SRI GURU GOBIND SINGH COLLEGE OF COMMERCE
University Of Delhi

MACHINE LEARNING

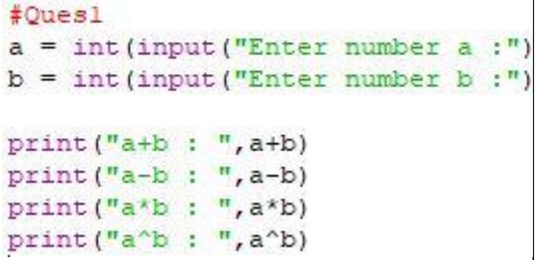
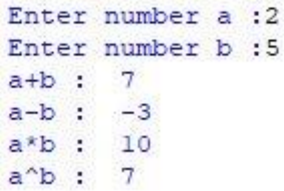
PRACTICAL FILE

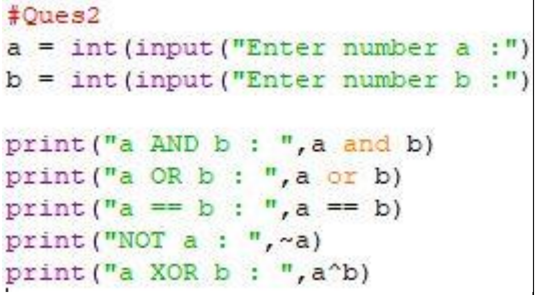
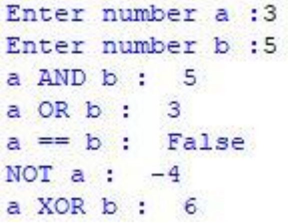
Name :- Garvit Bansal

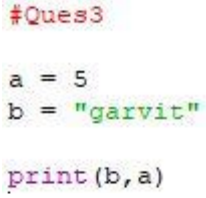
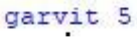
Roll No :- 19078570015

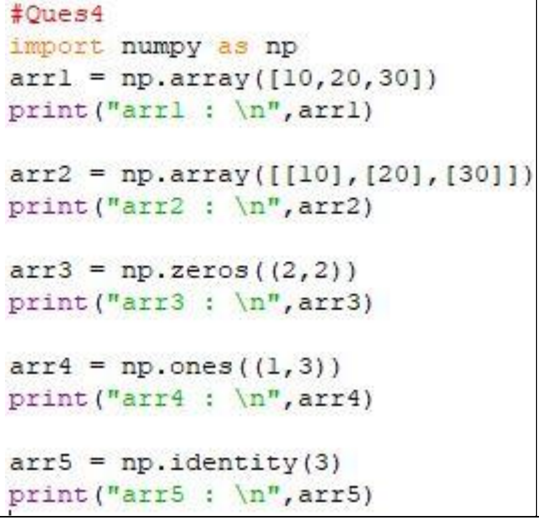
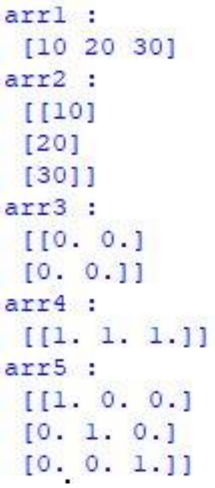
INDEX

Question	Code	Output

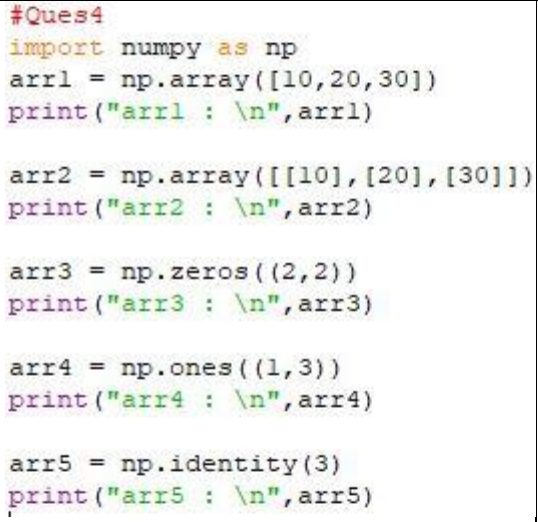
Q1) Perform elementary mathematical operations in Octave/MATLAB like addition, subtraction, multiplication, division and exponentiation.	
	#Ques1
	a = int(input("Enter number a :"))
	b = int(input("Enter number b :"))
	
	print("a+b : ",a+b)
	print("a-b : ",a-b)
	print("a*b : ",a*b)
	print("a^b : ",a^b)
		
		Enter number a :2
		Enter number b :5
		a+b : 7
		a-b : -3
		a*b : 10
		a^b : 7

Q2) Perform elementary logical operations in Octave/MATLAB (like OR, AND, Checking for Equality, NOT, XOR).	
	#Ques2
	a = int(input("Enter number a :"))
	b = int(input("Enter number b :"))
	
	print("a AND b : ",a and b)
	print("a OR b : ",a or b)
	print("a == b : ",a == b)
	print("NOT a : ",~a)
	print("a XOR b : ",a^b)
		
		Enter number a :3
		Enter number b :5
		a AND b : 5
		a OR b : 3
		a == b : False
		NOT a : -4
		a XOR b : 6

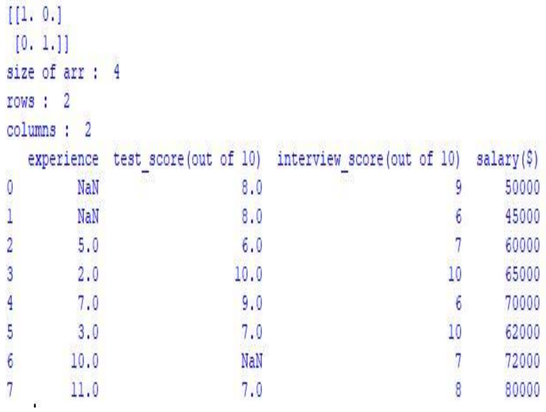
Q3) Create, initialize and display simple variables and simple strings and use simple formatting for variable.	
	#Ques3
	
	a = 5
	b = "garvit"
	
	print(b,a)
		
		garvit 5

Q4) Create/Define single dimension / multi-dimension arrays, and arrays with specific values like array of all ones, all zeros, array with random values within a range, or a diagonal matrix.	
	#Ques4
	import numpy as np
	arr1 = np.array([10,20,30])
	print("arr1 : \n",arr1)
	
	arr2 = np.array([[10],[20],[30]])
	print("arr2 : \n",arr2)
	
	arr3 = np.zeros((2,2))
	print("arr3 : \n",arr3)
	
	arr4 = np.ones((1,3))
	print("arr4 : \n",arr4)
	
	arr5 = np.identity(3)
	print("arr5 : \n",arr5)
		
		arr1 :
		 [10 20 30]
		arr2 :
		 [[10]
		 [20]
		 [30]]
		arr3 :
		 [[0. 0.]
		 [0. 0.]]
		arr4 :
		 [[1. 1. 1.]]
		arr5 :
		 [[1. 0. 0.]
		 [0. 1. 0.]
		 [0. 0. 1.]]

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



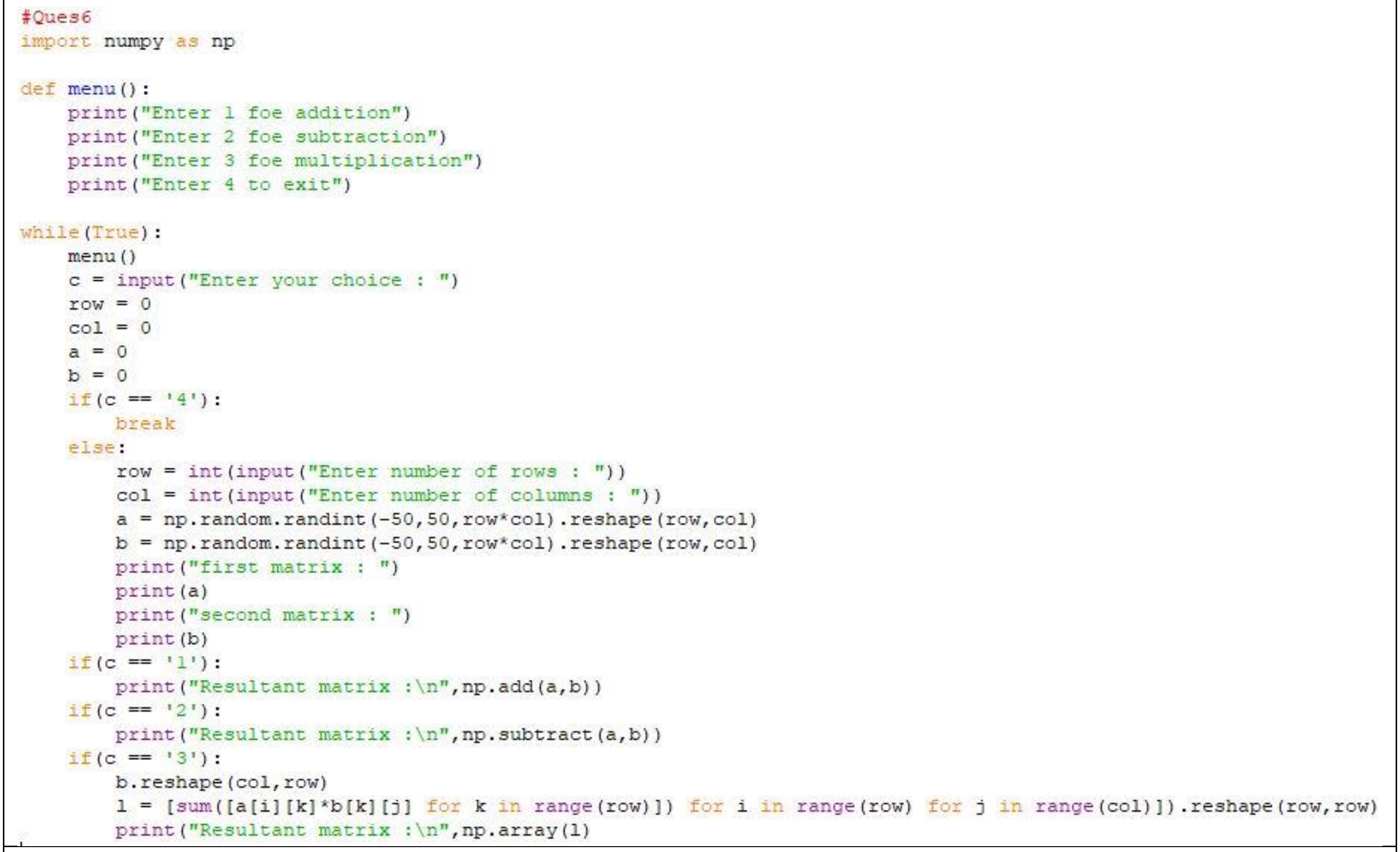
```
#Ques4
import numpy as np
arr1 = np.array([10,20,30])
print("arr1 : \n",arr1)

arr2 = np.array([[10],[20],[30]])
print("arr2 : \n",arr2)

arr3 = np.zeros((2,2))
print("arr3 : \n",arr3)

arr4 = np.ones((1,3))
print("arr4 : \n",arr4)

arr5 = np.identity(3)
print("arr5 : \n",arr5)
```



```
[[1. 0.]
 [0. 1.]]
size of arr :  4
rows :  2
columns :  2
   experience  test_score(out of 10)  interview_score(out of 10)  salary($)
0         NaN                    8.0                           9      50000
1         NaN                    8.0                           6      45000
2         5.0                    6.0                           7      60000
3         2.0                   10.0                          10      65000
4         7.0                    9.0                           6      70000
5         3.0                    7.0                          10      62000
6        10.0                    NaN                           7      72000
7        11.0                    7.0                           8      80000
```

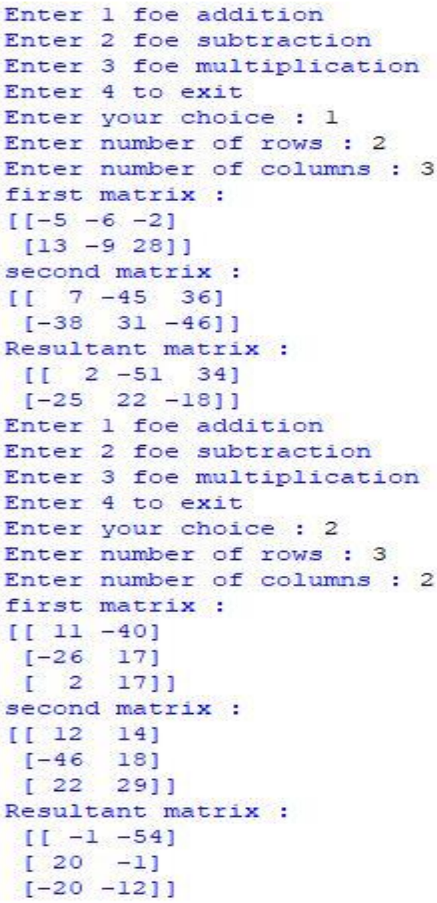
Q6) Perform basic operations on matrices (like addition, subtraction, multiplication) and display specific rows or columns of the matrix.

Code:

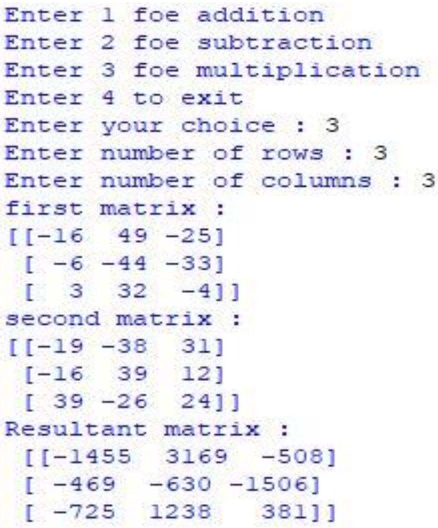


```
#Ques6
import numpy as np

def menu():
    print("Enter 1 foe addition")
    print("Enter 2 foe subtraction")
    print("Enter 3 foe multiplication")
    print("Enter 4 to exit")

while(True):
    menu()
    c = input("Enter your choice : ")
    row = 0
    col = 0
    a = 0
    b = 0
    if(c == '4'):
        break
    else:
        row = int(input("Enter number of rows : "))
        col = int(input("Enter number of columns : "))
        a = np.random.randint(-50,50,row*col).reshape(row,col)
        b = np.random.randint(-50,50,row*col).reshape(row,col)
        print("first matrix : ")
        print(a)
        print("second matrix : ")
        print(b)
    if(c == '1'):
        print("Resultant matrix :\n",np.add(a,b))
    if(c == '2'):
        print("Resultant matrix :\n",np.subtract(a,b))
    if(c == '3'):
        b.reshape(col,row)
        l = [sum([a[i][k]*b[k][j] for k in range(row)]) for i in range(row) for j in range(col)]).reshape(row,row)
        print("Resultant matrix :\n",np.array(l)
```

Output:



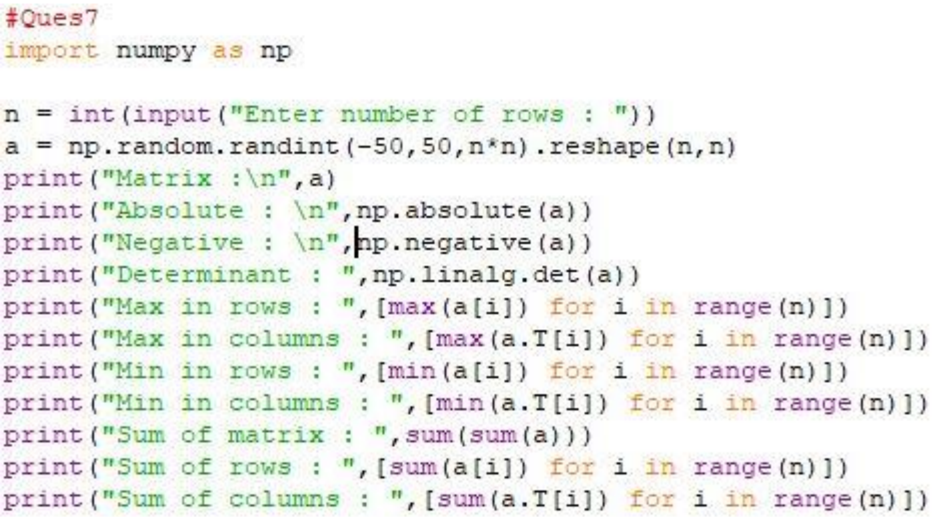
```
Enter 1 foe addition
Enter 2 foe subtraction
Enter 3 foe multiplication
Enter 4 to exit
Enter your choice : 1
Enter number of rows : 2
Enter number of columns : 3
first matrix :
[[-5 -6 -2]
 [13 -9 28]]
second matrix :
[[  7 -45  36]
 [-38  31 -46]]
Resultant matrix :
 [[  2 -51  34]
 [-25  22 -18]]
Enter 1 foe addition
Enter 2 foe subtraction
Enter 3 foe multiplication
Enter 4 to exit
Enter your choice : 2
Enter number of rows : 3
Enter number of columns : 2
first matrix :
[[ 11 -40]
 [-26  17]
 [  2  17]]
second matrix :
[[ 12  14]
 [-46  18]
 [ 22  29]]
Resultant matrix :
 [[ -1 -54]
 [ 20  -1]
 [-20 -12]]
```



```
Enter 1 foe addition
Enter 2 foe subtraction
Enter 3 foe multiplication
Enter 4 to exit
Enter your choice : 3
Enter number of rows : 3
Enter number of columns : 3
first matrix :
[[-16  49 -25]
 [ -6 -44 -33]
 [  3  32  -4]]
second matrix :
[[-19 -38  31]
 [-16  39  12]
 [ 39 -26  24]]
Resultant matrix :
 [[-1455  3169  -508]
 [ -469  -630 -1506]
 [ -725  1238   381]]
```

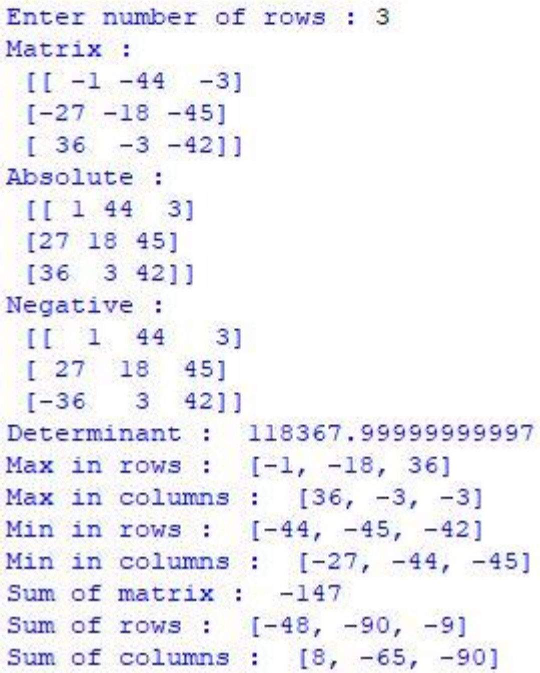
Q7) Perform other matrix operations like converting matrix data to absolute values, taking the negative of matrix values, adding/removing rows/columns from a matrix, finding the maximum or minimum values in a matrix or in a row/column, and finding the sum of some/all elements in a matrix.

Code:



```
#Ques7
import numpy as np

n = int(input("Enter number of rows : "))
a = np.random.randint(-50,50,n*n).reshape(n,n)
print("Matrix :\n",a)
print("Absolute : \n",np.absolute(a))
print("Negative : \n",np.negative(a))
print("Determinant : ",np.linalg.det(a))
print("Max in rows : ",[max(a[i]) for i in range(n)])
print("Max in columns : ",[max(a.T[i]) for i in range(n)])
print("Min in rows : ",[min(a[i]) for i in range(n)])
print("Min in columns : ",[min(a.T[i]) for i in range(n)])
print("Sum of matrix : ",sum(sum(a)))
print("Sum of rows : ",[sum(a[i]) for i in range(n)])
print("Sum of columns : ",[sum(a.T[i]) for i in range(n)])
```

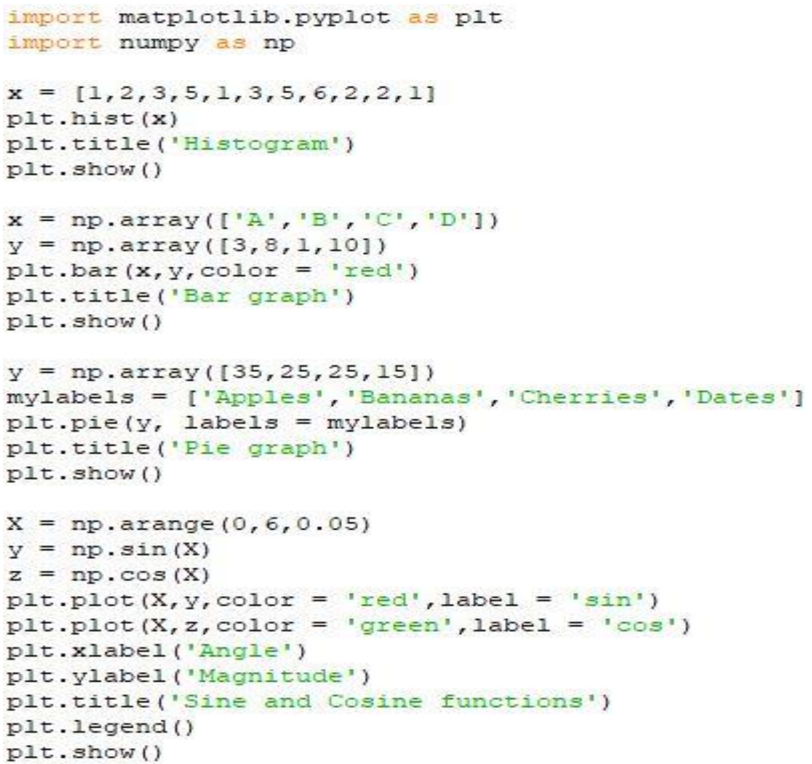
Output:



```
Enter number of rows : 3
Matrix :
 [[ -1 -44  -3]
 [-27 -18 -45]
 [ 36  -3 -42]]
Absolute :
 [[ 1 44  3]
 [27 18 45]
 [36  3 42]]
Negative :
 [[  1  44   3]
 [ 27  18  45]
 [-36   3  42]]
Determinant :  118367.99999999997
Max in rows :  [-1, -18, 36]
Max in columns :  [36, -3, -3]
Min in rows :  [-44, -45, -42]
Min in columns :  [-27, -44, -45]
Sum of matrix :  -147
Sum of rows :  [-48, -90, -9]
Sum of columns :  [8, -65, -90]
```

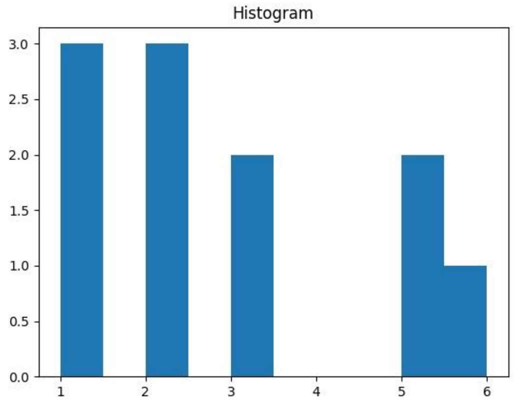
Q8) Create various types of plots/charts like histograms, plot based on sine/cosine function based on data from a matrix. Further label different axes in a plot and data in a plot.

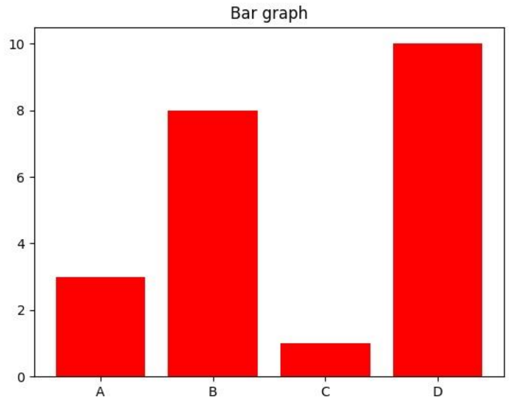
Code:

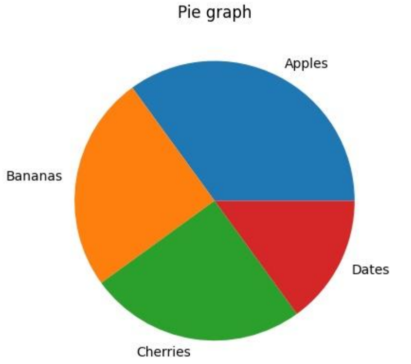


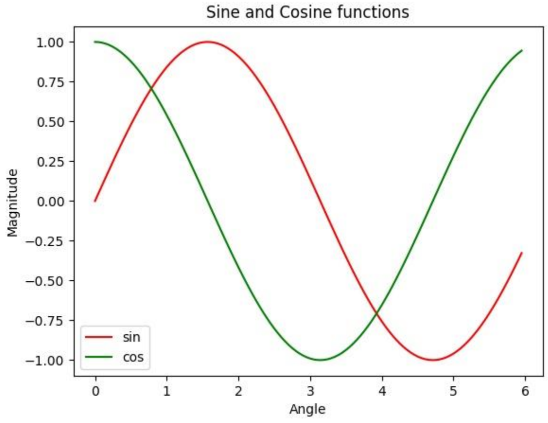
```
import matplotlib.pyplot as plt
import numpy as np

x = [1,2,3,5,1,3,5,6,2,2,1]
plt.hist(x)
plt.title('Histogram')
plt.show()

x = np.array(['A','B','C','D'])
y = np.array([3,8,1,10])
plt.bar(x,y,color = 'red')
plt.title('Bar graph')
plt.show()

y = np.array([35,25,25,15])
mylabels = ['Apples','Bananas','Cherries','Dates']
plt.pie(y, labels = mylabels)
plt.title('Pie graph')
plt.show()

X = np.arange(0,6,0.05)
y = np.sin(X)
z = np.cos(X)
plt.plot(X,y,color = 'red',label = 'sin')
plt.plot(X,z,color = 'green',label = 'cos')
plt.xlabel('Angle')
plt.ylabel('Magnitude')
plt.title('Sine and Cosine functions')
plt.legend()
plt.show()
```

Output:



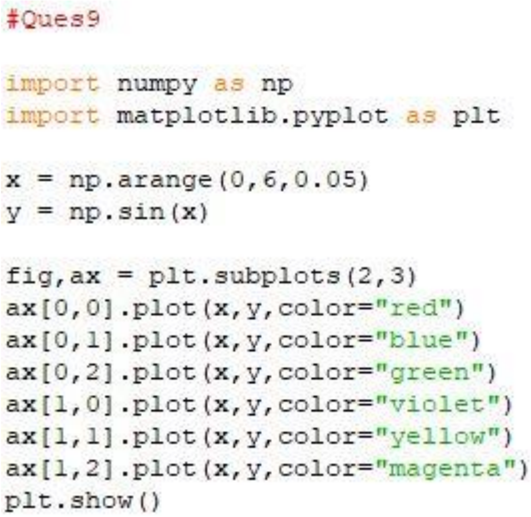






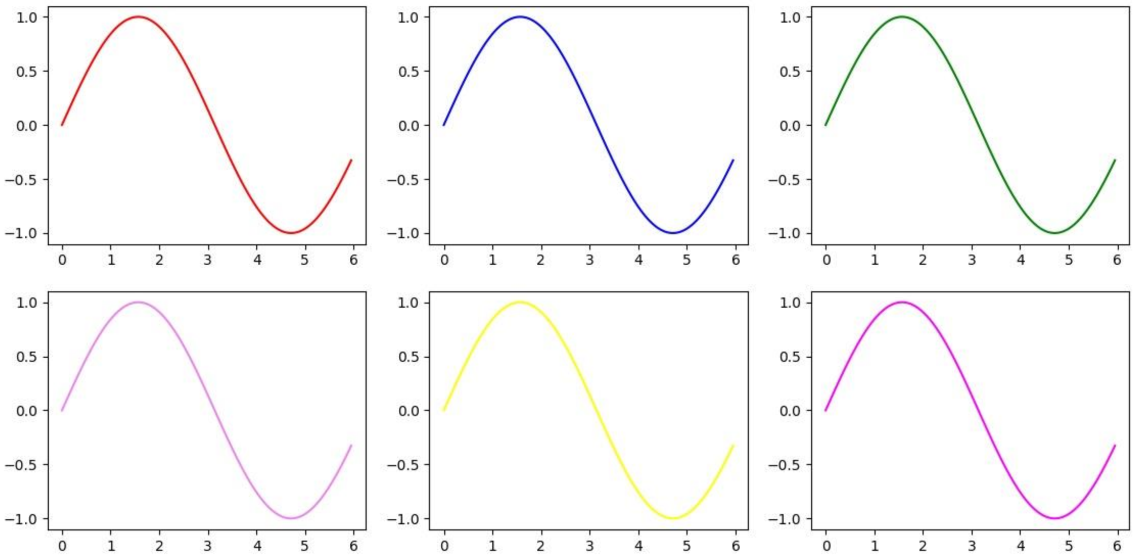
Q9) Generate different subplots from a given plot and color plot data.

Code:



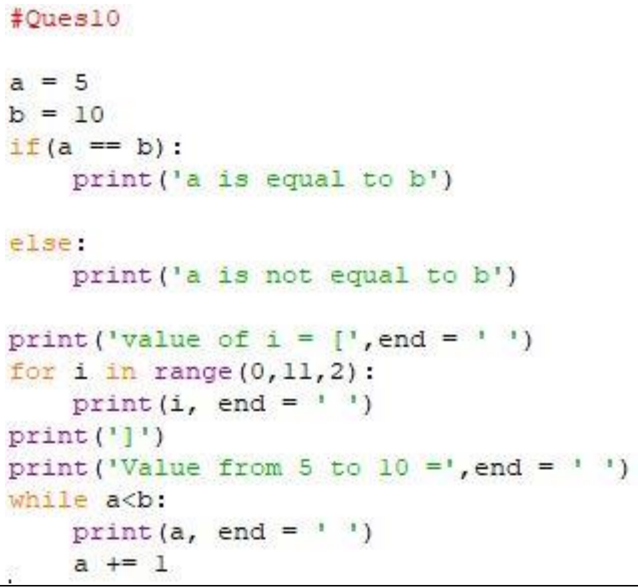
```
#Ques9

import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0,6,0.05)
y = np.sin(x)

fig,ax = plt.subplots(2,3)
ax[0,0].plot(x,y,color="red")
ax[0,1].plot(x,y,color="blue")
ax[0,2].plot(x,y,color="green")
ax[1,0].plot(x,y,color="violet")
ax[1,1].plot(x,y,color="yellow")
ax[1,2].plot(x,y,color="magenta")
plt.show()
```

Outputs:



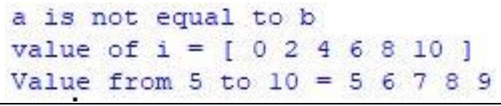
Q10) Use conditional statements and different type of loops based on simple examples.

Code:



```
#Ques10

a = 5
b = 10
if(a == b):
    print('a is equal to b')

else:
    print('a is not equal to b')

print('value of i = [',end = ' ')
for i in range(0,11,2):
    print(i, end = ' ')
print(']')
print('Value from 5 to 10 =',end = ' ')
while a<b:
    print(a, end = ' ')
    a += 1
```

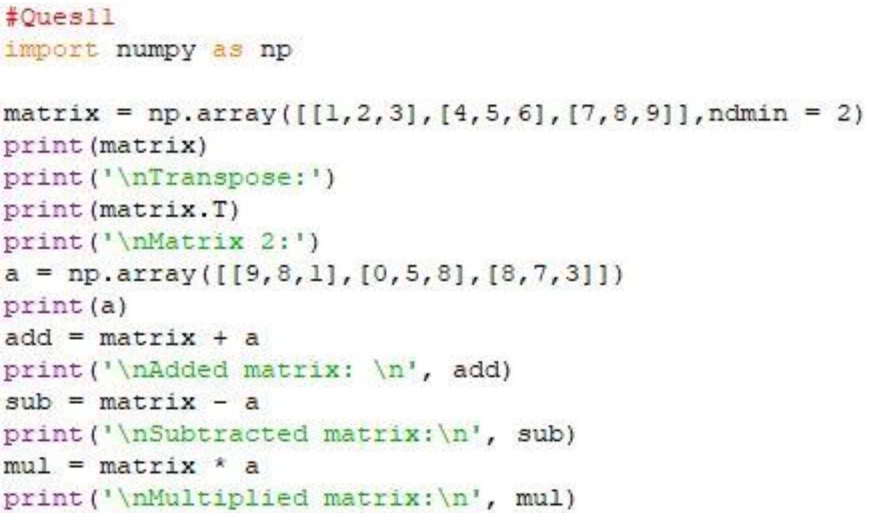
Output:



```
a is not equal to b
value of i = [ 0 2 4 6 8 10 ]
Value from 5 to 10 = 5 6 7 8 9
```

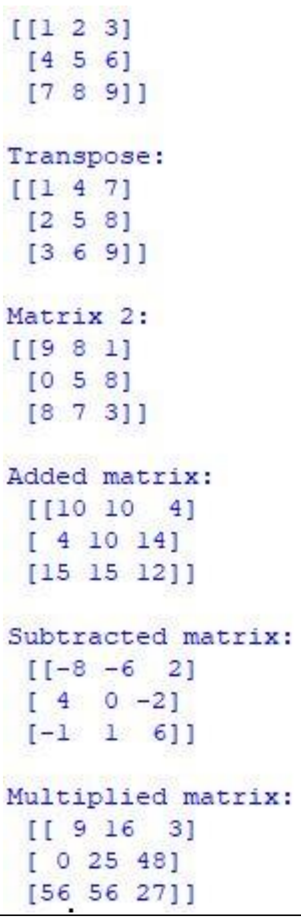
Q11) Perform vectored implementation of simple matrix operation like finding the transpose of a matrix, adding, subtracting or multiplying two matrices.

Code:



```
#Ques11
import numpy as np

matrix = np.array([[1,2,3],[4,5,6],[7,8,9]],ndmin = 2)
print(matrix)
print('\nTranspose:')
print(matrix.T)
print('\nMatrix 2:')
a = np.array([[9,8,1],[0,5,8],[8,7,3]])
print(a)
add = matrix + a
print('\nAdded matrix: \n', add)
sub = matrix - a
print('\nSubtracted matrix:\n', sub)
mul = matrix * a
print('\nMultiplied matrix:\n', mul)
```

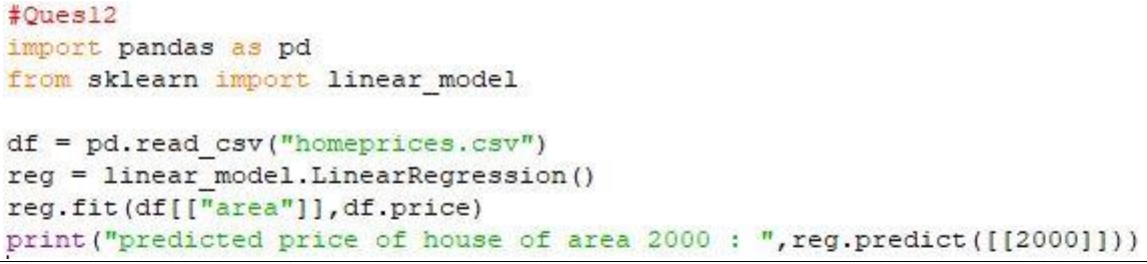
Output:



```
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Transpose:
[[1 4 7]
 [2 5 8]
 [3 6 9]]

Matrix 2:
[[9 8 1]
 [0 5 8]
 [8 7 3]]

Added matrix:
 [[10 10  4]
 [ 4 10 14]
 [15 15 12]]

Subtracted matrix:
 [[-8 -6  2]
 [ 4  0 -2]
 [-1  1  6]]

Multiplied matrix:
 [[ 9 16  3]
 [ 0 25 48]
 [56 56 27]]
```

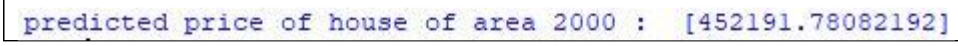
Q12) Implement Linear Regression problem. For example, based on a dataset comprising of existing set of prices and area/size of the houses, predict the estimated price of a given house.

Code:



```
#Ques12
import pandas as pd
from sklearn import linear_model

df = pd.read_csv("homeprices.csv")
reg = linear_model.LinearRegression()
reg.fit(df[["area"]],df.price)
print("predicted price of house of area 2000 : ",reg.predict([[2000]]))
```

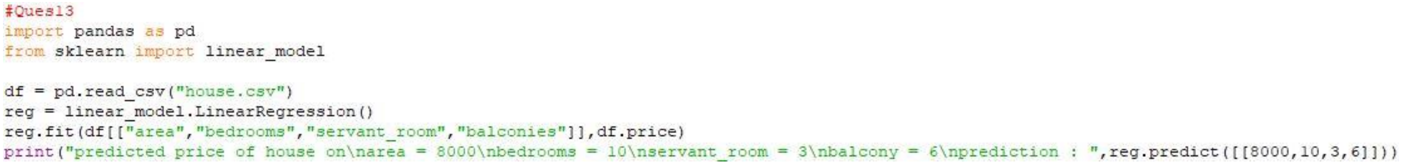
Output:



```
predicted price of house of area 2000 :  [452191.78082192]
```

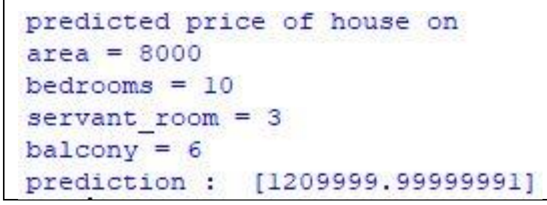
Q13) Based on multiple features/variables perform Linear Regression. For example, based on a number of additional features like number of bedrooms, servant room, number of balconies, number of houses of years a house has been built predict the price of a house.

Code:



```
#Ques13
import pandas as pd
from sklearn import linear_model

df = pd.read_csv("house.csv")
reg = linear_model.LinearRegression()
reg.fit(df[["area","bedrooms","servant_room","balconies"]],df.price)
print("predicted price of house on\narea = 8000\nbedrooms = 10\nservant_room = 3\nbalcony = 6\nprediction : ",reg.predict([[8000,10,3,6]]))
```

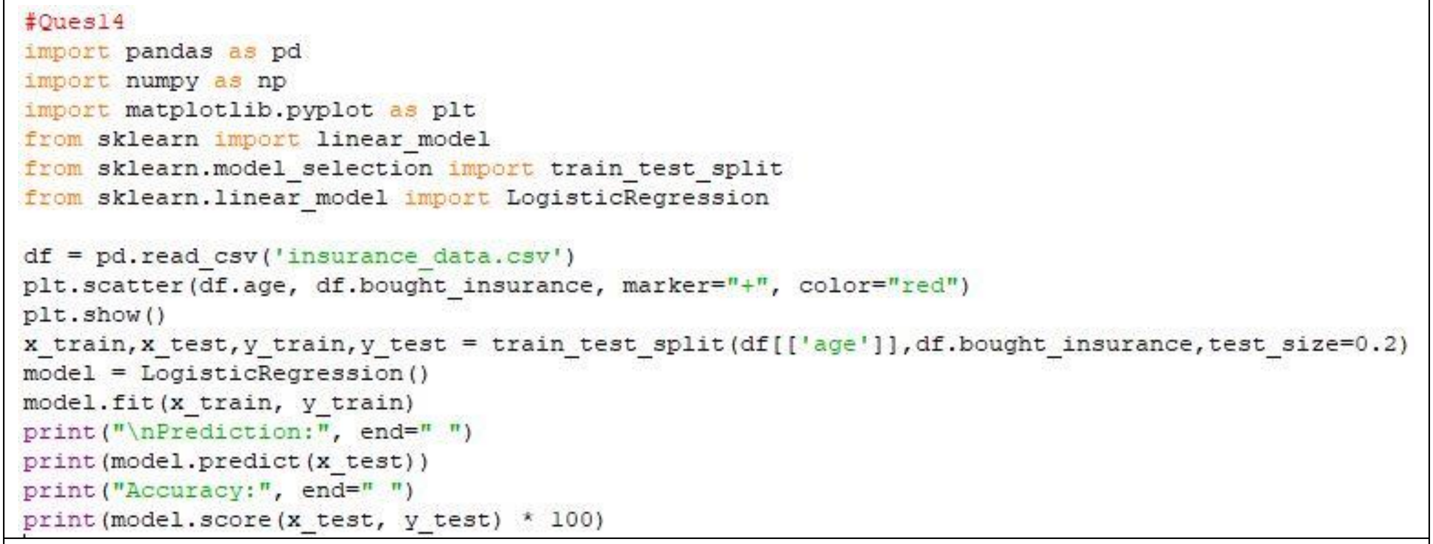
Output:



```
predicted price of house on
area = 8000
bedrooms = 10
servant_room = 3
balcony = 6
prediction :  [1209999.99999991]
```

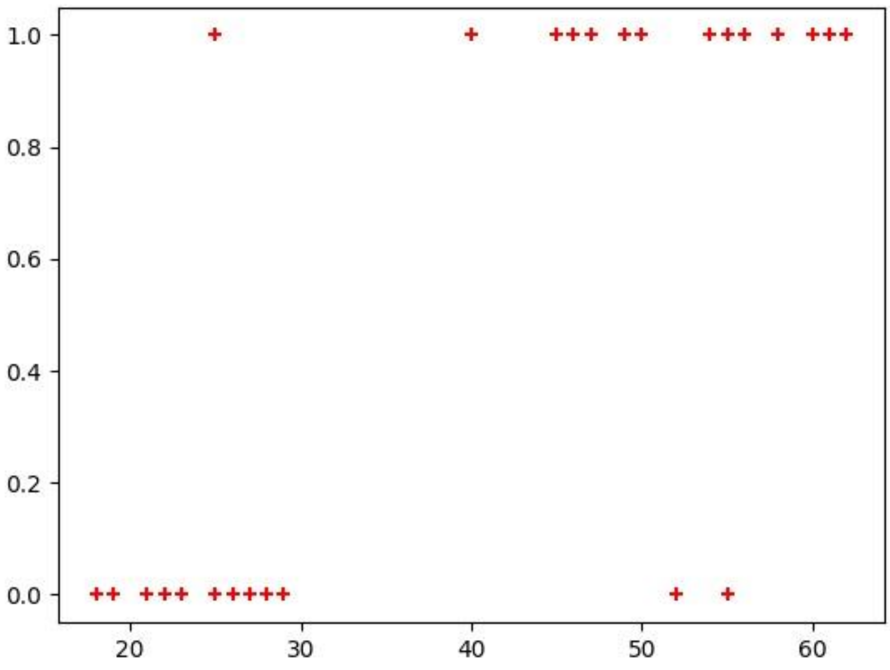
Q14) Implement a classification/ logistic regression problem. For example based on different features of student's data, classify, whether a student is suitable for a particular activity. Based on the available dataset, a student can also implement another classification problem like checking whether an email is spam or not.

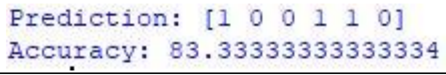
Code:



```
#Ques14
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

df = pd.read_csv('insurance_data.csv')
plt.scatter(df.age, df.bought_insurance, marker="+", color="red")
plt.show()
x_train,x_test,y_train,y_test = train_test_split(df[['age']],df.bought_insurance,test_size=0.2)
model = LogisticRegression()
model.fit(x_train, y_train)
print("\nPrediction:", end=" ")
print(model.predict(x_test))
print("Accuracy:", end=" ")
print(model.score(x_test, y_test) * 100)
```

Output:

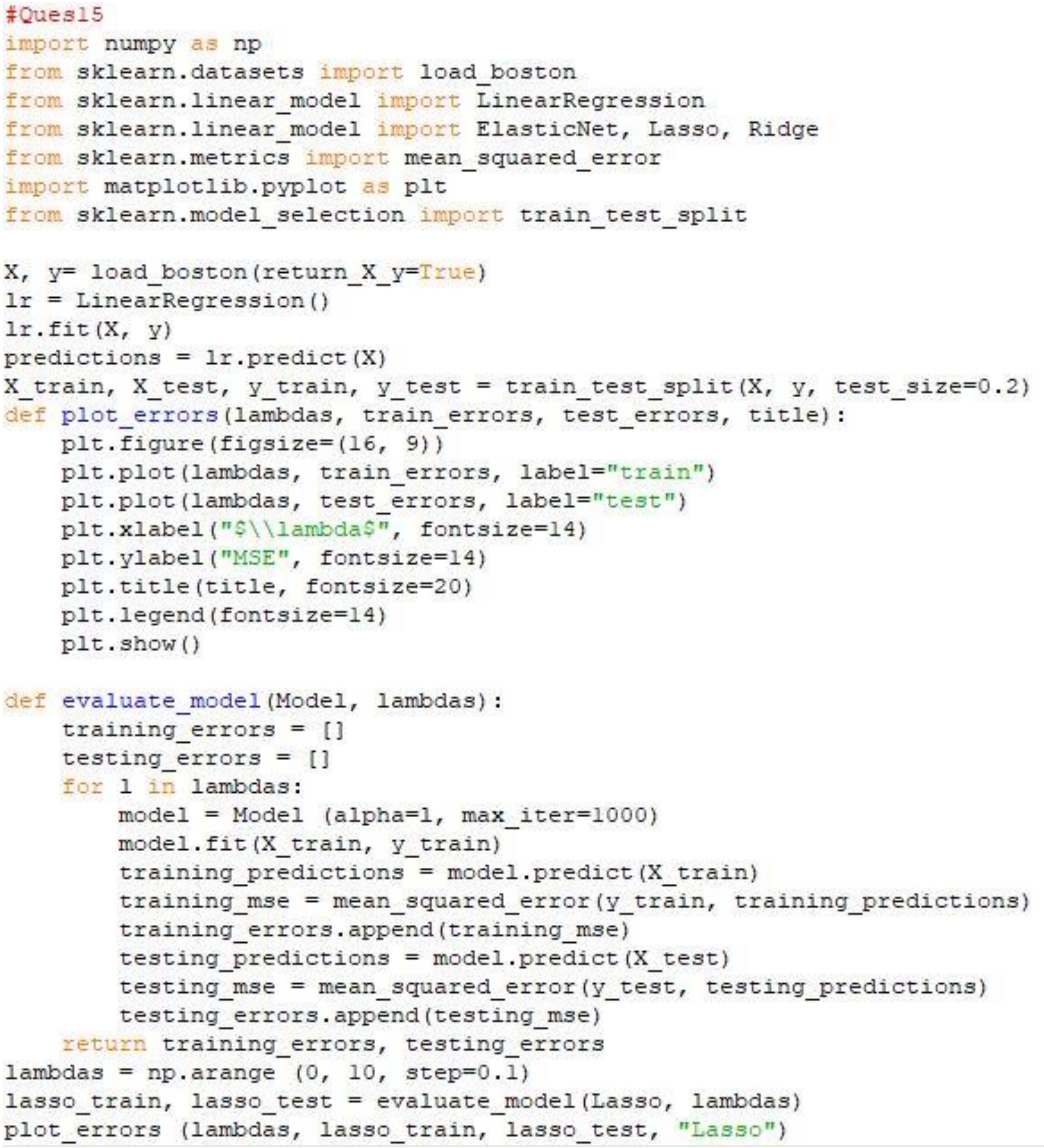




```
Prediction: [1 0 0 1 1 0]
Accuracy: 83.33333333333334
```

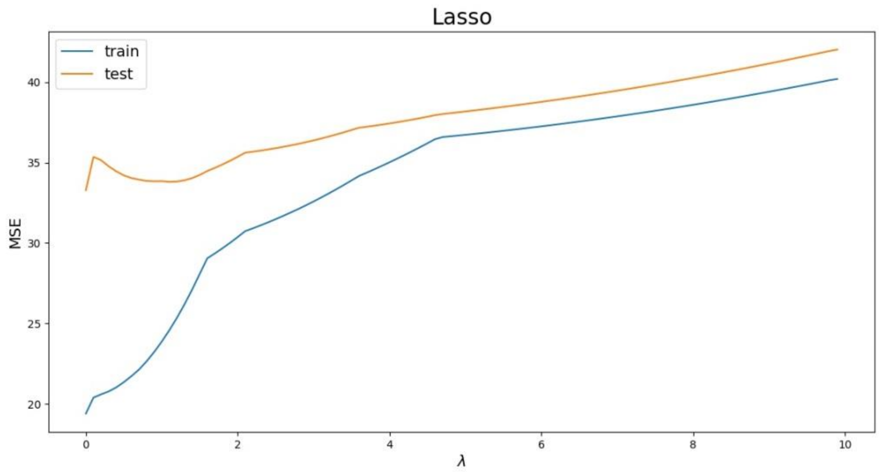
Q15) Use some function for regularization of dataset based on problem 14.

Code:



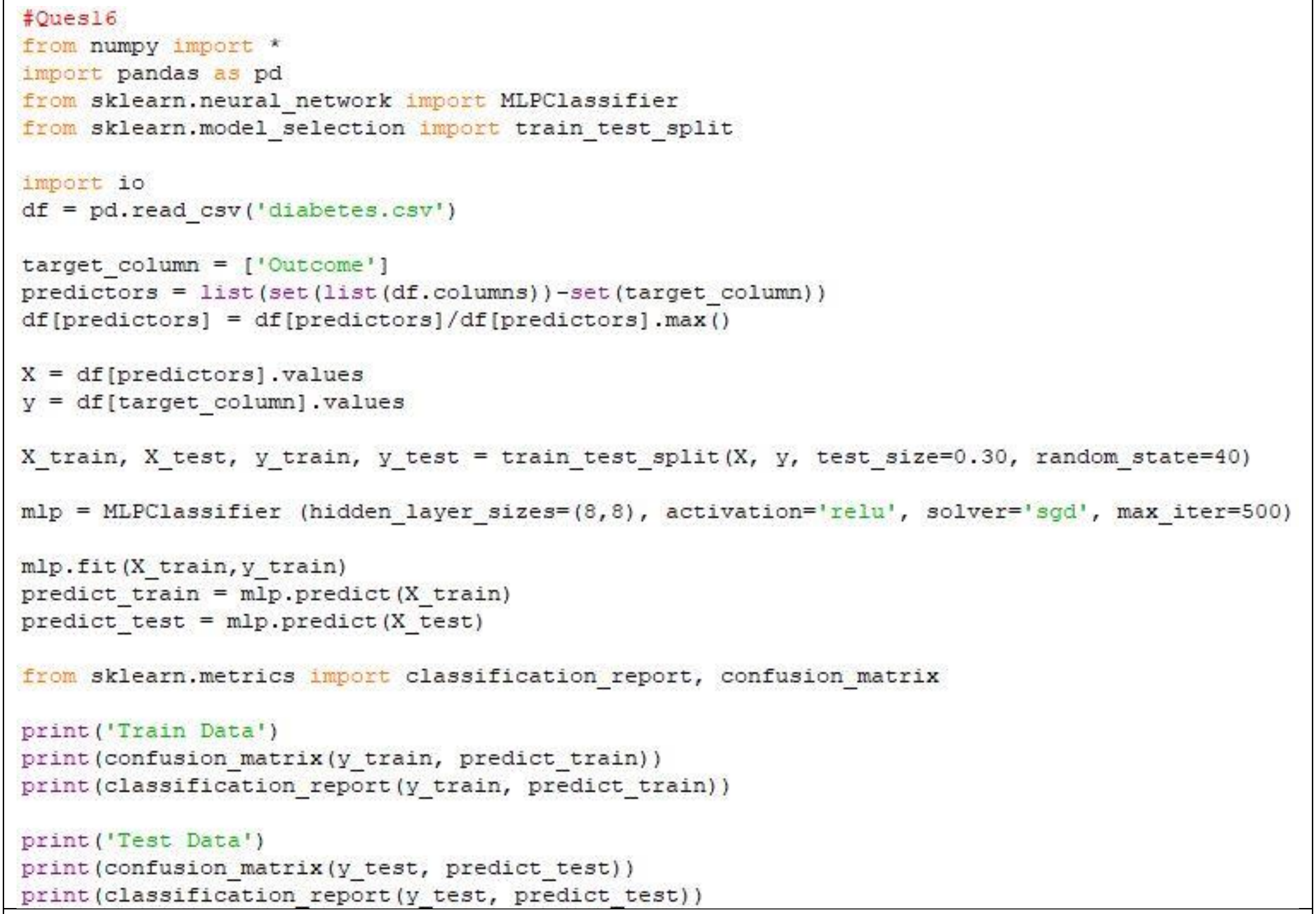
```
#Ques15
import numpy as np
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import ElasticNet, Lasso, Ridge
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

X, y= load_boston(return_X_y=True)
lr = LinearRegression()
lr.fit(X, y)
predictions = lr.predict(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
def plot_errors(lambdas, train_errors, test_errors, title):
    plt.figure(figsize=(16, 9))
    plt.plot(lambdas, train_errors, label="train")
    plt.plot(lambdas, test_errors, label="test")
    plt.xlabel("$\\lambda$", fontsize=14)
    plt.ylabel("MSE", fontsize=14)
    plt.title(title, fontsize=20)
    plt.legend(fontsize=14)
    plt.show()

def evaluate_model(Model, lambdas):
    training_errors = []
    testing_errors = []
    for l in lambdas:
        model = Model (alpha=l, max_iter=1000)
        model.fit(X_train, y_train)
        training_predictions = model.predict(X_train)
        training_mse = mean_squared_error(y_train, training_predictions)
        training_errors.append(training_mse)
        testing_predictions = model.predict(X_test)
        testing_mse = mean_squared_error(y_test, testing_predictions)
        testing_errors.append(testing_mse)
    return training_errors, testing_errors
lambdas = np.arange (0, 10, step=0.1)
lasso_train, lasso_test = evaluate_model(Lasso, lambdas)
plot_errors (lambdas, lasso_train, lasso_test, "Lasso")
```

Output:



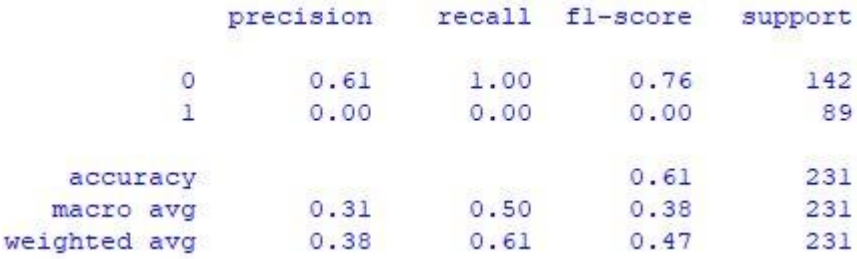
Q16) Use some function for neural networks, like Stochastic Gradient Descent or back propagation - algorithm to predict the value of a variable based on the dataset of problem 14

Code:



```
#Ques16
from numpy import *
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split

import io
df = pd.read_csv('diabetes.csv')

target_column = ['Outcome']
predictors = list(set(list(df.columns))-set(target_column))
df[predictors] = df[predictors]/df[predictors].max()

X = df[predictors].values
y = df[target_column].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=40)

mlp = MLPClassifier (hidden_layer_sizes=(8,8), activation='relu', solver='sgd', max_iter=500)

mlp.fit(X_train,y_train)
predict_train = mlp.predict(X_train)
predict_test = mlp.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix

print('Train Data')
print(confusion_matrix(y_train, predict_train))
print(classification_report(y_train, predict_train))

print('Test Data')
print(confusion_matrix(y_test, predict_test))
print(classification_report(y_test, predict_test))
```

Output:



```
              precision    recall  f1-score   support

           0       0.61      1.00      0.76       142
           1       0.00      0.00      0.00        89

    accuracy                           0.61       231
   macro avg       0.31      0.50      0.38       231
weighted avg       0.38      0.61      0.47       231
```