Name: SNEHA RANI

College Roll No.: 2020350

Examination Roll No.: 20066570052

Semester: VIth

Subject: MACHINE LEARNING

UNIQUE PAPER CODE: 32347607

CONTENT

1. Perform elementary mathematical operations in Octave/MATLAB/R like addition, multiplication, division and exponentiation.
2. Perform elementary logical operations in Octave/MATLAB/R (like OR, AND, Checking for Equality, NOT, XOR).
3. Create, initialize and display simple variables and simple strings and use simple formatting for variable.
4. Create/Define single dimension / multi-dimension arrays, and arrays with specific values like array of all ones, all zeros, array with random values within a range, or a diagonal matrix.
5. Use command to compute the size of a matrix, size/length of a particular row/column, load data from a text file, store matrix data to a text file, finding out variables and their features in the current scope.
6. Perform basic operations on matrices (like addition, subtraction, multiplication) and display specific rows or columns of the matrix.
7. Perform other matrix operations like converting matrix data to absolute values, taking the negative of matrix values, 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.
8. Create various type of plots/charts like histograms, plot based on sine/cosine function based on data from a matrix. Further label different axes in a plot and data in a plot.
9. Generate different subplots from a given plot and color plot data.
10. Use conditional statements and different type of loops based on simple example/s.

11. Perform vectorized implementation of simple matrix operation like finding the transpose of a matrix, adding, subtracting or multiplying two matrices.
12. Implement Linear Regression problem. For example, based on the "Advertising" dataset comprising of budget of TV, Radio etc. and the sales data, predict the estimated sales for TV budget.
13. Based on multiple features/variables perform Linear Regression on "Advertising" dataset. For example, based on the budget of TV, Radio and Newspaper, predict the overall sales.
14. Implement a classification/ logistic regression problem. For example, based on different features of "diabetes" data, classify, whether a woman is diabetic or not.
15. Use some function for regularization of "BOSTON" dataset available in 'sklearn library'.
16. Use some function for neural networks, like Stochastic Gradient Descent or backpropagation - algorithm to predict the value of a variable based on the dataset of problem 14.
17. Implement Simple Linear Regression on "Advertising" dataset using Analytical Method.
18. Implement Multiple Linear Regression on "Advertising" dataset using Normal Equation Method.

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

<u>CODE:</u>

```
n1=int(input("ENTER THE FIRST NUMBER: "))
n2=int(input("ENTER THE SECOND NUMBER: "))
print("SELECT ANY ONE OPTION:\n\t1. ADDITION\n\t2. SUBTRACTION\n\t3. MULTIPLICATION\n\t4. DIVISION\n\t5. EXPONENTIATION")
option='y'
while(option=='y' or option=='Y'):
    choice=int(input("ENTER YOUR CHOICE:"))
    if choice==1:
        s=n1+n2
        print("Addition:",s)
    elif choice==2:
        d=n1-n2
        print("Subtraction:",d)
    elif choice==3:
        m=n1*n2
        print("Multiplication:",m)
    elif choice==4:
        div=n1/n2
        print("Division:",div)
    elif choice==5:
        e=n1**n2
        print("Exponentiation:",e)
    else:
```

```
        print("WRONG CHOICE")

    print("DO YOU WANT TO CONTINUE? ('Y/N')")
    option=input()
print("END")
```

OUTPUT:

```
ENTER THE FIRST NUMBER: 7
ENTER THE SECOND NUMBER: 4
SELECT ANY ONE OPTION:
        1. ADDITION
        2. SUBTRACTION
        3. MULTIPLICATION
        4. DIVISION
        5. EXPONENTIATION
ENTER YOUR CHOICE:1
Addition: 11
DO YOU WANT TO CONTINUE? ('Y/N')
y
ENTER YOUR CHOICE:2
Subtraction: 3
DO YOU WANT TO CONTINUE? ('Y/N')
y
ENTER YOUR CHOICE:3
Multiplication: 28
DO YOU WANT TO CONTINUE? ('Y/N')
y
ENTER YOUR CHOICE:4
Division: 1.75
DO YOU WANT TO CONTINUE? ('Y/N')
y
ENTER YOUR CHOICE:5
Exponentiation: 2401
DO YOU WANT TO CONTINUE? ('Y/N')
y
ENTER YOUR CHOICE:6
WRONG CHOICE
DO YOU WANT TO CONTINUE? ('Y/N')
n
END
```

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

CODE:

```
X=True
Y=False
print("X AND Y is: ", X and Y)
```

```
print("X OR Y is: ", X or Y)
print("NOT X is: ", not X)
print("NOT Y is: ", not Y)
print("X XOR y is: ", X ^ Y)
print("X == Y is: ", X==Y)
```

OUTPUT:

```
X AND Y is:  False
X OR Y is:  True
NOT X is:  False
NOT Y is:  True
X XOR y is:  True
X == Y is:  False
```

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

CODE:

```
x=45
y=25
s="Hello"
z=x+y
print(z)
print(s)
print(s+" World")
```

OUTPUT:

```
70
Hello
Hello World
```

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
m=np.array([1,2,3])
print("Matrix 1: ", m)

n=np.array([[4,5,6],[1,3,7]])
print("Matrix 2:\n", n,'\n')

b=np.ones((2,3)).astype('int32')
print("Ones Matrix:\n", b,'\n')

a=np.zeros((2,2)).astype('int32')
print("Zeroes Matrix:\n", a,'\n')

c=np.random.randint(1,7,size=(3,3))
print("Random Value Matrix:\n", c,'\n')

d=np.diag([1,2,3,4])
print("Diagonal Matrix:\n",d)
```

OUTPUT:

```
Matrix 1:  [1 2 3]
Matrix 2:
 [[4 5 6]
 [1 3 7]]

Ones Matrix:
 [[1 1 1]
 [1 1 1]]

Zeroes Matrix:
 [[0 0]
 [0 0]]

Random Value Matrix:
 [[5 5 2]
 [1 4 6]
 [1 1 5]]

Diagonal Matrix:
 [[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
```

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


x=np.array([[1,2,3,4],[5,6,7,8]])
print("SHAPE: ", x.shape,'\n')
print("SIZE: ", x.size,'\n')
print("ITEM SIZE: ", x.itemsize,'\n')


file=np.random.randint(1,20,size=(3,3))
np.savetxt('data1.txt',file)
np.genfromtxt('data1.txt',delimiter=' ')
```

```
file=file.astype('int32')
print("MATRIX:\n", file)
```

OUTPUT:

```
SHAPE:  (2, 4)

SIZE:  8

ITEM SIZE:  4

MATRIX:
 [[ 4 19 17]
 [13 18 15]
 [14 18  3]]
```

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

CODE:

```
import numpy as np

a=np.array([[1,2,3],[4,5,6]])
b=np.array([[4,5,6],[7,8,9]])

s=a+b
print("Matrix after ADDITION:\n", s,'\n')
print("3rd Column of Sum Matrix:\n", s[:,2],'\n')

d=b-a
print("Matrix after SUBTRACTION:\n", d,'\n')
print("2nd Row of Difference Matrix:\n", d[1,:],'\n')
```

```
m=a*b
print("Matrix after MULTIPLICATION:\n", m,'\n')
print("M[1,2]: ", m[1,2],'\n')
```

OUTPUT:

```
Matrix after ADDITION:
 [[ 5  7  9]
 [11 13 15]]

3rd Column of Sum Matrix:
 [ 9 15]

Matrix after SUBTRACTION:
 [[3 3 3]
 [3 3 3]]

2nd Row of Difference Matrix:
 [3 3 3]

Matrix after MULTIPLICATION:
 [[ 4 10 18]
 [28 40 54]]

M[1,2]:  54
```

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


c=np.array([[1,2,4],[5,6,8]])
print("MATRIX:\n", c, '\n')


v=np.sin(c)
print("SINE MATRIX:\n", v,'\n')
```

```
print("ABSOLUTE MATRIX:\n", np.abs(v),'\n')

d=np.append(c,np.array([[9,10,12]]), axis=0)
print("APPENDED MATRIX:\n", d,'\n')

print("2nd ROW DELETED:\n", np.delete(d,1,0),'\n')
print("2nd COLUMN DELETED:\n", np.delete(d,1,1),'\n')

print("MAX. ELEMENT OF THE MATRIX:", np.max(d),'\n')
print("MAX. ELEMENT OF THE MATRIX:", np.min(d),'\n')
print("SUM OF ALL THE ELEMENTS OF THE MATRIX:",np.sum(d),'\n')
print("COLUMN-WISE SUM OF ELEMENTS\n", np.sum(d,axis=0))
```

OUTPUT:

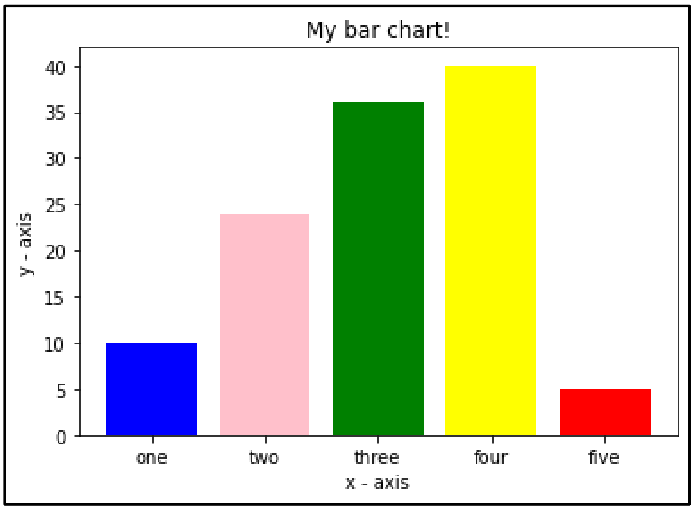
```
MATRIX:
 [[1 2 4]
 [5 6 8]]

SINE MATRIX:
 [[ 0.84147098  0.90929743 -0.7568025 ]
 [-0.95892427 -0.2794155   0.98935825]]

ABSOLUTE MATRIX:
 [[0.84147098 0.90929743 0.7568025 ]
 [0.95892427 0.2794155  0.98935825]]

APPENDED MATRIX:
 [[ 1  2  4]
 [ 5  6  8]
 [ 9 10 12]]

2nd ROW DELETED:
 [[ 1  2  4]
 [ 9 10 12]]

2nd COLUMN DELETED:
 [[ 1  4]
 [ 5  8]
 [ 9 12]]

MAX. ELEMENT OF THE MATRIX: 12

MAX. ELEMENT OF THE MATRIX: 1

SUM OF ALL THE ELEMENTS OF THE MATRIX: 57

COLUMN-WISE SUM OF ELEMENTS
 [15 18 24]
```

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.**

CODE:

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

# x-coordinates of left sides of bars
left = [1, 2, 3, 4, 5]
# heights of bars
height = [10, 24, 36, 40, 5]
# labels for bars
tick_label = ['one', 'two', 'three', 'four', 'five']

# plotting a bar chart
plt.bar(left, height, tick_label = tick_label, width = 0.8, color =
['blue', 'pink','green','yellow','red'])

# naming the x-axis
plt.xlabel('x - axis')
# naming the y-axis
plt.ylabel('y - axis')
# plot title
plt.title('My bar chart!')
# function to show the plot
plt.show()
```

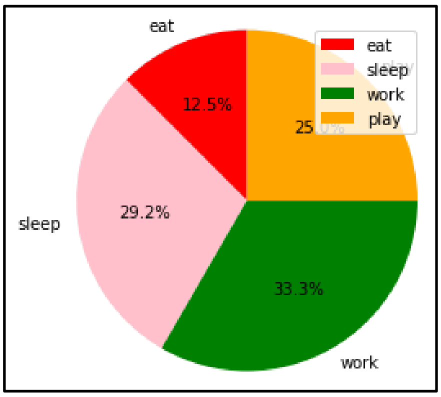
OUTPUT:



CODE:

```
# defining labels
activities = ['eat', 'sleep', 'work', 'play']
# portion covered by each label
slices = [3, 7, 8, 6]
# color for each label
colors = ['red', 'pink', 'green', 'orange']

# plotting the pie chart
plt.pie(slices, labels = activities, colors=colors, startangle=90, radius
= 1.2, autopct = '%1.1f%%')

# plotting legend
plt.legend()
# showing the plot
plt.show()
```

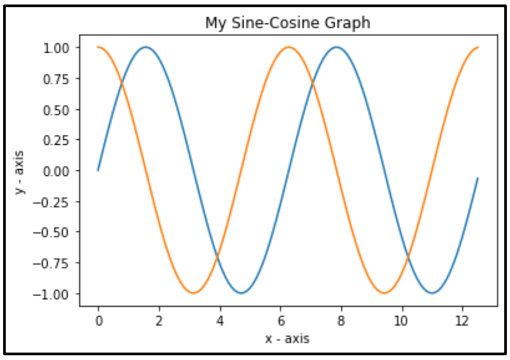
OUTPUT:



CODE:

```
x = np.arange(0,4*np.pi,0.1)   # start,stop,step
y = np.sin(x)
z = np.cos(x)
plt.plot(x,y,x,z)
# naming the x-axis
plt.xlabel('x - axis')
# naming the y-axis
plt.ylabel('y - axis')
# plot title
plt.title('My Sine-Cosine Graph')
plt.show()
```

OUTPUT:

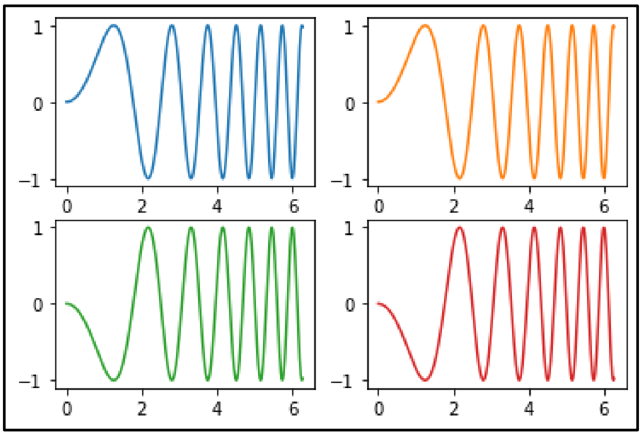


9. Generate different subplots from a given plot and color plot data.

CODE:

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

x = np.linspace(0, 2 * np.pi, 400)
y = np.sin(x ** 2)

fig, axs = plt.subplots(2, 2)
axs[0, 0].plot(x, y)
axs[0, 1].plot(x, y, 'tab:orange')
axs[1, 0].plot(x, -y, 'tab:green')
axs[1, 1].plot(x, -y, 'tab:red')
```

OUTPUT:



10. **Use conditional statements and different type of loops based on simple example/s.**

CODE:

```
s=input("Enter a String: ")
s1=""
for i in s:
   s1=i+s1
if(s1==s):
   print(s, "is Palindrome")
else:
   print(s, "is not Palindrome")
```

OUTPUT:

```
Enter a String: mama
mama is not Palindrome
```

```
Enter a String: maam
maam is Palindrome
```

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

CODE:

```
import numpy as np
a=np.array([[1,2,3],[4,5,6],[1,1,1]])
b=np.array([[6,7,8],[9,10,11],[2,2,2]])
print("MATRIX 1:\n", a,'\n')
```

```
print("MATRIX 2:\n", b,'\n')
print("TRANSPOSE OF MATRIX 1:\n", np.transpose(a),'\n')
print("M1+M2:\n", a+b,'\n')
print("M1-M2:\n", b-a,'\n')
print("M1*M2:\n", np.matmul(a,b),'\n')
```

OUTPUT:

```
MATRIX 1:
 [[1 2 3]
 [4 5 6]
 [1 1 1]]

MATRIX 2:
 [[ 6  7  8]
 [ 9 10 11]
 [ 2  2  2]]

TRANSPOSE OF MATRIX 1:
 [[1 4 1]
 [2 5 1]
 [3 6 1]]

M1+M2:
 [[ 7  9 11]
 [13 15 17]
 [ 3  3  3]]

M1-M2:
 [[5 5 5]
 [5 5 5]
 [1 1 1]]

M1*M2:
 [[30 33 36]
 [81 90 99]
 [17 19 21]]
```

12. Implement Linear Regression problem. For example, based on the "Advertising" dataset comprising of budget of TV, Radio etc. and the sales data, predict the estimated sales for TV budget.

CODE:

```
import pandas as pd
import numpy as np
```

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics

adv = pd.read_csv("Advertising.csv")
adv.head()
```

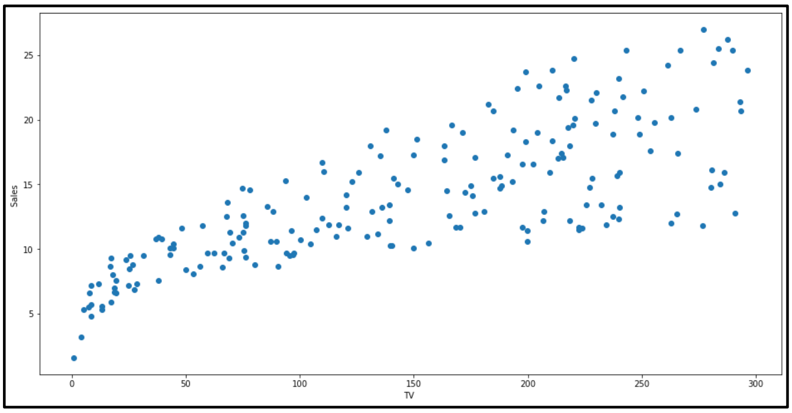
OUTPUT:

	Unnamed: 0	TV	radio	newspaper	sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

CODE:

```
#check for nulls in the data
adv.isnull().sum()
plt.figure(figsize=(16, 8))
plt.scatter(adv['TV'], adv['sales'])
plt.xlabel("TV ")
plt.ylabel("Sales ")
plt.show()
```

OUTPUT:



CODE:

```
x = adv['TV'].values.reshape(-1,1)
y = adv['sales'].values.reshape(-1,1)

# split data into train and test
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.3,random_state=0)

#fit the model using Linear Regression
linreg = LinearRegression()
linreg.fit(x_train, y_train)


print("INTERCEPT: ", linreg.intercept_[0])        #Intercept
print("\nCOEFFICIENT: ", linreg.coef_[0][0])        #Coefficient
print("\nThe linear model is: y = {:.5} +
{:.5}TV".format(linreg.intercept_[0], linreg.coef_[0][0]))
```

```
# Make predictions using the testing set
y_pred = linreg.predict(x_test)
y=linreg.predict(np.array([1000]).reshape(1,-1))        #Prediction
print("\nPredicted Value for the SALES of TV: ", y)



#Accuracy Score
print("\nAccuracy Score: ", linreg.score(x_test,y_test))
print('Mean Squared Error :',
metrics.mean_squared_error(y_test,y_pred))
print('Root Mean Squared Error :',
np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

OUTPUT:

```
INTERCEPT:  7.310810165411684

COEFFICIENT:  0.04581434217189621

The linear model is: y = 7.3108 + 0.045814TV

Predicted Value for the SALES of TV:  [[53.12515234]]

Accuracy Score:  0.725606346597073
Mean Squared Error : 7.497479593464676
Root Mean Squared Error : 2.738152587688399
```

CODE:

```
print('Train Score :', linreg.score(x_train,y_train))
print('Test Score:', linreg.score(x_test,y_test))
```

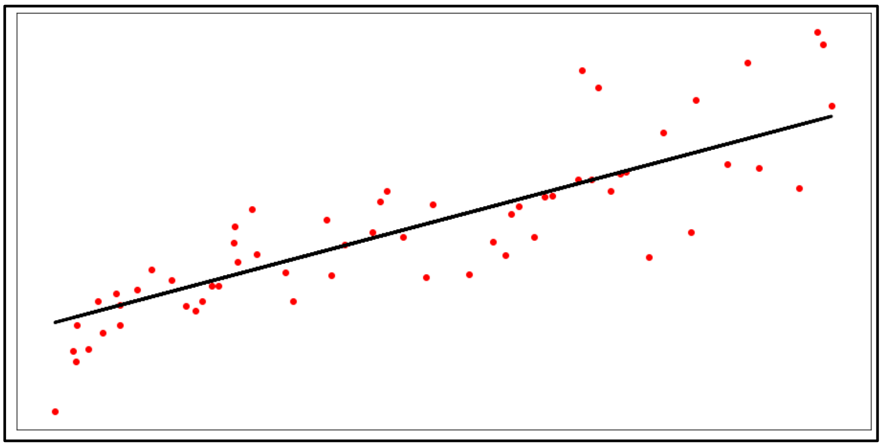
OUTPUT:

```
Train Score : 0.5552336104251211
Test Score: 0.725606346597073
```

CODE:

```
plt.figure(figsize=(16, 8))
plt.scatter(x_test, y_test, color="red")
plt.plot(x_test, y_pred, color="black", linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

OUTPUT:

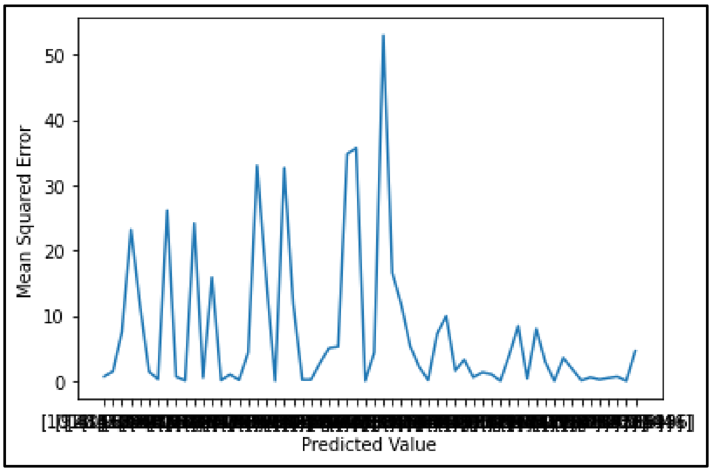


CODE:

```
errors = list()
for i in range(len(y_test)):
    # calculate error
    err = (y_test[i] - y_pred[i])**2
    # store error
    errors.append(err)

# plot errors
plt.plot(errors)
plt.xticks(ticks=[i for i in range(len(errors))], labels=y_pred)
plt.xlabel('Predicted Value')
```

```
plt.ylabel('Mean Squared Error')
plt.show()
```

OUTPUT:



13. Based on multiple features/variables perform Linear Regression on “Advertising” dataset. For example, based on the budget of TV, Radio and Newspaper, predict the overall sales.

CODE:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics

adv = pd.read_csv("Advertising.csv")
adv.head()
```

OUTPUT:

	Unnamed: 0	TV	radio	newspaper	sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

CODE:

```
x = adv.drop(['sales', 'Unnamed: 0'], axis=1)
y = adv['sales'].values.reshape(-1,1)

# split data into train and test
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.3,random_state=0)

#fit the model using Linear Regression
linreg = LinearRegression()
linreg.fit(x_train, y_train)

print("INTERCEPT: ", linreg.intercept_[0])        #Intercept
print("\nCOEFFICIENT: ", linreg.coef_)        #Coefficient
print("The linear model is: Y = {:.5} + {:.5}*TV + {:.5}*radio +
{:.5}*newspaper".format(linreg.intercept_[0], linreg.coef_[0][0],
linreg.coef_[0][1], linreg.coef_[0][2]))

# Make predictions using the testing set
y_pred = linreg.predict(x_test)
```

```
y=linreg.predict(np.array([275,55.7,80.6]).reshape(1,-1)) #Prediction

print("\nPredicted Value for the SALES for given instance: ", y)


#Accuracy Score

print("\nAccuracy Score: ", linreg.score(x_test,y_test))

predictions = linreg.predict(x_test)

print('Mean Squared Error :', metrics.mean_squared_error(y_test,predictions))

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,predictions)))
```

OUTPUT:

```
INTERCEPT:  2.880255286331325

COEFFICIENT:  [[0.04391531 0.20027962 0.00184368]]
The linear model is: Y = 2.8803 + 0.043915*TV + 0.20028*radio + 0.0018437*newspaper

Predicted Value for the SALES for given instance:  [[26.26114198]]

Accuracy Score:  0.8649018906637792
Mean Squared Error : 3.6913948456986083
Root Mean Squared Error: 1.9213003007595164
```

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

CODE:

```
import pandas as pd

# split X and y into training and testing sets

from sklearn.model_selection import train_test_split

# import the class

from sklearn.linear_model import LogisticRegression

from sklearn import metrics
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
# load dataset
pima = pd.read_csv("diabetes.csv", header= None, names=col_names)
pima.head()
```

OUTPUT:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

CODE:

```
#split dataset in features and target variable
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
x = pima[feature_cols] # Features
y = pima.label # Target variable

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=0)
```

```
print(x_train)

logreg = LogisticRegression()
logreg.fit(x_train,y_train)
y_pred=logreg.predict(x_test)

print("\ny_pred: ", y_pred)

#Accuracy Score
print("\nAccuracy Score: ", logreg.score(x_test,y_test))
mse = metrics.mean_squared_error(y_test, logreg.predict(x_test))
#Mean Square Error
print("\nMean Squared Error: ",mse)
#Root Mean Square Error
print("\nRoot Mean Squared Error: ",np.sqrt(mse))
```

OUTPUT:

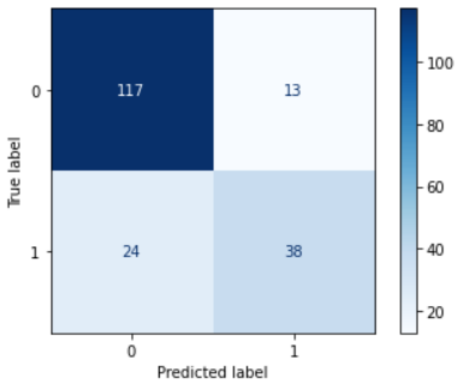
```
     pregnant  insulin   bmi  age  glucose  bp  pedigree
762         9        0  22.5   33       89  62     0.142
127         1       94  33.3   23      118  58     0.261
564         0        0  32.4   27       91  80     0.601
375        12      325  39.2   58      140  82     0.528
663         9      130  37.9   40      145  80     0.637
..        ...      ...   ...  ...      ...  ..       ...
763        10      180  32.9   63      101  76     0.171
192         7        0  30.4   36      159  66     0.383
629         4        0  24.7   21       94  65     0.148
559        11        0  30.1   35       85  74     0.300
684         5        0   0.0   69      136  82     0.640

[576 rows x 7 columns]

y_pred:  [1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0
 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 1
 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0
 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0
 0 0 0 1 0 0 1 0 1 0 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1
 0 1 0 0 0 0 0]

Accuracy Score:  0.8072916666666666

Mean Squared Error:  0.19270833333333334

Root Mean Squared Error:  0.4389855730355308
```

CODE:

```
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
print("CONFUSION MATRIX:\n", cnf_matrix)


metrics.plot_confusion_matrix(logreg, x_test, y_test,
cmap=plt.cm.Blues)
plt.show()
```

OUTPUT:

```
CONFUSION MATRIX:
 [[117  13]
 [ 24  38]]
```



CODE:

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
print("f1-Score:",metrics.f1_score(y_test,y_pred))
```

OUTPUT:

```
Accuracy: 0.8072916666666666
Precision: 0.7450980392156863
Recall: 0.6129032258064516
f1-Score: 0.672566371681416
```

15. Use some function for regularization of "BOSTON" dataset available in 'sklearn library'.

CODE:

```
from sklearn.datasets import load_boston
import pandas as pd
# split X and y into training and testing sets
from sklearn.model_selection import train_test_split
# import the class
from sklearn.linear_model import RidgeCV, LassoCV
from sklearn import metrics
import numpy as np

boston_dataset = load_boston()
boston = pd.DataFrame(boston_dataset.data,
columns=boston_dataset.feature_names)

boston['MEDV'] = boston_dataset.target
boston.head()

x = boston.drop(['MEDV'], axis=1)
y = boston['MEDV']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,rand
om_state=0)

alpha_range=[0.00001, 0.01, 0.05, 0.1, 0.5, 1, 1.5, 3, 5, 6, 7, 8, 9, 10]
ridgecv=RidgeCV(alphas=alpha_range, normalize=True,
scoring='neg_mean_squared_error')
ridgecv.fit(x_train,y_train)
```

```
print("\nAlpha Range: ", alpha_range)
print("\nAlpha Value: ", ridgecv.alpha_)
print("\nCoefficient: ", ridgecv.coef_)
```

OUTPUT:

```
Alpha Range:  [1e-05, 0.01, 0.05, 0.1, 0.5, 1, 1.5, 3, 5, 6, 7, 8, 9, 10]

Alpha Value:  0.01

Coefficient:  [-1.13861015e-01  4.12256877e-02 -1.93278669e-02  2.44871215e+00
 -1.44968099e+01  3.82199823e+00 -7.81325806e-03 -1.37020341e+00
  2.08689005e-01 -9.82481934e-03 -9.72643953e-01  8.43720305e-03
 -4.90564712e-01]
```

CODE:

```
y_pred=ridgecv.predict(x_test)


print("\nAccuracy Score: ", ridgecv.score(x_test,y_test))
print("\nMean Absolute Error: ",
metrics.mean_absolute_error(y_test,y_pred))
print("\nMean Squared Score: ",
metrics.mean_squared_error(y_test,y_pred))
print("\nRoot Mean Squared Error ",
np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

OUTPUT:

```
Accuracy Score:  0.6322070128554689

Mean Absolute Error:  3.6679777452193574

Mean Squared Score:  30.048324934720537

Root Mean Squared Error  5.481635242764748
```

CODE:

```
#Lasso
lambda_values = [0.000001, 0.0001, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5]
lassocv=LassoCV(alphas=alpha_range, normalize=True)
lassocv.fit(x_train,y_train)


print("\nAlpha Range: ", lambda_values)
print("\nAlpha Value: ", lassocv.alpha_)
print("\nCoefficient: ", lassocv.coef_)
```

OUTPUT:

```
Alpha Range:  [1e-06, 0.0001, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5]

Alpha Value:  1e-05

Coefficient:  [-1.17660911e-01  4.39875866e-02 -5.81576667e-03  2.39328094e+00
 -1.55815760e+01  3.76933909e+00 -7.02220691e-03 -1.43422387e+00
  2.39751536e-01 -1.12833576e-02 -9.85455195e-01  8.44173540e-03
 -4.99134840e-01]
```

CODE:

```
y_pred=lassocv.predict(x_test)


print("\nAccuracy Score ", lassocv.score(x_test,y_test))
print("\nMean Absolute Error: ", metrics.mean_absolute_error(y_test,y_pred))
print("\nMean Squared Score: ", metrics.mean_squared_error(y_test,y_pred))
print("\nRoot Mean Squared Error ", np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

OUTPUT:

```
Accuracy Score  0.6354423976617791

Mean Absolute Error:  3.6683036000753866

Mean Squared Score:  29.783997181481826

Root Mean Squared Error  5.457471683983512
```

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:

```
import pandas as pd
# split X and y into training and testing sets
from sklearn.model_selection import train_test_split
# import the class
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import plot_confusion_matrix
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi',
'pedigree', 'age', 'label']
# load dataset
pima = pd.read_csv("diabetes.csv", header= None,
names=col_names)
```

```
#split dataset in features and target variable
feature_cols = ['pregnant', 'insulin', 'bmi',
'age','glucose','bp','pedigree']
x = pima[feature_cols] # Features
y = pima.label # Target variable

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,rand
om_state=0)

# Create model object
mlp = MLPClassifier(hidden_layer_sizes=(10,10,10),
                    random_state=5,
                    verbose=True,
                    solver='sgd',
                    learning_rate_init=0.001)

# Fit data onto the model
mlp.fit(x_train,y_train)
# Make prediction on test dataset
y_pred=mlp.predict(x_test)
```

OUTPUT:

```
Iteration 1, loss = 6.92712434
Iteration 2, loss = 2.55294936
Iteration 3, loss = 1.48103042
Iteration 4, loss = 0.82137147
Iteration 5, loss = 0.81144794
Iteration 6, loss = 0.80703054
Iteration 7, loss = 0.75884397
Iteration 8, loss = 0.72298408
Iteration 9, loss = 0.68969498
Iteration 10, loss = 0.68911240
Iteration 11, loss = 0.66665728
Iteration 12, loss = 0.65552754
Iteration 13, loss = 0.64524602
Iteration 14, loss = 0.64094625
Iteration 15, loss = 0.63795461
Iteration 16, loss = 0.63472240
Iteration 17, loss = 0.62134055
Iteration 18, loss = 0.61283931
Iteration 19, loss = 0.61447460
```

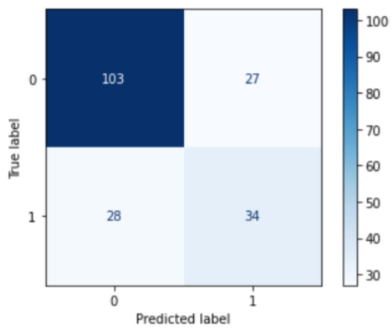
```
Iteration 124, loss = 0.57051958
Iteration 125, loss = 0.56880629
Iteration 126, loss = 0.57143467
Iteration 127, loss = 0.56771758
Iteration 128, loss = 0.56671296
Iteration 129, loss = 0.56656943
Iteration 130, loss = 0.56510840
Iteration 131, loss = 0.56725110
Iteration 132, loss = 0.56848550
Iteration 133, loss = 0.56690378
Iteration 134, loss = 0.56779400
Iteration 135, loss = 0.56909970
Iteration 136, loss = 0.56969714
Iteration 137, loss = 0.56782940
Iteration 138, loss = 0.56550063
Iteration 139, loss = 0.57265766
Iteration 140, loss = 0.57030657
Iteration 141, loss = 0.56999220
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
```

CODE:

```
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
print("CONFUSION MATRIX:\n", cnf_matrix)


plot_confusion_matrix(mlp, x_test, y_test,  cmap=plt.cm.Blues)
plt.show()
```

OUTPUT:

```
CONFUSION MATRIX:
 [[103  27]
 [ 28  34]]
```



CODE:

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
print("f1-Score:",metrics.f1_score(y_test,y_pred))
```

OUTPUT:

```
Accuracy: 0.7135416666666666
Precision: 0.5573770491803278
Recall: 0.5483870967741935
f1-Score: 0.5528455284552846
```

17. Implement Simple Linear Regression on "Advertising" dataset using Analytical Method.

CODE:

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

def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
```

```
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return (b_0, b_1)

# observations / data
adv = pd.read_csv("Advertising.csv")
print(adv.head())
x = adv['TV']
y = adv['sales']
# estimating coefficients
b = estimate_coef(x, y)
print("\nEstimated coefficients:\nb_0(Intercept) = {} \
    \nb_1(Coefficient) = {}".format(b[0], b[1]))
```

OUTPUT:

```
   Unnamed: 0     TV  radio  newspaper  sales
0           1  230.1   37.8       69.2   22.1
1           2   44.5   39.3       45.1   10.4
2           3   17.2   45.9       69.3    9.3
3           4  151.5   41.3       58.5   18.5
4           5  180.8   10.8       58.4   12.9

Estimated coefficients:
b_0(Intercept) = 7.032593549127706
b_1(Coefficient) = 0.04753664043301968
```

CODE:

```
y_pred = b[0] + b[1]*x

y_bar=np.mean(y)
ssr=np.sum((y-y_pred)*(y-y_pred))
sst=np.sum((y-y_bar)*(y-y_bar))
```

```
acc=1-(ssr/sst)
print("Accuracy Score: ", acc)

var=np.mean(y-y_pred)
MSE=var*var
print("Mean Squared Error: ", MSE)
print("Root Mean Squared Error: ", np.sqrt(MSE))
```

OUTPUT:

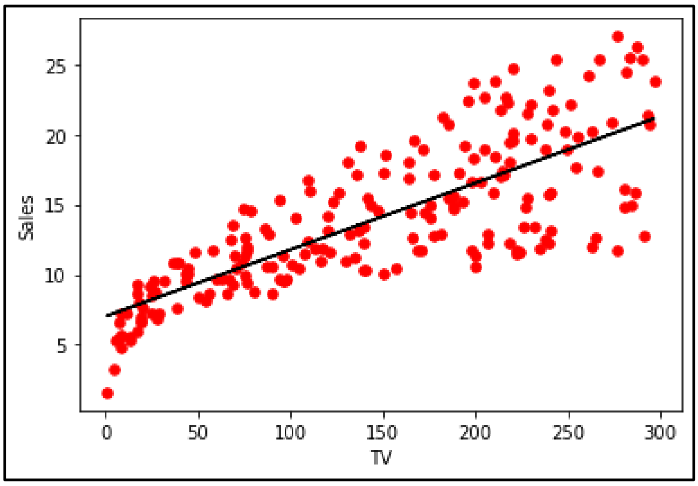
```
Accuracy Score:  0.611875050850071
Mean Squared Error:  7.753102470215777e-30
Root Mean Squared Error:  2.7844393457598924e-15
```

CODE:

```
#Plot Regression Line

# plotting the actual points as scatter plot
plt.scatter(x, y, color = "r", s = 30)

# predicted response vector
y_pred = b[0] + b[1]*x

# plotting the regression line
plt.plot(x, y_pred, color = "black")

# putting labels
plt.xlabel('TV')
plt.ylabel('Sales')
```

```
# function to show plot
plt.show()
```

OUTPUT:



18. Implement Multiple Linear Regression on "Advertising" dataset using Normal Equation Method.

CODE:

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


adv = pd.read_csv("Advertising.csv")
print(adv.head())


x1=adv['TV']
x2=adv['radio']
x3=adv['newspaper']
y=adv['sales']
```

OUTPUT:

	Unnamed: 0	TV	radio	newspaper	sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

CODE:

```
x1=np.array(x1)
x2=np.array(x2)
x3=np.array(x3)
y=np.array(y)

n=len(x1)

x_bias=np.ones((n,1))
x1_n=np.reshape(x1,(n,1))
x2_n=np.reshape(x2,(n,1))
x3_n=np.reshape(x3,(n,1))

x=np.append(x_bias,x1_n,axis=1)
x=np.append(x,x2_n,axis=1)
x=np.append(x,x3_n,axis=1)
print('X= \n', x)
```

OUTPUT:

```
X=
[[  1.   230.1  37.8  69.2]
 [  1.    44.5  39.3  45.1]
 [  1.    17.2  45.9  69.3]
 [  1.   151.5  41.3  58.5]
 [  1.   180.8  10.8  58.4]
 [  1.     8.7  48.9  75. ]
 [  1.    57.5  32.8  23.5]
 [  1.   120.2  19.6  11.6]
 [  1.     8.6   2.1   1. ]
 [  1.   199.8   2.6  21.2]
 [  1.    66.1   5.8  24.2]
 [  1.   214.7  24.    4. ]
 [  1.    23.8  35.1  65.9]
 [  1.    97.5   7.6   7.2]
 [  1.   204.1  32.9  46. ]
 [  1.   195.4  47.7  52.9]
 [  1.    67.8  36.6 114. ]
 [  1.   281.4  39.6  55.8]
```

```
 [  1.   218.5   5.4  27.4]
 [  1.    56.2   5.7  29.7]
 [  1.   287.6  43.   71.8]
 [  1.   253.8  21.3  30. ]
 [  1.   205.   45.1  19.6]
 [  1.   139.5   2.1  26.6]
 [  1.   191.1  28.7  18.2]
 [  1.   286.   13.9   3.7]
 [  1.    18.7  12.1  23.4]
 [  1.    39.5  41.1   5.8]
 [  1.    75.5  10.8   6. ]
 [  1.    17.2   4.1  31.6]
 [  1.   166.8  42.    3.6]
 [  1.   149.7  35.6   6. ]
 [  1.    38.2   3.7  13.8]
 [  1.    94.2   4.9   8.1]
 [  1.   177.    9.3   6.4]
 [  1.   283.6  42.   66.2]
 [  1.   232.1   8.6   8.7]]
```

CODE:

```
x_trans=np.transpose(x)
x_trans_dot_x=x_trans.dot(x)
temp1=np.linalg.inv(x_trans_dot_x)
temp2=x_trans.dot(y)

theta=temp1.dot(temp2)

b0=theta[0]
b1=theta[1]
b2=theta[2]
b3=theta[3]

print("\nbeta0: ",b0)
print("\nbeta1: ",b1)
print("\nbeta2: ",b2)
```

```
print("\nbeta3: ",b3)
```

OUTPUT:

```
beta0:  2.9388893694594387

beta1:  0.0457646454553974

beta2:  0.1885300169182036

beta3:  -0.0010374930424761342
```

CODE:

```
y_pred=b0+x1*b1+x2*b2+x3*b3
y_bar=np.mean(y)
ssr=np.sum((y-y_pred)*(y-y_pred))
sst=np.sum((y-y_bar)*(y-y_bar))
acc=1-(ssr/sst)
print("Accuracy Score: ", acc)

var=np.mean(y-y_pred)
MSE=var*var
print("Mean Squared Error: ", MSE)
print("Root Mean Squared Error: ", np.sqrt(MSE))
```

OUTPUT:

```
Accuracy Score:  0.8972106381789522
Mean Squared Error:  2.7144900963881323e-28
Root Mean Squared Error:  1.6475709685437324e-14
```