2020-2024

Practical File

Submitted by: Divija

102018056

COMPUTATIONAL STATISTICS

LAB-BASIC STATISTICS

```
import numpy as np
import pandas as pd
from bisect import bisect_right as upper_bound
import statistics
import math



#random.randit is used to generate random numbers in a particular range
data = np.random.randint(0, 100, size=(5, 3))

#pd.dataframe prints data in tabular form ,thus easy to access
df = pd.DataFrame(data, columns=['random_1', 'random_2', 'random_3'])
print(df)

#iloc helps to extract rows and col to make x array
x=df.iloc[:,:].values

#.shape gives the dimensions of the dataframe
print(x.shape)

n=x.shape[0]
m=x.shape[1]
```

```
#for harmonic mean


sumh=0
for i in range(n):
   sumh=sumh+1/x[i][0]


hm=(n)/sumh
print('harmonic mean by formula is :', round(hm, 2))
print('harmonic mean by inbuilt function is :'
,round(statistics.harmonic_mean(df.random_1), 2))




#for geometric mean


product=1
for i in range(n):
   product=product * x[i][0]


gm = (float)(math.pow(product, (1 / n)))
print('geometeric mean by formula is :', round(gm, 2))
print('geometeric mean by inbuilt function is :'
,round(statistics.geometric_mean(df.random_1), 2))
```

```
#for mean of data


sum =0
for i in range(n):
    for j in range(m):
        sum=sum+x[i][j]
me=sum/(n*m)
print('mean by formula is :', round(me, 2))
print('mean by inbuilt function is :' ,round(np.mean(x), 2))
```

```
#for median of data


#first we have to sort the data
a=np.array(x)
def sortRowWise(a):
    # One by one sort individual rows.
    for i in range(len(a)):
        a[i].sort()
    return 0
```

```
MAX = 100;
# Function to find median in the matrix
def median_using_binary(a, n, m):
    mi = a[0][0]
    mx = 0
    for i in range(n):
        #finding min
        if a[i][0] < mi:
            mi = a[i][0]
        #finding max
        if a[i][m-1] > mx :
            mx =  a[i][m-1]

     #stores the value of median if matrix is put in array format
    desired = (n * m + 1) // 2

    while (mi < mx):
        mid = mi + (mx - mi) // 2
        #place stores the number of elements less than mid in the matrix
        place = [0];

        # Find count of elements smaller than mid
        for i in range(n):
            j = upper_bound(a[i], mid)
            place[0] = place[0] + j
```

```
        if place[0] < desired:
            mi = mid + 1
        else:
            mx = mid
    return mi

#calling of functions
sortRowWise(a)
mi=median_using_binary(a, n, m)



print (''Median by formula is'', mi)
print('median by inbuilt function is :' ,round(np.median(x), 2))
var=0
for i in range(n):
    for j in range(m):
        var=var+(x[i][j]-sum/(n*m))**2







#for variance


print('variance by inbuilt function is',round(np.var(x),2))
```

```
def variance(a, n, m, me):
    sum1 = 0;
    for i in range(n):
        for j in range(m):

            # subtracting mean from elements
            a[i][j] -= me;

            # squaring each terms
            a[i][j] *= a[i][j];

    # taking sum
    for i in range(n):
        for j in range(m):
            sum1 += a[i][j];

    return sum1/(n*m)
variance(x,n,m,me)




print('variance by formula is', round(var/(n*m),2))

def covariance_data():
```

```
data=exercise.iloc[1: , 0:5].values


csum=[]
sum=0

for i in range(data.shape[1]):
   for j in range(data.shape[0]):
       sum+=data[j][i]
   csum.append(sum)
   sum=0
cmean=[]
for i in csum:
   mean=0
   mean=i/data.shape[0]
   cmean.append(mean)

cov_mat=np.empty((data.shape[0],data.shape[1]))
cov=0
for i in range (data.shape[1]):
   for j in range (data.shape[1]):
      for k in range (data.shape[0]):
         cov+=(data[k][i]-cmean[i])*(data[k][j]-cmean[j])
      cov=cov/(data.shape[0]-1)
      cov_mat[i][j]=cov
print(cov_mat)
```

```
   random_1  random_2  random_3
0        52        22         3
1        17        46        95
2        97        34        34
3        53        87        84
4        71        77        35
(5, 3)
harmonic mean by formula is : 41.21
harmonic mean by inbuilt function is : 41.21
geometeric mean by formula is : 50.32
geometeric mean by inbuilt function is : 50.32
mean by formula is : 53.8
mean by inbuilt function is : 53.8
Median by formula is 52
median by inbuilt function is : 52.0
variance by inbuilt function is 837.36
variance by formula is 837.36
```

LAB-GRAPH PLOTTING

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df = pd.read_csv(r'C:\Users\hp\Desktop\Iris.csv')
print(df.head(5))


def simpleline():
    #straight line graph
    plt.title('Linear graph')
    x = np.linspace(0, 10, 50)
    y=np.linspace(0, 5, 50)
    plt.plot(x,"o",y,"r")
    plt.xlabel('x-axis')
    plt.ylabel('Y-axis')
    plt.show()


def trigno():
    x= np.linspace(0,10,50)
    y= np.linspace(0,5,50)
    sin_x = np.sin(x)
    cos_y =np.cos(y)
```

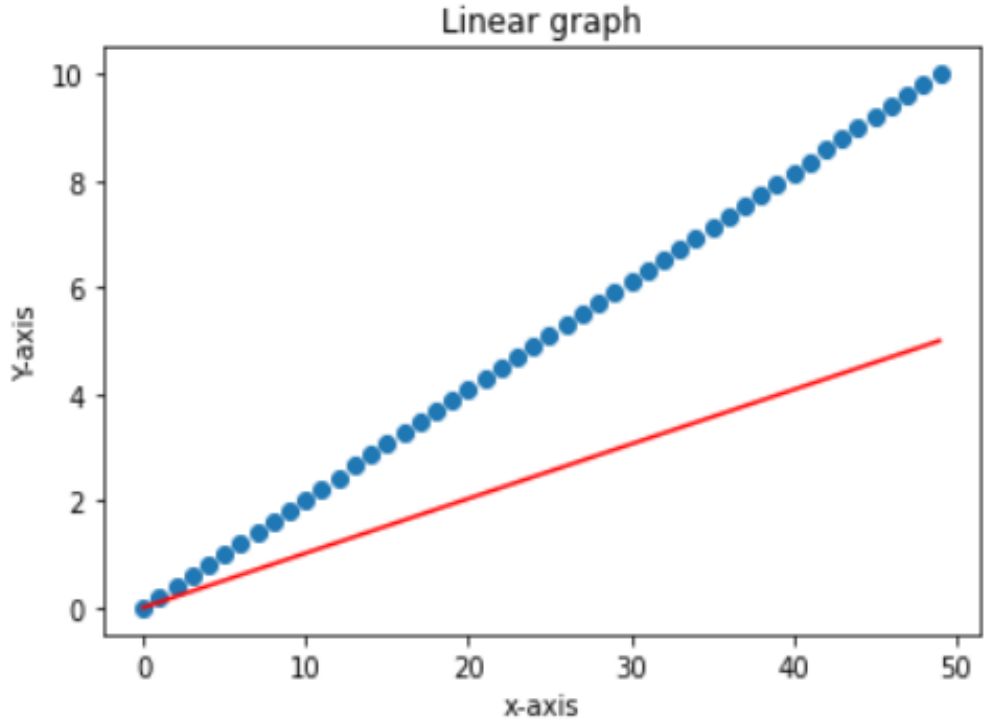
```
    z=sin_x + cos_y
    w=sin_x**2
    plt.title('Trignometry')
    plt.plot(z,label ='z=sin_x+cos_y',color ='green',linestyle ='-')
    plt.plot(sin_x, label='sinwave', color ='red', linestyle = '--')
    plt.plot(cos_y, label='coswave', color ='blue', linestyle = '-')
    plt.plot(w, label='sin_x**2', color ='yellow', linestyle = '-')

    plt.legend()
    plt.show()


def subplots():
    names = ['group_1', 'group_2', 'group_3']
    values = [10, 50, 200]

    plt.figure(figsize=(10, 3))

    #bar-graph
    # 1x3 grid for 1st plot
    plt.subplot(131)
    plt.bar(names, values)

    #scatter plot
    plt.subplot(132)
    plt.scatter(names, values,c=['red'])
```

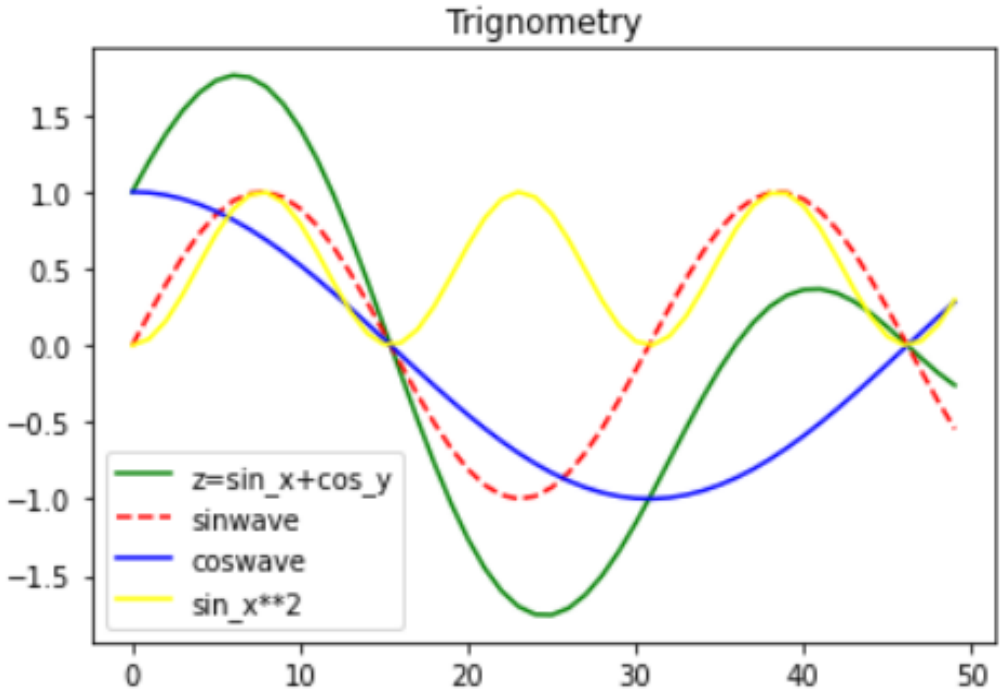
```
    #normal plot
    plt.subplot(133)
    plt.plot(names, values,"g",)


    plt.suptitle('Categorical Plotting')
    plt.show()


def symbols():
    #plot using diff symbols
    #arrange takes values start,stop and step
    t = np.arange(0, 5, 0.2)

    plt.plot(t, t, 'g--', label='linear' )
    plt.plot(t, t**2, 'rs',label='square')
    plt.plot(t, t**3, 'b^',label='cube')
    plt.legend()
    plt.show()


def random_data():
    data1 = np.random.randn(2, 10)
    print(data1)
    print(data1.shape)
```

```
    plt.plot(data1)
    plt.show()



def pie_chart():
    species = ['SETOSA', 'VERSICOlOR', 'VIRGINIA']

    data = [50,50,50]

    # Creating plot

    fig = plt.figure(figsize =(10, 7))
    plt.pie(data, labels = species)
    plt.title('Different species of Irirs')

    # show plot
    plt.show()



def scatterplot_fileread():
    print('Enter any two attributes of the data(SepalWidthCm,SepalLengthCm
,PetalLengthCm,PetalWidthCm )')
    atr1=input()
    atr2=input()
    colors = {'Iris-setosa':'r', 'Iris-virginica':'g', 'Iris-versicolor':'y'}
    if atr1 == 'SepalWidthCm' and atr2 == 'SepalLengthCm' :
        plt.xlabel('SepalWidthCm')
```

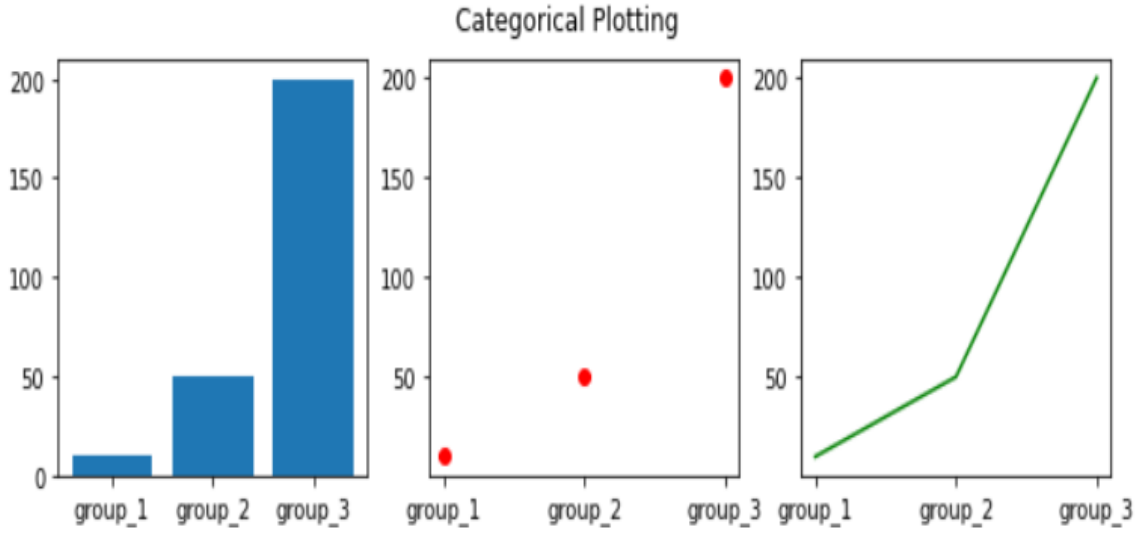
```
        plt.ylabel('SepalLengthCm')
        plt.scatter(df.SepalWidthCm, df.SepalLengthCm,
c=df['Species'].apply(lambda col_vector: colors[col_vector]), s = 100)
    elif atr1 == 'SepalWidthCm' and atr2 == 'PetalLengthCm' :
        plt.xlabel('SepalWidthCm')
        plt.ylabel('PetalLengthCm')
        plt.scatter(df.SepalWidthCm, df.PetalLengthCm,
c=df['Species'].apply(lambda col_vector: colors[col_vector]), s = 100)
    elif atr1 =='PetalWidthCm' and atr2 == 'PetalLengthCm' :
        plt.xlabel('PetalWidthCm')
        plt.ylabel('PetalLengthCm')
        plt.scatter(df.PetalWidthCm, df.PetalLengthCm,
c=df['Species'].apply(lambda col_vector: colors[col_vector]), s = 100)
    elif atr1 == 'PetalWidthCm' and atr2 =='SepalLengthCm' :
        plt.xlabel('PetalWidthCm')
        plt.ylabel('SepalLengthCm')
        plt.scatter(df.PetalWidthCm, df.SepalLengthCm,
c=df['Species'].apply(lambda col_vector: colors[col_vector]), s = 100)


print("\n----------------------------------------------\n")
print('1 Simpleline')
print('2 Trigno_functions')
print('3 Subplots')
print('4 Random_data_plot')
print('5 pie chart')
print('6 scatterplot_fileread')
print("\n----------------------------------------------\n")
```

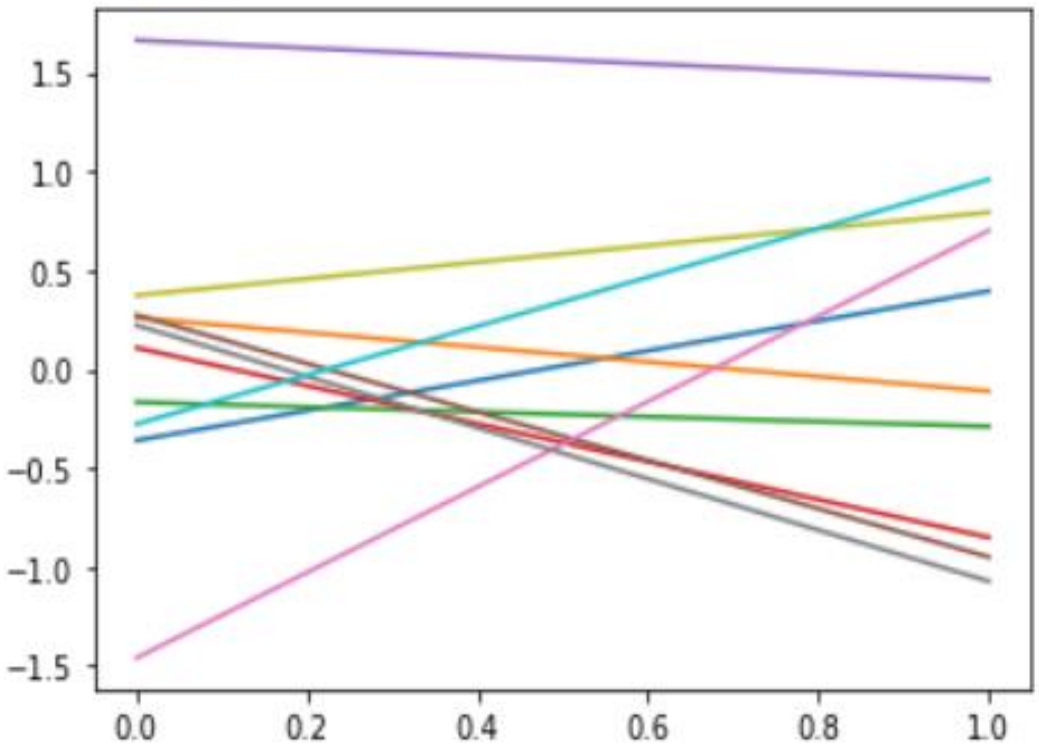
```
x=int(input("Enter your choice: "))

if x==1:
    simpleline()

elif x==2:
    trigno()

elif x==3:
    subplots()

elif x==4:
    random_data()

elif x==5:
    pie_chart()

elif x==6:
    scatterplot_fileread()

else:
    print("Please enter valid choice")
```


Linear graph
Y-axis
x-axis
0
2
4
6
8
10
0
10
20
30
40
50

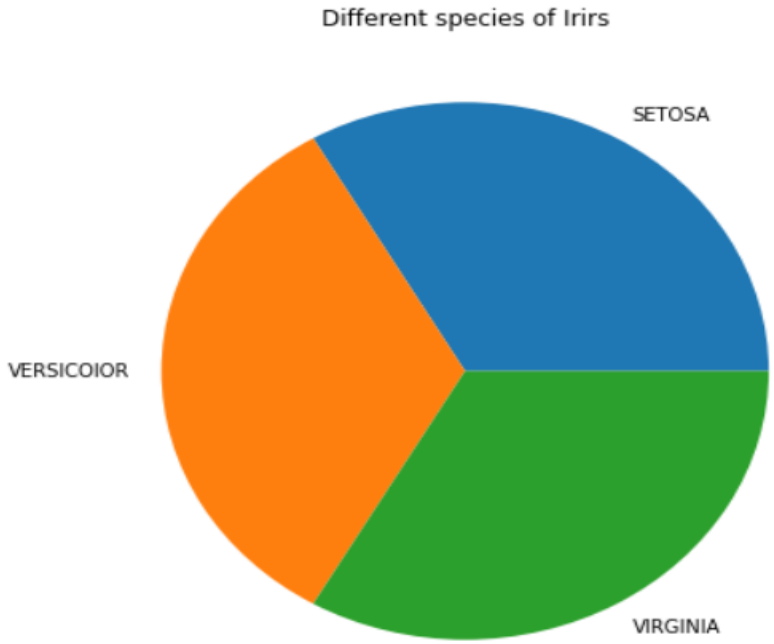

Trignometry
z=sin_x+cos_y
sinwave
coswave
sin_x**2
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
10
20
30
40
50

```
Enter your choice: 3
```

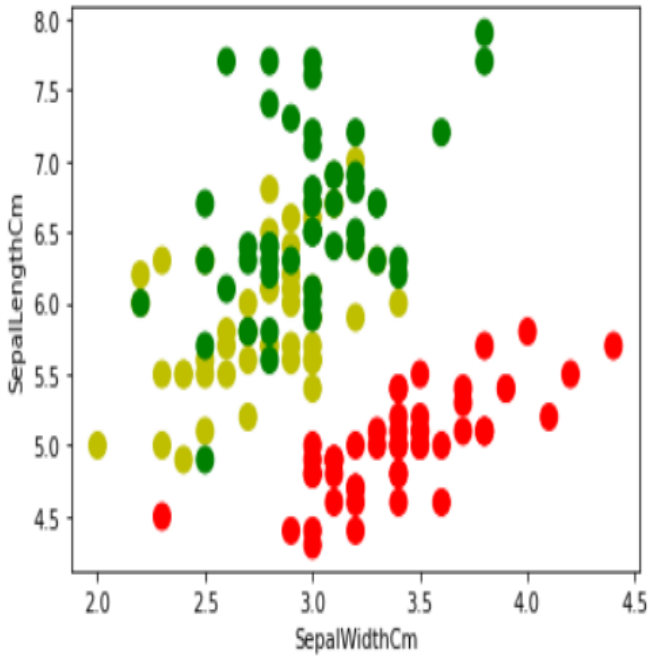




```
Enter your choice: 5
```




```
Enter your choice: 6
Enter any two attributes of the data(SepalWidthCm,SepalLengthCm ,PetalLengthCm,PetalWidthCm )
SepalWidthCm
SepalLengthCm
```



LAB- LINEAR REGRESSION

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data = pd.read_csv("headbrain.csv")

#head size is independent
#brain weight is dependent

#values() returns a list of all the values available in given data
x = data['Head Size(cm^3)'].values
y = data['Brain Weight(grams)'].values
print(data.head())

#237 input values
# total no.of input values
l = len(x)

sum_x =0
sum_y =0

# calculating the mean of x and y
for i in range(l):
    sum_x += x[i]
```

```
mean_x = sum_x/l

for i in range(l):
    sum_y += y[i]

mean_y = sum_y/l

# using the formula to calculate m & c

numer = 0
denom = 0
for i in range(l):

    numer += (x[i] - mean_x) * (y[i] - mean_y)
    denom += (x[i] - mean_x) ** 2

m = numer / denom
c = mean_y - (m * mean_x)

print('The value of m =' ,m)
print('The value of c =' ,c)

#m is b1 and c is b0
# y=b0 +b1*x

y_reg=[]
```

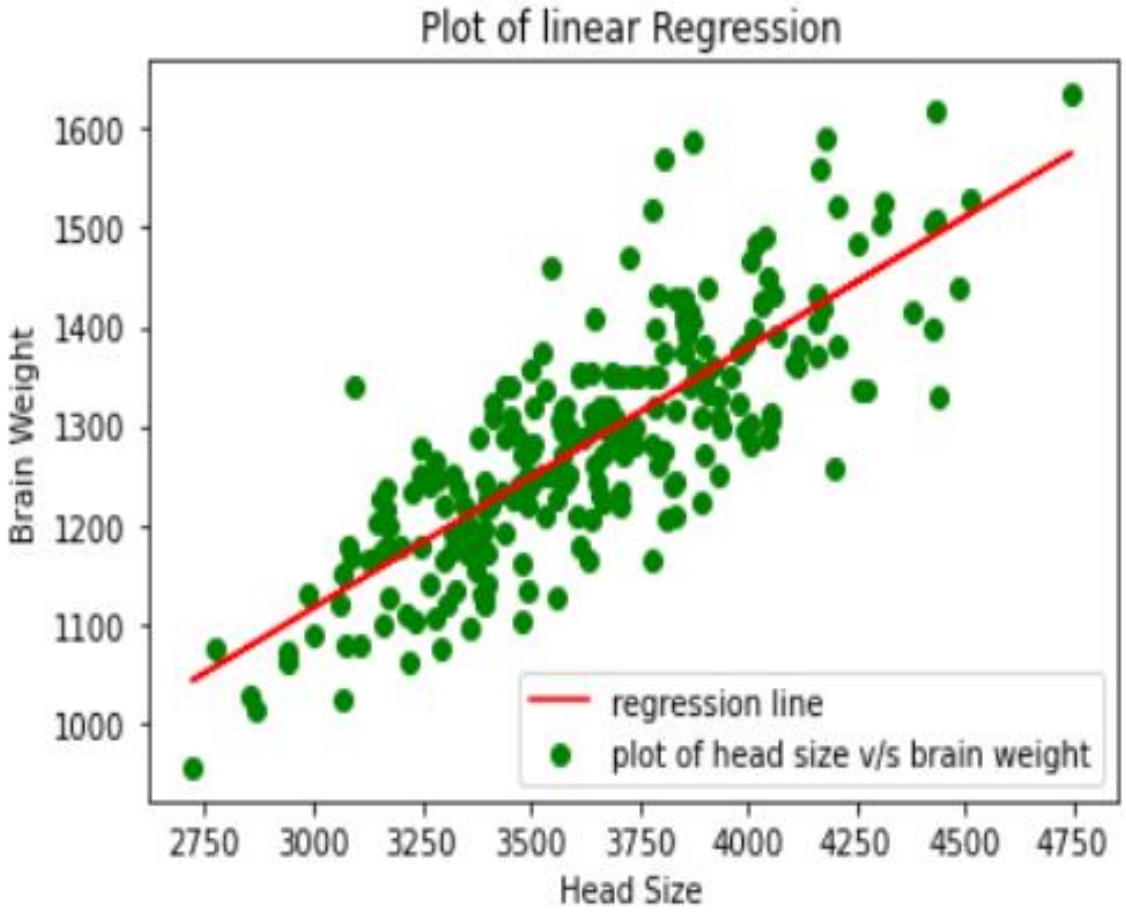
```
y_reg= c+(m*x)

plt.figure(figsize =(10,8))

plt.scatter(x,y, color ="green" ,label ="plot of head size v/s brain weight")
plt.plot(x,y_reg ,color ="red" , label ="regression line")

plt.xlabel("Head Size")
plt.ylabel("Brain Weight")
plt.legend()

plt.title("Plot of linear Regression")
plt.show()

from sklearn.linear_model import LinearRegression
Reg_model = LinearRegression()
x=np.array(x)
y=np.array(y)

#array will get reshaped in such a way that the resulting array has only 1 column
#no reshaping,error while fitting it in the model.
x=x.reshape(-1,1)
y=y.reshape(-1,1)

#takes array as an input
Reg_model.fit(x,y)
```

```
Reg_model.coef_

Reg_model.intercept_

# calculating R-squared value for measuring goodness of our model.

ss_t = 0 #total sum of squares
ss_r = 0 #total sum of square of residuals

for i in range(len(x)):

    y_pred = c + m * x[i]
    ss_t += (y[i] - mean_y) ** 2
    ss_r += (y[i] - y_pred) ** 2

r2 = 1 - (ss_r/ss_t)

print(r2)
```

```
The value of m = 0.26342933948939945
The value of c = 325.57342104944223
```

```
In [22]: Reg_model.coef_
Out[22]: array([[0.26342934]])
```

```
In [23]: Reg_model.intercept_
Out[23]: array([325.57342105])
```



LAB MULTICOLLINEARITY AND VIF

```
import pandas as pd
import math

data=pd.read_csv(r'C:\Users\hp\Desktop\cs_data.csv')
data.head()

df=data.iloc[:,0:4]
print(df)

# data = data.drop(columns="sales")
# print(data)

#using karl pearson method to calculate VIF
def calculate(x1,x2):
    n = len(x1)
    sum_x1 = 0
    sum_sq_x1 = 0
    sum_x2 = 0
    sum_sq_x2 = 0
    sum_x1_x2=0


    for i in range(n):
```

```
        sum_x1 += x1[i]
        sum_sq_x1 +=(x1[i]*x1[i])
        sum_x2 += x2[i]
        sum_sq_x2 +=(x2[i]*x2[i])
        sum_x1_x2 +=x1[i]*x2[i]



    r=0
    num =0
    den= 0

    num = (n*sum_x1_x2) - (sum_x1*sum_x2)
    den = math.sqrt((n*sum_sq_x1)-(sum_x1**2))*math.sqrt((n*sum_sq_x2)-
(sum_x2**2))

    r=num/den

    r_sq = r*r
    print('r_sq=', r_sq)

    vif=1/(1-r_sq)
    print('vif =' ,vif)

    for i in range(df.shape[1]):
        for j in range(i+1,df.shape[1]):
            calculate(df.iloc[:,i],df.iloc[:,j])
```

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant


vif = pd.DataFrame()
vif["features"] = df.columns
vif["VIF Factor"] = [variance_inflation_factor(df.values, i)
                     for i in range(df.shape[1])]


print(vif)
```

	resudial sugar	density	ph	sulphate
0	-0.758	-1.146	-1.968	0.828
1	-0.741	-0.935	1.835	0.236
2	0.237	0.546	-0.032	0.236
3	0.980	1.076	-0.378	-0.650
4	-0.842	-1.993	0.382	-1.686
5	-0.707	0.546	0.175	-0.207
6	-0.741	-0.935	1.835	0.236
7	-0.741	-0.829	-0.170	0.680
8	0.237	0.546	-0.032	0.236
9	-0.876	-0.829	0.451	1.124
10	-0.539	-0.194	0.382	2.011
11	0.169	0.917	-0.931	0.384
12	1.621	1.393	-0.585	-1.538
13	2.600	1.023	0.521	-0.354
14	0.102	0.811	-1.484	-1.538

```
r_sq= 0.5848383739781509
vif = 2.4087004610280913
r_sq= 0.026415119168281372
vif = 1.027131809139646
r_sq= 0.16101118187073493
vif = 1.1919109985634249
r_sq= 0.1082040325715211 4
vif = 1.121332722420276
r_sq= 0.066686392929948
vif = 1.0714512168522823
r_sq= 0.02977681082773628 7
vif = 1.0306906814432462
```

LAB MULTIPLE LINEAR REGRESSION

```
import pandas as pd
import math
import numpy as np
from statistics import stdev
data=pd.read_csv('./bodyfat.csv')
data.head()

df=data.iloc[:,:]
print(df.head())

df_normalized = df.copy()
for i in df_normalized.columns:
    df_normalized[i] = (df_normalized[i] - df_normalized[i].mean()) /
df_normalized[i].std()

df=df_normalized
df1=df_normalized.iloc[:,1:6]
X = df1.to_numpy()
print(len(X))
ALL_ONE=[1]*len(X)
X1=np.insert(X,0,ALL_ONE,axis=1)
print(X1)
```

```
y=df_normalized.iloc[:,0]

Y=y.to_numpy()

X1_T = X1.T

print(X1_T.shape[0])

X1_mul = np.matmul(X1_T,X1)

print(X1_mul.shape[0])

print(X1_mul.shape[1])

X_inv = np.linalg.inv(X1_mul)

print(X_inv.shape[0])

print(X_inv.shape[1])

X_final = np.matmul(X_inv, X1_T)

b=np.matmul(X_final,Y)

Y_pred = []

for i in range(len(Y)):

   Y_pred=b[0] +
b[1]*df_normalized.iloc[:,1][i]+b[2]*df_normalized.iloc[:,2][i]+b[3]*df_normali
zed.iloc[:,3][i]+b[4]*df_normalized.iloc[:,4][i]+b[5]*df_normalized.iloc[:,5]

print(Y_pred)

e=0

for i in range(len(Y)):

   e+=(Y[i]-Y_pred[i])**2

print(e/len(Y))
```

	BodyFat	Age	Weight	Height	Abdomen	Thighs
0	12.3	23	154.25	67.75	36.2	93.1
1	6.1	22	173.25	72.25	38.5	93.6
2	25.3	22	154.00	66.25	34.0	95.8
3	10.4	26	184.75	72.25	37.4	101.8
4	28.7	24	184.25	71.25	34.4	97.3

```
      BodyFat       Age    Weight    Height   Abdomen    Thighs
0   -0.818617 -1.736617 -0.839575 -0.654901 -0.737198 -0.916224
1   -1.559469 -1.815970 -0.193078  0.573648  0.208949 -0.856916
2    0.734783 -1.815970 -0.848082 -1.064418 -1.642207 -0.595958
3   -1.045652 -1.498561  0.198223  0.573648 -0.243556  0.115746
4    1.141057 -1.657265  0.181210  0.300637 -1.477660 -0.418032
..        ...       ...       ...       ...       ...       ...
247 -0.973957  1.992938 -1.520098 -0.859660 -1.271976 -1.378832
248  1.726569  2.151642  0.751148 -0.108879  1.196232  0.910482
249  1.212752  2.151642  0.266275 -1.132671  0.373496  1.218887
250  0.818427  2.151642  0.402379  0.095879  0.373496  0.886758
251  1.523432  2.310347  0.972318 -0.040627  1.155095  1.373089

[252 rows x 6 columns]
```

```
[[ 1.         -1.73661733 -0.83957503 -0.65490144 -0.73719764 -0.91622428]
 [ 1.         -1.81596957 -0.19307815  0.57364816  0.20894884 -0.85691564]
 [ 1.         -1.81596957 -0.84808157 -1.06441797 -1.64220731 -0.59595765]
 ...
 [ 1.          2.15164211  0.26627489 -1.13267073  0.37349606  1.2188866 ]
 [ 1.          2.15164211  0.40237949  0.09587887  0.37349606  0.88675824]
 [ 1.          2.31034658  0.97231753 -0.04062664  1.15509532  1.37308905]]
```

```
print(b[0])
```

```
[ 6.11316553e-15  1.99760068e-01  4.05870323e-01 -1.94345010e-01
 -2.06892100e-01  4.93039067e-01]
6.113165529342268e-15
```

```
print(Y_pred)
```

```
0      0.173331
1      0.202572
2      0.331235
3      0.682133
4      0.418959
         ...
247   -0.054753
248    1.073968
249    1.226024
250    1.062272
251    1.302052
Name: Thighs, Length: 252, dtype: float64
```

Error:

```
0.9387720798503724
```