COMPUTATIONAL STATISTICS LAB-CS261

Aim:

1.Write a python program to find the best fit straight line and draw the scatter plot.

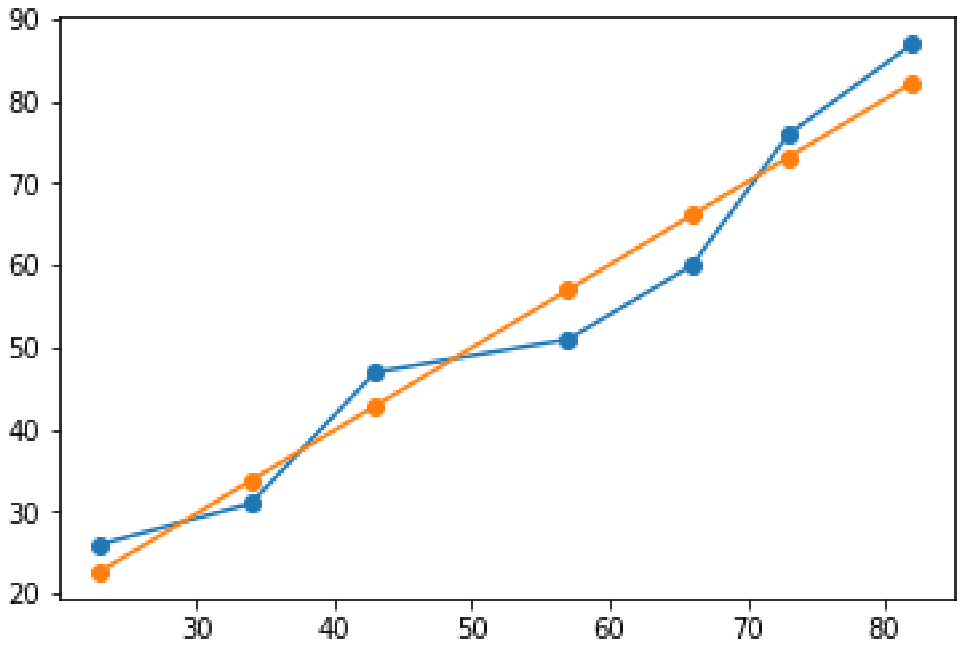
Source Code:

```
import pandas as pd
import matplotlib.pyplot as plt
df=pd.read_csv("C:\\Users\\y20cs170\\Desktop\\data.csv")
print(df)
x=df['col1']
y=df['col2']
def summation(l):
  sum=0
  for i in l:
    sum=sum+i
  return sum
sumx=summation(x)
print('sum of x values=',sumx)
sumy=summation(y)
print('sum of y values=',sumy)
sumxx=summation(x*x)
print('sum of x2 values=',sumxx)
sumxy=summation(x*y)
print('sum of xy values=',sumxy)
slope=((len(x)*sumxy)-(sumx*sumy))/((len(x)*sumxx)-(sumx)**2)
print(‘slope=’,slope)
intercept=((sumxx*sumy)-(sumx*sumxy))/((len(x)*sumxx)-(sumx)**2)
print(intercept)
yexp=slope*x+intercept
ycal=df['col2']
plt.plot(x,ycal)
```

```
plt.scatter(x,ycal)
plt.plot(x,yexp)
plt.scatter(x,yexp)
plt.show()
```

<u>output:</u>

```
  col1 col2
0 23 26
1 34 31
2 43 47
3 57 51
4 66 60
5 73 76
6 82 87
sum of x values= 502
sum of y values= 528
sum of x2 values= 30304
sum of xy values= 31432
slope=0.965279410612117
intercept=4.34297358727173
```



#error calculation

```
import statistics
ybar = statistics.mean(yexp)
residual=summation((yexp-ycal)**2)
sst=summation((yexp-ybar)**2)
r2=1-(residual/sst)
print(r2)
if r2>0.90:
  print("its a good fit")
else:
  print("its not a good fit")
```

0.9508238244196502

Output:

its a good fit

Aim:

2.Write a python program to fit a second degree parabola of the form y=a+bx+cx2 and draw the scatter plot.

Source Code:

```
import pandas as pd
import numpy as np
df=pd.read_csv("data.csv")
print(df)
   x   y
0  0  1.0
1  1  1.8
2  2  1.3
3  3  2.5
4  4  6.3
df["xy"]=df["x"]*df["y"]
df["x2y"]=(df["x"]**2)*df["y"]
df["x3"]=df["x"]**3
df["x4"]=df["x"]**4
sum_x=sum(df["x"])
sum_y=sum(df["y"])
sum_xy=sum(df["xy"])
sum_x2y=sum(df["x2y"])
sum_x3=sum(df["x3"])
sum_x4=sum(df["x4"])
sum_x2=sum(df["x"]**2)
#by crammer's rule
m1=[len(df["x"]),sum_x,sum_x2]
m2=[sum_x,sum_x2,sum_x3]
m3=[sum_x2,sum_x3,sum_x4]
m4=[sum_y,sum_xy,sum_x2y]
delta=np.linalg.det(np.array([m1,m2,m3]))
```

delta1=np.linalg.det(np.array([m4,m2,m3]))

delta2=np.linalg.det(np.array([m1,m4,m3]))

delta3=np.linalg.det(np.array([m1,m2,m4]))

a=delta1/delta

b=delta2/delta

c=delta3/delta

y_obs=a+b*df["x"]+c*(df["x"]**2)

print(y_obs)

0 1.42

1 0.90

2 1.48

3 3.16

4 5.94

Name: x, dtype: float64

import matplotlib.pyplot as plt

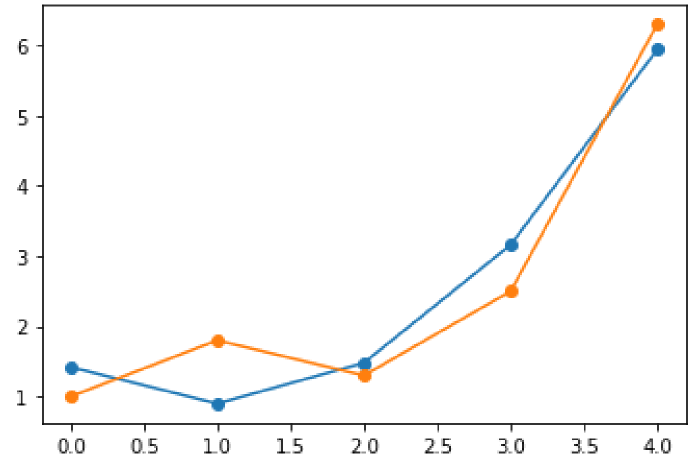
plt.plot(df["x"],y_obs)

plt.scatter(df["x"],y_obs)

plt.plot(df["x"],df["y"])

plt.scatter(df["x"],df["y"])

plt.show()



```
#error calculation
import statistics
ybar = statistics.mean(df["y"])
residual=sum((df["y"]-y_obs)**2)
sst=sum((df["y"]-ybar)**2)
r2=1-(residual/sst)
print(r2)
if r2>0.90:
  print("its a good fit")
else:
  print("its not a good fit")
```

```
0.9147837314396385
its a good fit
```

Aim:

3.Write a python program to find Karl Pearson's correlation coefficient.

Source Code:

```
import pandas as pd
import math

df=pd.read_csv("C:\\Users\\y20cs170\\Desktop\\relation.csv")
print(df)
def summation(l):
    sum=0
    for i in l:
        sum=sum+i
    return sum
   col1  col2
0    56    58
1    12    10
2    23    25
3    67    69
4    79    75
5    99    94
6    45    43
x=df['col1']
y=df['col2']
sumx=summation(x)
print('sum of x values=',sumx)
sumy=summation(y)
print('sum of y values=',sumy)
sumxx=summation(x*x)
print('sum of x2 values=',sumxx)
sumxy=summation(x*y)
```

```
print('sum of xy values=',sumxy)
sum of x values= 381
sum of y values= 374
sum of x2 values= 26365
sum of xy values= 25732
xbar=sumx/len(x)
ybar=sumy/len(y)
print(xbar)
print(ybar)
54.42857142857143
53.42857142857143
#numerator
devx=0
devy=0
devx=x-xbar
devy=y-ybar
com_dev=devx*devy
dev=summation(com_dev)
print(dev)
5375.714285714286
#denominator
devx=devx**2
devy=devy**2
dx=summation(devx)
dy=summation(devy)
deno=math.sqrt(dx)*math.sqrt(dy)
print(deno)
5398.027107477437
 print(dev/deno)
0.9958664857884387
```

Aim:

4.Write a python program to find the Spearman's correlation coefficient between x and y variables.

Source Code:

```
import pandas as pd

d=pd.read_excel("C:\\Users\\y20cs170\\Desktop\\cs.xlsx")

df=pd.DataFrame(d)

df

x	y

0	68	62

1	64	58

2	75	68

3	50	45

4	64	81

5	80	60

6	75	68

7	40	48

8	55	50

9	64	70

def rank(x):

   n=len(x)

   rankx=[]

   for i in range (n):

      r=1

      s=1

      for j in range (i):

         if x[j] > x[i]:

            r= r+1

         if x[j] == x[i]:

            s= s+1

      for j in range (i+1,n):
```

```
            if x[j] > x[i]:
                r = r+1
            if x[j] == x[i]:
                s = s+1
        rankx.append(r + (s-1) * 0.5)
    return rankx
df['r1']=rank(df['x'])
df['r2']=rank(df['y'])
df['d']=df['r1']-df['r2']
df['dsq']=df['d']**2
df
```

	x	y	r1	r2	d	dsq
0	68	62	4.0	5.0	-1.0	1.0
1	64	58	6.0	7.0	-1.0	1.0
2	75	68	2.5	3.5	-1.0	1.0
3	50	45	9.0	10.0	-1.0	1.0
4	64	81	6.0	1.0	5.0	25.0
5	80	60	1.0	6.0	-5.0	25.0
6	75	68	2.5	3.5	-1.0	1.0
7	40	48	10.0	9.0	1.0	1.0
8	55	50	8.0	8.0	0.0	0.0
9	64	70	6.0	2.0	4.0	16.0

```
sumdisq=sum(df['dsq'])
sumdisq
```

72.0

```
count1={}
count2={}
for i in df['x']:
    if i not in count1:
        count1[i]=1
    else:
```

```
        count1[i]+=1
for i in df['y']:
    if i not in count2:
        count2[i]=1
    else:
        count2[i]+=1
print(count1,count2)
{68: 1, 64: 3, 75: 2, 50: 1, 80: 1, 40: 1, 55: 1} {62: 1, 58: 1, 68: 2, 45: 1, 81: 1, 60: 1, 48: 1, 50: 1, 70: 1}
cf=0
for val in count1.values():
    if (val>1):
        cf+=val*(val*val-1)/12
for val in count2.values():
    if (val>1):
        cf+=val*(val*val-1)/12
cf
3.0
sumdisq+=cf
sumdisq
75.0
n=len(df['x'])
row=1-(6*sumdisq/(n*((n*n)-1)))
row
Rank coefficient:0.5454545454545454
```

Aim:

5.Write a python program to classify the data based on one way Anova.

Source Code:

```
fa=[13,10,8,11,8]
fb=[13,11,14,14]
fc=[4,1,3,4,2,4]
ati=sum(fa)
print(ati)
bti=sum(fb)
print(bti)
cti=sum(fc)
print(cti)
ti=ati+bti+cti
a=len(fa)
print(a)
len_fb=len(fb)
len_fc=len(fc)
tisq_ni=(ati*ati/a)+(bti*bti/len_fb)+(cti*cti/len_fc)
print(tisq_ni)
print(ti)
rss=0
for i in fa:
  rss+=i**2
for i in fb:
  rss+=i**2
for i in fc:
  rss+=i**2
print(rss)
len=len(fa)+len(fb)+len(fc)
cf=ti**2/len
print(cf)
sst=rss-cf
```

```
sstr=tisq_ni-cf
sse=sst-sstr
doftr=2
dofe=len-3
msos=sstr/doftr
msose=sse/dofe
if msos>msose:
  f=msos/msose
else:
  f=msose/msos
print(f)
los=0.05
tab=stats.f.ppf(1-los,doftr,dofe)
print(tab)
```

output:

50.625

Aim:

6.Write a python program to classify the data based on two way Anova.

Source Code:

```
d=pd.read_excel("/content/data.csv.xlsx")

df=pd.DataFrame(d)

t1sq=sum(df.loc[0])**2

t2sq=sum(df.loc[1])**2

t3sq=sum(df.loc[2])**2

b1sq=sum(df['b1'])**2

b2sq=sum(df['b2'])**2

b3sq=sum(df['b3'])**2

b4sq=sum(df['b4'])**2

g=sum(df.loc[0])+sum(df.loc[1])+sum(df.loc[2])

tisq=t1sq+t2sq+t3sq

bjsq=b1sq+b2sq+b3sq+b4sq

rss=0

c=0

h=0

for i in df.loc[0]:

  rss+=i**2

  c=c+1

  h+=1

for i in df.loc[1]:

  rss+=i**2

  c+=1

for i in df.loc[2]:

  rss+=i**2

  c+=1

cf=g**2/c

stsq=rss-cf

k=c//h

strsq=(tisq/h)-cf
```

```
sbsq=(bjsq/k)-cf
sesq=stsq-sbsq-strsq
num1=strsq/(k-1)
num2=sbsq/(h-1)
den=sesq/((k-1)*(h-1))
ftr=num1/den
fb=num2/den
print("cal value for treatments",ftr)
print("cal value for blocks",fb)
l=0.05
ftrtab=stats.f.ppf(1-l,k-1,(k-1)*(h-1))
fbtab=stats.f.ppf(1-l,h-1,(k-1)*(h-1))
print("tab value for treatments",ftrtab)
print("tab value for blocks",fbtab)
if ftr<ftrtab:
  print("there is homogenity among treatments")
else:
  print("there is no homogenity among treatments")
if fb<fbtab:
  print("there is homogenity among blocks")
else:
  print("there is no homogenity among blocks")
```

Output:

```
cal value for treatments 3.230769230769231
cal value for blocks 3.4615384615384617
tab value for treatments 5.143252849784718
tab value for blocks 4.757062663089414
there is homogenity among treatments
there is homogenity among blocks
```

Aim:

7.Write a python program to fit a multiple regression model for any given data.

Source Code:

```
import pandas as pd
import numpy as np
import statistics
import math as m
df=pd.read_csv("C:\\Users\\y20cs142\\Desktop\\data.csv")
#finding equation
y_obs = np.transpose([df['y']]) #should use trans otherwise it appends nrmlly row wise
df['c'] = 1 #adding a 1's column temporarily to get x vector
x_t = np.array( [ df['c'], df['x1'], df['x2'] ] ) #x_transpose - as the np.array appends row wise
x = np.transpose( x_t ) #x - x_transpose transpose
first_part = np.linalg.inv( np.matmul ( np.transpose(x), x) ) #inverse(x_transpose*x)
second_part = np.matmul( x_t, y_obs ) #x_transpose*y
beta = np.matmul(first_part, second_part)
beta
y_fitted = beta[0] + (beta[1] * df['x1']) + (beta[2] * df['x2']) #equation beta0+beta1*x1+beta2*x2
y_fitted = list( y_fitted ) #for further calculations
#y_fitted
#testing goodness of fit using coefficient of determination
df['y_fitted'] = y_fitted #appending fitted value to df
df['error'] = df['y'] - df['y_fitted'] #finding e - error
sse=sum ( df ['error'] ** 2)
print("sse: \n",sse)
y_bar = statistics.mean( df['y'] )
sst = sum( (df['y']-y_bar) ** 2)
print("sst: \n",sst)
ssr = sst - sse
print("ssr: \n",ssr)
```

```
r_square = ssr / sst
if(r_square > 0.90):
   print("the model is good fit and r^2 value is: ",r_square)
else:
   print("the model is not a good fit and r^2 value is: ",r_square)
df
```

sse:

1.7142857142857144

sst:

73.71428571428571

ssr:

72.0

the model is good fit and r^2 value is: 0.9767441860465117

	x1	x2	y	c	y_fitted	error
0	-5	5	11	1	11.571429	-0.571429
1	-4	4	11	1	10.571429	0.428571
2	-1	1	8	1	7.571429	0.428571
3	2	-3	2	1	2.571429	-0.571429
4	2	-2	5	1	4.571429	0.428571
5	3	-2	5	1	5.571429	-0.571429
6	3	-3	4	1	3.571429	0.428571

```
#goodness of fit using ANOVA model
```

```
dof_reg = len(beta) - 1 # dof of regression = k - 1

dof_error = len(df) - len(beta) #dof of regreesion = n - k

msr_square = ssr / dof_reg #mean sum of squares

mse_square = sse / dof_error # mean sum of squares for error

f = msr_square / mse_square

# creation of anova table

anova_table = {"source of variation" : ["regression", "error" , "total"] , "sum of squares" : [ssr , sse , sst] , "mean sum of squares" : [msr_square , mse_square , " "] , "variance ratio" : [" ", f , " "]}

anova = pd.DataFrame(anova_table)

print(anova)

import scipy.stats

f_table = scipy.stats.f.ppf( q=1-.05 , dfn=dof_reg, dfd=dof_error) #getting table value of f

#inference

if(f > f_table):

   print("we accept the model")

else:

   print("we reject the model")
```

```
  source of variation  sum of squares  mean sum of squares   variance ratio

0        regression       72.000000             36.0

1            error         1.714286          0.428571                   84.0

2            total      73.714286

we accept the model
```

```
# test of indudival parameter

cjj = []

rows , col = first_part.shape

for i in range(rows):

   for j in range(col):

      if(i==j):

         cjj.append(first_part[i,j]) #coefficient matrix of diagonal elements
```

```
stand_error = []
#calculation of standard error coloumn

for i in cjj:
    stand_error.append( m.sqrt (mse_square * i ) )
t=[]

#calculation of t coloumn
for i in range(len(beta)):
    t.append(int(beta[i] / (stand_error[i])))

# making the table
test_table = {"predictor" : ["beta 0", "beta 1" , "beta 2"] , "coefficient" :  list(beta) , "standard error" : stand_error , "t" : t}
test_table = pd.DataFrame(test_table)
print(test_table)
  predictor         coefficient  standard error   t
0    beta 0  [6.571428571428571]       0.247436  26
1    beta 1  [1.000000000000007]       0.464621   2
2    beta 2               [2.0]       0.464621   4

#inference

t_table = scipy.stats.t.ppf(q=1-.05/2,df=dof_error)
print(t_table)
for i in range(len(beta)):
    if abs(t[i]) > t_table:
        print("we reject beta",i)
        print("its contributing the model")
    else:
        print("we accept beta",i)
```

```
    print("its not contributing the model")
2.7764451051977987
we reject beta 0
its contributing the model
we accept beta 1
its not contributing the model
we reject beta 2
its contributing the model
```

Aim:

8.Write a python program to fit a multivariate regression model for any given data

Source Code:

```
df=pd.read_csv(&quot;data.csv&quot;);
df
```

	y1	y2	x1	x2	x3
0	10	100	9	62	1.0
1	12	110	8	58	1.3
2	11	105	7	64	1.4
3	9	94	14	60	0.8
4	9	95	12	63	0.8
5	10	99	10	57	0.9
6	11	104	7	55	1.0
7	12	108	4	56	1.2
8	11	105	6	59	1.1
9	10	98	5	61	1.0
10	11	103	7	57	1.2
11	12	110	6	60	1.2

```
y=np.transpose(np.array([df["y1"],df["y2"]]))
df["c1"]=1
x=np.transpose(np.array([df["c1"],df["x1"],df["x2"],df["x3"]]))
first=np.linalg.inv(np.dot(np.transpose(x),x))
second=np.dot(np.transpose(x),y)
beta=np.dot(first,second)
y1=beta[0][0]+beta[1][0]*df["x1"]+beta[2][0]*df["x2"]+beta[3][0]*df["x3"]
```

```
y2=beta[0][1]+beta[1][1]*df["x1"]+beta[2][1]*df["x2"]+beta[3][1]*df["x3"]
#goodness of fit
df["yo1"]=y1
df["yo2"]=y2
df["e1"]=df["y1"]-df["yo1"]
df["e2"]=df["y2"]-df["yo2"]
y1bar=statistics.mean(df["y1"])
y2bar=statistics.mean(df["y2"])
#calculations for y1
e1=transpose(np.array([df["e1"]]))
sse_1=np.dot(np.transpose(e1),e1)
sst_1=sum((df["y1"]-y1bar)**2)
ssr_1=sst_1-sse_1
r1=ssr_1/sst_1
#calculations for y2
e2=np.transpose(np.array(df["e2"]))
sse_2=np.dot(np.transpose(e2),e2)
sst_2=sum((df["y2"]-y2bar)**2)
ssr_2=sst_2-sse_2
r2=ssr_2/sse_2

#inference
if(r1>0.90):
  print("r1 is good fit")
if(r2>0.90):
  print("r2 is good fit")
else:
  print("its not a good fit")
```

r2 is good fit

Aim:

9.Write a python program to classify the treatments based on MANOVA Test.

Source Code:

```
import pandas as pd

from scipy import  stats

import scipy.stats

import numpy as np

t11=np.asarray([9,3])

t12=np.asarray([6,2])

t13=np.asarray([9,7])

t21=np.asarray([0,4])

t22=np.asarray([2,0])

t31=np.asarray([3,8])

t32=np.asarray([1,9])

t33=np.asarray([2,7])

t1=[t11,t12,t13]

t2=[t21,t22]

t3=[t31,t32,t33]

t1=np.asarray(t1)

t2=np.asarray(t2)

t3=np.asarray(t3)

print(t3)

lentr=3

t4=[t1,t2,t3]

print(t1[0])

y1bar=[]

y2bar=[]

sum=0

c=0

s=0
```

```
tot=0
sum1=sum2=0
for t in t4:
  for j in range(len(t)):
    sum+=t[j][c]
    sum1+=sum
    s+=t[j][c+1]
    sum2+=s
    tot+=(j+1)
  y1bar.append(sum//(j+1))
  y2bar.append(s//(j+1))
  sum=0
  s=0
ybb=[]
ybb.append(sum1//tot)
ybb.append(sum2//tot)
table={"treatments":["t1","t2","t3"],"values":[t1,t2,t3],"yibar":[y1bar,y2bar,""],"ybb":["",ybb,""]}
table=pd.DataFrame(table)
print(table)
  treatments                  values          yibar     ybb
0        t1       [[9, 3], [6, 2], [9, 7]]    [8, 1, 2]
1        t2         [[0, 4], [2, 0]]           [4, 2, 8]  [4, 5]
2        t3       [[3, 8], [1, 9], [2, 7]]
#FOR y1
ssey1=0
ssty1=0
p=0
for t in t4:
  for j in range(len(t)):
    ssey1+=(t[j][0]-y1bar[p])**2
    ssty1+=(t[j][0]-ybb[0])**2
```

```
  p+=1
ssry1=ssty1-ssey1
print(ssey1,ssty1,ssry1)
10 88 78
#FOR y2
ssey2=0
ssty2=0
p=0
for t in t4:
  for j in range(len(t)):
    ssey2+=(t[j][1]-y2bar[p])**2
    ssty2+=(t[j][1]-ybb[1])**2
  p+=1
ssry2=ssty2-ssey2
print(ssey2,ssty2,ssry2)
24 72 48
#cross product of y1 and y2
sse=sst=0
p=0
for t in t4:
  for j in range(len(t)):
    sse+=(t[j][0]*t[j][1]-y1bar[p]*y2bar[p])
    sst+=(t[j][0]*t[j][1]-ybb[0]*ybb[1])
  p+=1
ssr=sst-sse
print(sse,sst,ssr)
1 -11 -12
#monava table
import math
b=np.matrix([[ssry1,ssr],[ssr,ssry2]])
w=np.matrix([[ssey1,sse],[sse,ssey2]])
```

```
t=np.matrix([[ssty1,sst],[sst,ssty2]])

df1=len(t4)-1

df3=len(t4[0])+len(t4[1])+len(t4[2])-1

df2=df3-df1

wv=np.linalg.det(w)/np.linalg.det(t)

f=((df2-1)/(df1))*((1-math.sqrt(wv))/math.sqrt(wv))

monava_table={"source of variation":["regression","error","total"],"sum of squares":[b,w,t],"degrees of freedom":[df1,df2,df3],"wilks value":["",wv,""],"f-statistic":["",f,""]}

monava_table=pd.DataFrame(monava_table)

print(monava_table)

  source of variation                    sum of squares  degrees of freedom  \

0         regression  [[[[[ 78 -12]]]], [[[[-12  48]]]]]                  2

1              error      [[[[[10  1]]]], [[[[ 1 24]]]]]                  5

2              total  [[[[[ 88 -11]]]], [[[[-11  72]]]]]                  7


  wilks value f-statistic

0

1    0.038455     8.19886

2

#inference

tab_val=scipy.stats.f.ppf(1-0.05,df1,df2)

print(tab_val)

if(f>tab_val):

  print("we reject h0")

else:

  print("we accept h0")

5.786135043349964

we reject h0
```

Aim:

10.Write a python program to classify the given observations using Linear Discriminant Analysis.

discriminant analysis using regression

Source Code:

```
import pandas as pd
from scipy import  stats
import scipy.stats
import numpy as np
sat=np.asarray([1300,1260,1220,1180,1060,1140,1100,1020,980,940])
gpa=np.asarray([2.7,3.7,2.9,2.5,3.9,2.1,3.5,3.3,2.3,3.1])
grad=["yes","yes","yes","yes","yes","no","no","no","no","no"]
l=[1,1,1,1,1,1,1,1,1,1]
for i in range(len(grad)):
  if grad[i]=="yes":
    grad[i]=1
  else:
    grad[i]=0
y=np.matrix(grad)
l=np.asarray(l)
x=[sat,gpa]
x.insert(0,l)
x_t=np.matrix(x)
x=np.transpose(x)
y=np.transpose(y)
xtx=np.matmul(x_t,x)
xty=np.matmul(x_t,y)
xtxinv=np.linalg.inv(xtx)
b=np.matmul(xtxinv,xty)
print("y=",b[0],"+",b[1],"*sat+",b[2],"*gpa")
y= [[-3.83919635]] + [[0.00323263]] *sat+ [[0.2395496]] *gpa
```

```
s=1000
g=2.9
y1=b[0]+b[1]*s+b[2]*g
print(y1)
if y1>0.5:
  print("incoming candidate is graduated")
else:
  print("incoming candidate is not graduated")
```

```
[[0.08812923]]
incoming candidate is not graduated
```

Discriminant analysis using fischer's formulae

Source Code:

```
import pandas as pd
import numpy as np
cur=[2.95,2.53,3.57,3.16,2.58,2.16,3.27]
dia=[6.63,7.79,5.65,5.47,4.46,6.22,3.52]
qua=["passed","passed","passed","passed","not passed","not passed","not passed"]
for i in range(len(qua)):
  if qua[i]=="passed":
    qua[i]=1
  else:
    qua[i]=2
print(qua)
x_t=[cur,dia]
x_t=np.asarray(x_t)
print(x_t)
x=np.transpose(x_t)
print(x)
meanx=[np.mean(x_t[0]),np.mean(x_t[1])]
print(meanx)
x1=[]
```

```
x2=[]
for i in range(len(qua)):
  if qua[i]==1:
    x1.append([cur[i],dia[i]])
  else:
    x2.append([cur[i],dia[i]])
x1=np.asarray(x1)
x2=np.asarray(x2)
x1_t=np.transpose(x1)
x2_t=np.transpose(x2)
print(x1_t)
meanx1=[np.mean(x1_t[0]),np.mean(x1_t[1])]
meanx2=[np.mean(x2_t[0]),np.mean(x2_t[1])]
print(meanx1)
print(meanx2)
```

```
[1, 1, 1, 1, 2, 2, 2]
[[2.95 2.53 3.57 3.16 2.58 2.16 3.27]
 [6.63 7.79 5.65 5.47 4.46 6.22 3.52]]
[[2.95 6.63]
 [2.53 7.79]
 [3.57 5.65]
 [3.16 5.47]
 [2.58 4.46]
 [2.16 6.22]
 [3.27 3.52]]
[2.888571428571429, 5.677142857142857]
[[2.95 6.63]
 [2.53 7.79]
 [3.57 5.65]
 [3.16 5.47]] [[2.58 4.46]
 [2.16 6.22]
 [3.27 3.52]]
[[2.95 2.53 3.57 3.16]
 [6.63 7.79 5.65 5.47]]
[3.0525, 6.385]
[2.67, 4.733333333333333]
```

```
xsubu=[]
for i in range(len(cur)):
  xsubu.append([x_t[0][i]-meanx[0],x_t[1][i]-meanx[1]])
xsubu=np.asarray(xsubu)
print(xsubu)
```

```
mat=np.matrix(xsubu)

print(mat)

num=np.matmul(np.transpose(mat),mat)

c=num/len(cur)

print(c)

cinv=np.linalg.inv(c)

print(cinv)
```

```
[[ 0.06142857  0.95285714]
 [-0.35857143  2.11285714]
 [ 0.68142857 -0.02714286]
 [ 0.27142857 -0.20714286]
 [-0.30857143 -1.21714286]
 [-0.72857143  0.54285714]
 [ 0.38142857 -2.15714286]]
[[ 0.06142857  0.95285714]
 [-0.35857143  2.11285714]
 [ 0.68142857 -0.02714286]
 [ 0.27142857 -0.20714286]
 [-0.30857143 -1.21714286]
 [-0.72857143  0.54285714]
 [ 0.38142857 -2.15714286]]
[[ 0.20598367 -0.23093265]
 [-0.23093265  1.69216327]]
[[5.73171449 0.78221769]
 [0.78221769 0.69771022]]
```

```
#f1

mmeanx1=np.matrix(meanx1)

mmeanx2=np.matrix(meanx2)

xk=np.matrix([2.8,5.46])

p1=np.matmul(np.matmul(mmeanx1,cinv),np.transpose(xk))

p2=np.matmul(np.matmul(mmeanx1,cinv),np.transpose(mmeanx1))

p2=p2/2

print(p1-p2)

import math

f1=p1-p2+math.log(len(x1)/len(x))

print(f1)
```

[[44.1628916]]

[[43.60327581]]

```
#f2

p3=np.matmul(np.matmul(mmeanx2,cinv),np.transpose(xk))
```

```
p4=np.matmul(np.matmul(mmeanx2,cinv),np.transpose(mmeanx2))
p4=p4/2
f2=p3-p4+math.log(len(x2)/len(x))
print(f2)
[[43.67295843]]
if f2>f1:
  print("new record goes to second group")
else:
  print("new record goes to first group")
new record goes to second group
```

Aim:

11.Write a python program to find Principle components for the given variables.

Source Code:

```
#determinant of matrix
import numpy as np
def cofactor(a,i,j):
 temp=np.concatenate((a[:i,:j],a[:i,j+1:]),axis=1)
 new=np.concatenate((a[i+1:,:j],a[i+1:,j+1:]),axis=1)
 return np.concatenate((temp,new))
def det_of(a):
 if(len(a)==2):
  val=(a[0][0]*a[1][1])-(a[0][1]*a[1][0])
 elif(len(a)==1):
  val=a
 else:
  val=0
  for col in range(len(a)):
   sign=(-1)**col
   sub_det=det_of(cofactor(a,0,col))
   val+=(sign*a[0][col]*sub_det)
 return val
a=np.array([[6,9,1],[2,3,8],[10,7,9]])
import numpy as np
def transpose(a):
 rows,cols=a.shape
 new=np.zeros((cols,rows),dtype=float)
 for i in range(rows):
  for j in range(cols):
   new[j][i]=a[i][j]
 return new
```

```
import numpy as np
def multiply(a,b):
 row1,col1=a.shape
 row2,col2=b.shape
 new=np.zeros((row1,col2),dtype=float)
 for i in range(row1):
  for j in range(col2):
   for k in range(col1):
    new[i][j]=new[i][j]+(a[i][k]*b[k][j])
 return new
import numpy as np
def adjoint(a):
 rows,cols=a.shape
 sign=[+1.0,-1.0]
 temp=0
 adj=np.empty([rows,cols],dtype=float)
 for i in range(rows):
  if(i%2==0):
    temp=0
  else:
   temp=1
  for j in range(cols):
   if(temp==0):
    mul=sign[0]
    temp=1
   else:
    mul=sign[1]
    temp=0
   co=cofactor(a,i,j)
   val=det_of(co)
   #print(val)
```

```
      val=val*mul
      #print(mul)
      adj[i][j]=val
    return adj
def inverse(a):
  adj=adjoint(a)
  tran=transpose(adj)
  det=det_of(a)
  inv=tran/det
  return inv
```

Program

```
import numpy as np
import pandas as pd
#n=int(input("enter the size: "))
#p=int(input("enter the number of variables: "))
#l=[]
#for i in range(p):
  #l1=list(map(float,input().split()))
  #l.append(l1)
l=[[7,4,6,8,8,7,5,9,7,8],[4,1,3,6,5,2,3,5,4,2],[3,8,5,1,7,9,3,8,5,2]]
x=np.transpose(l)
mean=np.mean(x,axis=0)
xminusu=x-mean
#print(xminusu)
#print(mean)
c=multiply(transpose(xminusu),xminusu)/len(x)
#print(c)
values,vectors=np.linalg.eig(c)
vectors=transpose(vectors)
values=list(values)
values_desc=[]
```

```
values_desc=sorted(values)
values_desc=values_desc[::-1]
#print(values)
#print(vectors)
sum=0
component=[]
tot_sum=0
req_range=int(input("enter the range u want?"))
final=[]
for i in values_desc:
  tot_sum+=i
compo=[]
for i in values_desc:
  sum+=i
  component=(sum/tot_sum)*100
  compo.append([component,i])
print(pd.DataFrame(compo))
i=1 #no of items we have to take
for j in compo:
  if j[0]<=req_range:
    i+=1
iter=1
final=[]
for j in compo:
  if(iter<=i):
    ind=values.index(j[1])
    vec=vectors[ind]
    col=np.dot(x,vec)
    final.append(col)
  iter+=1
print(np.array([final]))
```

```
enter the range u want?90
          0        1
0   65.149154  7.446548
1   94.095054  3.308516
2  100.000000  0.674935
[[[ 0.91007402  6.86567938  3.21471006 -1.64502171  4.35525466
    7.16081008  1.4356753   5.17598665  2.82667958  0.31511981]
  [ 8.3560561   5.64167652  7.54228952  9.83071184 10.80830182
    8.67275691  6.29709269 11.7804188   8.90221581  7.46023502]]]
```

```
#using user-defined multiplication
if iter<=i:
    ind=values.index(j[1])
    vectors=list(vectors)
    vec=vectors[ind]
    vec=np.array([vec])
    vec=transpose(vec)
    col=multiply(x,vec)
    final.append(col)
```

Aim:

12.Write a python program to group the given variables using Factor Analysis

Source Code:

```
import numpy as np
import pandas as pd
import math
print('enter the number')
n = int(input())
x=[]
for i in range(n):
  a=[float(x) for x in input().split()]
  x.append(a)
mu = []
for i in range(len(x[0])):
  sum = 0
  for j in range(len(x)):
    sum = sum+x[j][i]
  mu.append(sum/len(x))
mu=np.array(mu)
sigma=[]
for i in range(len(x[0])):
  sum = 0
  for j in range(len(x)):
    sum = sum+((x[j][i]-mu[i])**2)
  sigma.append(math.sqrt(sum/(len(x)-1)))
sigma =np.array(sigma)
#print(sigma)
x_mu=x-mu
#standardised values
std=x_mu/sigma
```

```
#print(std)
c = np.dot(std.T,std)/len(x)
eigen,eigen_vec= np.linalg.eig(c)
id = np.argsort(eigen)[::-1]
eigen = eigen[id]
eigen_vec = eigen_vec[:,id]
print(eigen)
print(eigen_vec)
z = []
sum = 0
sumT = eigen.sum()
for i in eigen:
 sum=sum+i
 z.append((sum/sumT)*100)
data = {'Principal_Components':['z1','z2','z3'],
    'Variance_Explained':eigen,
    'Cumulative_Proportion':z}
df = pd.DataFrame(data)
print('enter threshold')
k = float(input())
count = 1
for i in z:
 if(k>=i):
   count += 1
PC = df['Principal_Components']
for i in range(count):
 print('{}=[{}x1]+[{}x2]+[{}x3]'.format(PC[i],eigen_vec[0][i],eigen_vec[1][i],eigen_vec[2][i]))
print('Principal Component Table',df)
#factor analysis
f1=[]
f2=[]
```

```
f=[]
for j in range(len(eigen_vec[0])):
  for k in range(len(eigen_vec[j])):
    f.append((eigen_vec[k][j])*math.sqrt(eigen[j]))
#print(f)
f1=f[:3]
#print(f1)
f2=f[3:6]
#print(f2)
h=[f1,f2]
#print(h)
h_sq=[]
per_var=[]
for i in range(len(f1)):
  sum=0
  for j in range(len(h)):
      sum=sum+h[j][i]**2
  h_sq.append(sum)
for i in range(len(eigen)):
  per_var.append((eigen[i]/sumT)*100)
h_sq=np.array(h_sq)
tot_com=h_sq.sum()
#print(tot_com)
#print(per_var)
#factor loading table
print('variables Estimated_Factor_loadings Communalities')
fac_data={'F1':f1,'F2':f2,'H_Sq':h_sq}
df = pd.DataFrame(fac_data, index=['Finance','Marketing','Buspolicies'])
print(df)
print('var_Exp=    {}    {}    {}'.format(round(eigen[0],4),round(eigen[1],4),round(tot_com,2)))
print('per_var=    {}    {}'.format(round(per_var[0],4),round(per_var[1],4)))
```