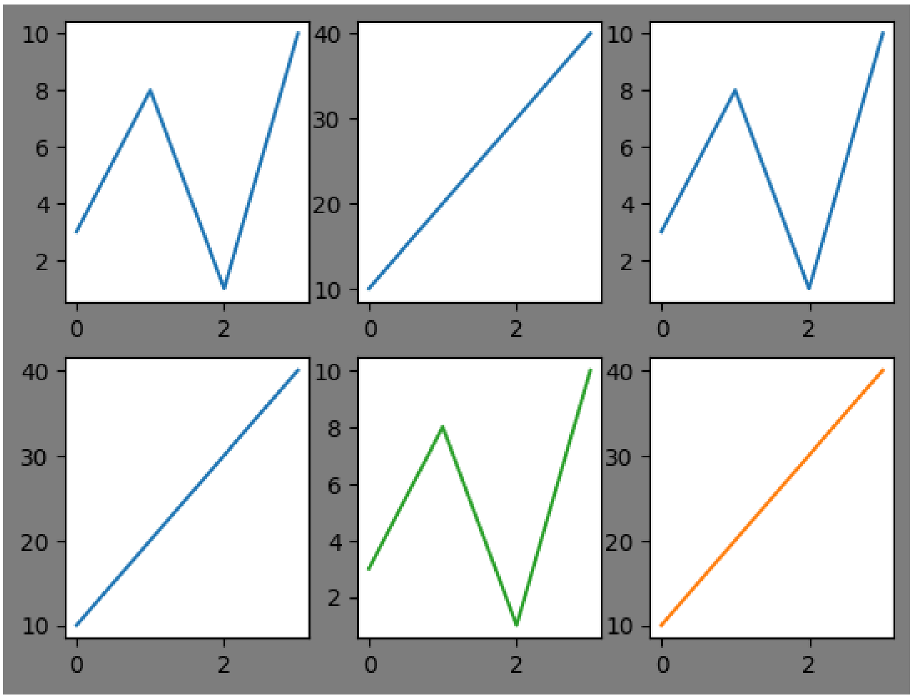


Q9. Generate different subplots from a given plot and color plot data

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 1, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 1, 2)
plt.plot(x,y)

plt.show()
```



```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.figure(facecolor='gray')


plt.subplot(2, 3, 1)

plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 2)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 5)
plt.plot(x,y,'tab:green')

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 6)
plt.plot(x,y,'tab:orange')



plt.show()
```



Q10. Use conditional statements and different type of loops based on simple examples.

> Add blockquote

Example1. Factorial

```
def factorial(n):
  fact = 1
  if n==0 :
    return 1
  else:
    for i in range(n):
      fact *= i+1
  return fact

print(factorial(0))
print(factorial(1))
print(factorial(5))
print(factorial(11))
```

```
1
1
120
39916800
```

Example2. Fibonnacci

```
def fibonaci(n):
  x=0
  y=1
  print(x,y,end=" ")
  for i in range(n):
    z = x+y
    print(z,end=" ")
    x=y
    y=z
fibonaci(10)
```

```
0 1 1 2 3 5 8 13 21 34 55 89
```

Example3. Enter a number and check if it is prime or not if it is prime then print the square of the number and if it is non-prime then print cube of the number

```
def prime(num):
  for i in range(2,int(num/2) + 1):
    if num%i==0:
      print("not a prime")
      return 0
  else:
    print("Prime")
    return 1
num = int(input("Enter a number"))
check = prime(num)
if(check==0):
  print(num*num*num)
else:
  print(num*num)
```

```
Enter a number4
not a prime
64
```

Example4. WAP to ask for month number and according to the number print the season correspond to that month.

```
def season(month):
  if(month>=3 and month<=5):
    print("SPRING")
  elif(month>=6 and month<=8):
    print("SUMMER")
  elif(month>=9 and month<=11):
    print("MONSOON")
  elif(month>=12 and month<=2):
    print("WINTER")
mon = int(input("Enter month number 1 to 12 "))
season(mon)
```

```
Enter month number 1 to 12 11
MONSOON
```

Example5. WAP to ask three strings such that: i) Find number of occurance of string 2 in string 1 ii) Replace string 2 in string 1 with string 3

```
def occurance(str1,str2):
  count = 0
  lst = str1.split()
  for i in lst:
    if(i==str2):
      count += 1
  return count

def exchange(str1,str2, str3):
  new_str=""
  lst = str1.split()
  for i in range(len(lst)):
    if(lst[i]==str2):

      new_str=str1.replace(str2,str3)
  return new_str

str1 = input("Enter string 1 ")
str2 = input("Enter string 2 ")

print(occurance(str1,str2))
str3 = input("Enter string 3 which will replace string 2 from string 1: ")
str1 = exchange(str1,str2,str3)
print(str1)
```

```
Enter string 1 hello world, hello duniya
Enter string 2 hello
2
Enter string 3 which will replace string 2 from string 1: bye
bye world, bye duniya
```

Example6. Pattern program:

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

```
def pattern(n):
  for i in range(0,n):
    for j in range(0,i+1):
      print(j+1,end=" ")
    print()
n = int(input("Enter number of rows in pattern : "))
pattern(n)
```

```
Enter number of rows in pattern : 6
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
```

Q11. Perform vectorized implementation of simple matrix operation like finding the transpose of a matrix, adding, subtracting or multiplying two matrices.

```
import numpy as np
matrix1 = [[1,2],[3,4]]
matrix2 = [[4,5],[7,8]]
mul = [[0,0],[0,0]]
print("Matrix1: ")
print(matrix1)
print("Matrix2: ")
print(matrix2)
print("Transpose of matrix1")
print(np.transpose(matrix1))
print("Addition")
print(np.add(matrix1,matrix2))
print("Subtraction")
print(np.subtract(matrix1,matrix2))
print("Multiplication")
for i in range(len(matrix1)):
   for j in range(len(matrix2[0])):
       for k in range(len(matrix2)):
           mul[i][j] += matrix1[i][k] * matrix2[k][j]
print(mul)
```

```
Matrix1:
[[1, 2], [3, 4]]
Matrix2:
[[4, 5], [7, 8]]
Transpose of matrix1
[[1 3]
 [2 4]]
Addition
[[ 5  7]
 [10 12]]
Subtraction
[[-3 -3]
 [-4 -4]]
Multiplication
[[18, 21], [40, 47]]
```

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.

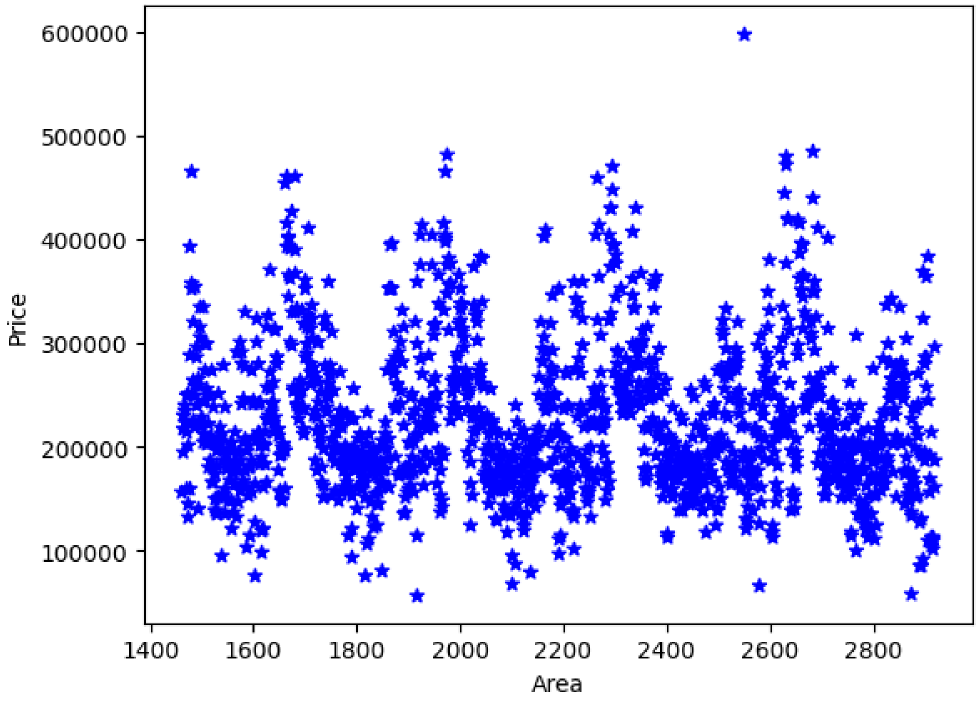
```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
df=pd.read_csv('/houseprice.csv')
df.head()
df.describe()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1459 entries, 0 to 1458
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   area    1459 non-null   int64  
 1   price   1459 non-null   float64
dtypes: float64(1), int64(1)
memory usage: 22.9 KB
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
#scatter plot for dataset
%matplotlib inline
plt.xlabel('Area')
plt.ylabel('Price')
plt.scatter(df.area,df.price,color='blue',marker='*')
```

```
<matplotlib.collections.PathCollection at 0x78b661e93070>
```



```
x_df=df.drop('price',axis='columns')
x_df.head()
x_df
```

```
      area
   0  1461
   1  1462
   2  1463
   3  1464
   4  1465
 ...   ...
1454  2915
1455  2916
1456  2917
1457  2918
1458  2919

1459 rows × 1 columns
```

```
price=df.price
price
```

```
0       156633.3906
1       195762.8959
2       217506.8443
3       230249.3576
4       240294.6583
```

```
           ...
1454    113062.8865
1455    112171.2937
1456    187684.1979
1457    161493.2661
1458    296906.9540
Name: price, Length: 1459, dtype: float64
```

Applying Linear Regression

```
# Create linear regression object
reg=linear_model.LinearRegression()
reg.fit(x_df,price)
```

```
▾ LinearRegression
LinearRegression()
```

```
m=reg.coef_
c=reg.intercept_
print('Coefficent,m=',m)
print('Intercept,c=',c)
```

```
Coefficent,m= [-10.60697754]
Intercept,c= 249516.05775421372
```

```
#Prediction
ans1=reg.predict([[3300]])
print('(1) Price of a house with area = 3300 sqr ft : ',ans1)
```

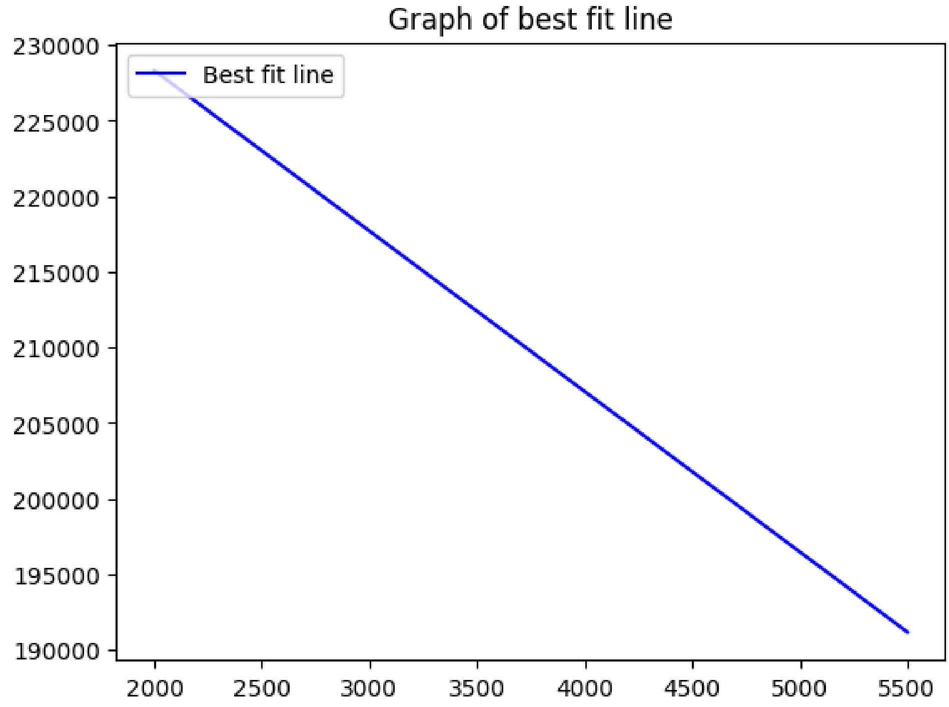
```
(1) Price of a house with area = 3300 sqr ft :  [214513.03186051]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature
  warnings.warn(
```

```
y=m*3300+c
print('y=m*3300+c =',y)
```

```
y=m*3300+c = [214513.03186051]
```

```
#Visualising Best fit line
x=np.linspace(2000,5500)
y=m*x+c
plt.plot(x,y,'-b',label='Best fit line')
plt.legend(loc='upper left')
plt. title('Graph of best fit line')
```

```
Text(0.5, 1.0, 'Graph of best fit line')
```



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.

Double-click (or enter) to edit

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import pandas as pd
# from sklearn import linear_model
import numpy as np
from sklearn import metrics

df=pd.read_csv('/content/houseprice2.csv')
x=df.drop('price',axis=1)
y=df.price
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
len(x_train)
```

```
17290
```

```
len(x_test)
```

```
4323
```

```
len(y_train)
```

```
17290
```

```
len(y_test)
```

```
4323
```

```
model = LinearRegression()
model.fit(x_train, y_train)
r_sq = model.score(x_test, y_test)
print('coefficient of determination= ', r_sq)
m1,m2,m3 = model.coef_
c= model.intercept_
print("intercept: ",model.intercept_)
print(f"Coefficients:\nm1= {m1} \nm2= {m2} \nm3= {m3}")
```

```
coefficient of determination=  0.4289281296057683
intercept:  -1095342.228545767
Coefficients:
m1= 32906.48219921126
m2= 0.11851431502284981
m3= 49726.648097891906
```

```
y_pred = model.predict(x_test)
y_pred
```

```
array([ 648444.49781132,  595566.69433035,  628372.43936179, ...,
        230668.99883437, 1025604.07440111,  628390.6905663 ])
```

```
ans1=model.predict([[3,4300,27]])
print("Price of home with 3 bathrooms, 4300 sq ft and 27 age: ", ans1)
```

```
Price of home with 3 bathrooms, 4300 sq ft and 27 age:  [346506.32824955]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature
  warnings.warn(
```

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.

```
import pandas as pd
from matplotlib import pyplot as plt
%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=0.4, stratify=df.target,rar
X_test.shape
```

```
(228, 30)
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(C=0.1,max_iter=4000)
```

```
model.fit(X_train, y_train)
```

```
▾          LogisticRegression
LogisticRegression(C=0.1, max_iter=4000)
```

```
y_predicted = model.predict(X_test)

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)))
```

```
Accuracy on the training subset: 0.950147
Accuracy on the test subset: 0.956140
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_predicted)
```

```
array([[ 77,   8],
       [  2, 141]])
```

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

```
import pandas as pd
from matplotlib import pyplot as plt
%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=0.3, stratify=df.target,random_state=52)
X_test.shape
```

```
(171, 30)
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(C=100,max_iter=5000)
```

```
model.fit(X_train, y_train)
```

```
▾          LogisticRegression
LogisticRegression(C=100, max_iter=5000)
```

```
model.coef_
```

```
array([[ 6.71831068e+00,  5.54472950e-02, -4.43187749e-01,
        -2.31590394e-02, -4.06899474e+00, -3.20591845e-02,
        -6.20644698e+00, -8.68347463e+00, -2.48047414e+00,
         8.75082493e-01,  2.02567023e-01,  6.68837280e-01,
         8.91561225e-01, -1.05919771e-01, -9.08233642e-01,
         5.23574218e+00,  6.36405260e+00, -7.36998194e-01,
         7.24189297e-01,  1.04306216e+00, -1.65446435e+00,
        -3.52388453e-01, -7.01262895e-02, -3.63675814e-03,
        -8.02661104e+00,  6.17395741e+00, -5.42728440e+00,
        -1.45099167e+01, -2.65996990e+00,  2.00591917e+00]])
```

```
y_predicted = model.predict(X_test)

print('Accuracy on the training subset:',(model.score(X_train, y_train)))
print('Accuracy on the test subset:',(model.score(X_test, y_test)))
```

```
Accuracy on the training subset: 0.9773869346733668
Accuracy on the test subset: 0.9824561403508771
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(C=1,max_iter=5000)
```

```
model.fit(X_train, y_train)
model.coef_
```

```
array([[ 0.73625115,  0.21388556, -0.35626021,  0.03419173, -0.12251383,
        -0.17954031, -0.44068608, -0.20689126, -0.1140189 , -0.03268531,
        -0.05404352,  0.34292718,  0.60236935, -0.09490759, -0.02412106,
         0.00493411, -0.08883045, -0.02635301, -0.01581772,  0.0029014 ,
        -0.20956825, -0.44890794, -0.12893428, -0.01230084, -0.24653726,
        -0.50776345, -1.11315768, -0.3791102 , -0.32229255, -0.0794862 ]])
```

```
y_predicted = model.predict(X_test)

print('Accuracy on the training subset:',(model.score(X_train, y_train)))
print('Accuracy on the test subset:',(model.score(X_test, y_test)))
```

```
Accuracy on the training subset: 0.9673366834170855
Accuracy on the test subset: 0.9590643274853801
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(C=20,max_iter=5000)
model.fit(X_train, y_train)
model.coef_
```

```
array([[ 2.69936145,  0.13324124, -0.31759455,  0.00919426, -1.44169509,
        -0.27046007, -2.96025157, -2.66145008, -0.82755783,  0.01086857,
        -0.1557228 ,  0.48652448,  0.88930616, -0.10863701, -0.36700544,
         1.27386834,  0.77215114, -0.29479471,  0.18575952,  0.26089219,
        -1.15516773, -0.4021781 , -0.03187428, -0.00867762, -2.93457094,
         0.668172  , -5.04762012, -4.22481663, -1.60206599,  0.15424386]])
```

```
y_predicted = model.predict(X_test)

print('Accuracy on the training subset:',(model.score(X_train, y_train)))
print('Accuracy on the test subset:',(model.score(X_test, y_test)))
```

```
Accuracy on the training subset: 0.9748743718592965
Accuracy on the test subset: 0.9766081871345029
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(C=500,max_iter=5000)
model.fit(X_train, y_train)
model.coef_
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
array([[ 7.71865320e+00,  3.11058186e-02, -4.83903186e-01,
        -2.78449920e-02, -8.67271556e+00,  9.61754349e-01,
        -1.10419613e+01, -1.78880502e+01, -4.81773693e+00,
         1.85384269e+00,  7.62086202e-01,  1.25930806e+00,
         7.92112035e-01, -1.40493009e-01, -1.98339869e+00,
         1.15068154e+01,  1.47507212e+01, -1.43990758e+00,
         1.69700610e+00,  2.30182313e+00, -2.30583695e+00,
        -4.04793874e-01, -8.26772481e-02,  1.37642043e-03,
        -1.72655079e+01,  1.30545088e+01, -7.12235664e+00,
        -2.96089508e+01, -6.08996353e+00,  4.25342254e+00]])
```

```
y_predicted = model.predict(X_test)

print('Accuracy on the training subset:',(model.score(X_train, y_train)))
print('Accuracy on the test subset:',(model.score(X_test, y_test)))
```

```
Accuracy on the training subset: 0.9849246231155779
Accuracy on the test subset: 0.9824561403508771
```

```
Start coding or generate with AI.
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(C=1000,max_iter=5000)
model.fit(X_train, y_train)
model.coef_
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
array([[ 9.97795713e+00, -1.09028553e-03, -6.29523501e-01,
        -3.76132222e-02, -1.04962844e+01,  8.81769522e-01,
        -1.34049111e+01, -2.16239062e+01, -5.54814157e+00,
         2.16652384e+00,  7.75083081e-01,  9.74858020e-01,
         1.15593508e+00, -1.61211750e-01, -2.37750919e+00,
         1.39056300e+01,  1.82791815e+01, -1.70030092e+00,
         2.04962091e+00,  2.75779100e+00, -3.42689147e+00,
        -3.95833308e-01, -9.86855173e-02,  9.51799224e-03,
        -2.09114834e+01,  1.55927451e+01, -6.91317899e+00,
        -3.58534660e+01, -6.84731770e+00,  4.92754629e+00]])
```

```
y_predicted = model.predict(X_test)

print('Accuracy on the training subset:',(model.score(X_train, y_train)))
print('Accuracy on the test subset:',(model.score(X_test, y_test)))
```

```
Accuracy on the training subset: 0.9874371859296482
Accuracy on the test subset: 0.9766081871345029
```

FROM THE ABOVE TRIAL OF DIFFERENT VALUES OF C , I GET C=500 GIVE MORE ACCURATE PRIDICTIONS

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

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
```

```
cancer['data'].shape
```

```
(569, 30)
```

```
X = cancer['data']
y = cancer['target']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# Fit only to the training data
scaler.fit(X_train)
```

```
▾ StandardScaler
StandardScaler()
```

```
# transformation to the data
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
# import the estimator(MLP classifier model)
from sklearn.neural_network import MLPClassifier
```

```
mlp = MLPClassifier(hidden_layer_sizes=(30,30,30))
# here is the 3 layers with same number of neurons
```

```
# fit the training data to our model
mlp.fit(X_train,y_train)
```

```
▾                MLPClassifier
MLPClassifier(hidden_layer_sizes=(30, 30, 30))
```

```
predictions = mlp.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,predictions))
```

```
[[56  3]
 [ 2 82]]
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,predictions)
```

```
0.965034965034965
```

```
Start coding or generate with AI.
```