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

**Source Code:**

#Import libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from statistics import mean

#Reading data

data=pd.read\_csv(‘csdata.csv’)

print(data)

x=data['x']

y=data['y']

def linear\_regression(x,y):

  sumx=0

  for i in range(len(x)):

    sumx=sumx+x[i]

  sumy=0

  for i in range(len(y)):

    sumy=sumy+y[i]

  meanx=sumx/len(x)

  meany=sumy/len(y)

  print("mean of x :",meanx)

  print("mean of y :",meany)

  n=sum((x-meanx)\*(y-meany))

  d=sum((x-meanx)\*\*2)

  b1=n/d

  b0=meany-b1\*meanx

  return b0,b1

b0,b1=linear\_regression(x,y)

print("B0 :",round(b0,5),", B1 :",round(b1,5))

print("Equation :")

print("y =",round(b0,5),"+ x \*",round(b1,5))

y\_hat=b0+x\*b1

print("y\_hat\n",y\_hat)

def plot1(x,y,y\_hat):

  plt.scatter(x,y)

  plt.xlabel("x values")

  plt.ylabel("y values")

  plt.title("Data")

  plt.show()

  plt.plot(x,y,'ro-')

  plt.plot(x,y\_hat,'bo-')

  plt.show()

#plot the graph

plot1(x,y,y\_hat)

def cost\_function(y,y\_hat):

  ybar=mean(y)

  sst=sum((y-ybar)\*\*2)

  ssr=sum((y\_hat-ybar)\*\*2)

  r2=ssr/sst

  return r2

r2=cost\_function(y,y\_hat)

print("Cost function is",round(r2,5))

if r2<0.9:

  print("Not best fit")

else:

  print("Best fit")

**Output 1**

x y

0 1 2.4

1 2 3.0

2 3 3.6

3 4 4.0

4 6 5.0

5 8 6.0

mean of x : 4.0

mean of y : 4.0

B0 : 1.97647 , B1 : 0.50588

Equation :

y = 1.97647 + x \* 0.50588

y\_hat :

0 2.482353

1 2.988235

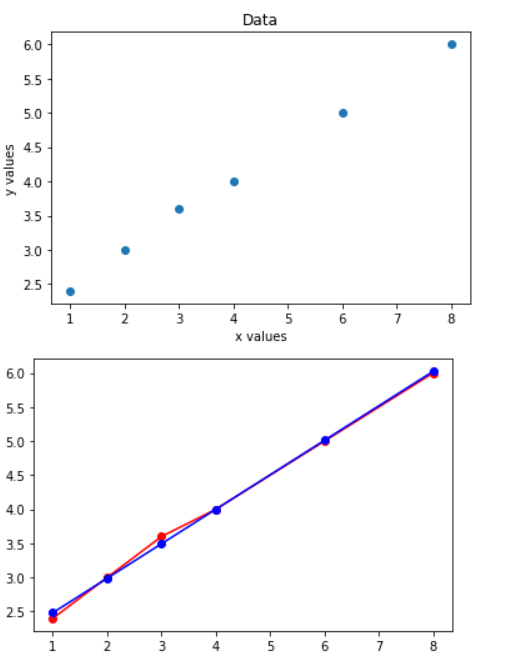
2 3.494118

3 4.000000

4 5.011765

5 6.023529

Name: x, dtype: float64



Cost function is 0.99784

Best fit

**Output 2**

x y

0 1 7.2

1 2 5.0

2 3 1.0

3 4 8.0

4 5 5.5

5 6 6.0

6 7 2.5

7 8 8.0

mean of x : 4.5

mean of y : 5.4

B0 : 5.1 , B1 : 0.06667

Equation :

y = 5.1 + x \* 0.06667

y\_hat :

0 5.166667

1 5.233333

2 5.300000

3 5.366667

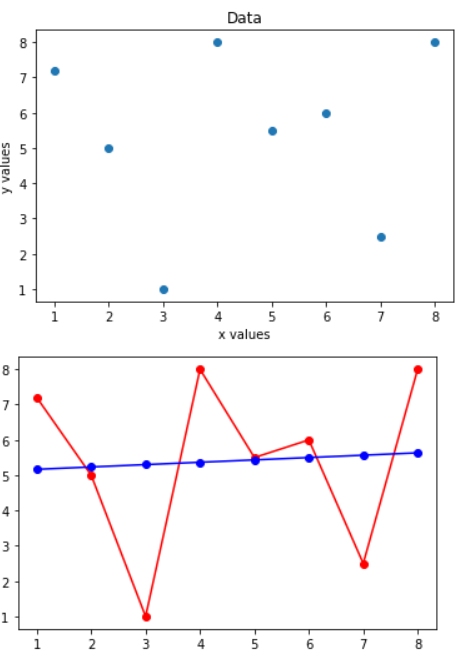
4 5.433333

5 5.500000

6 5.566667

7 5.633333

Name: x, dtype: float64

****

Cost function is 0.00414

Not best fit

**2) Aim: Write a python program to fit a second degree parabola of the form y=a+bx+cx^2 and draw the scatter plot.**

**Source Code:**

#importing libraries

import numpy as np

x=np.array([float(x) for x in input().split(" ")])

y=np.array([float(x) for x in input().split(" ")])

n=len(x)

sumx=np.sum(x)

sumy=np.sum(y)

sumxy=np.sum(x\*y)

sumx2=np.sum(x\*x)

sumx3=np.sum(x\*x\*x)

sumx4=np.sum(x\*x\*x\*x)

sumx2y=np.sum(x\*x\*y)

#calculating determinant

def getMinor(m,i,j):

    return [row[:j] + row[j+1:] for row in (m[:i]+m[i+1:])]

def getDeternminant(m):

    if len(m) == 2:

        return m[0][0]\*m[1][1]-m[0][1]\*m[1][0]

    determinant = 0

    for c in range(len(m)):

        determinant += ((-1)\*\*c)\*m[0][c]\*getDeternminant(getMinor(m,0,c))

    return determinant

#by using cramer's rule (without built-in)

p=getDeternminant([[n,sumx,sumx2],[sumx,sumx2,sumx3],[sumx2,sumx3,sumx4]])

q=getDeternminant([[sumy,sumxy,sumx2y],[sumx,sumx2,sumx3],[sumx2,sumx3,sumx4]])

r=getDeternminant([[n,sumx,sumx2],[sumy,sumxy,sumx2y],[sumx2,sumx3,sumx4]])

s=getDeternminant([[n,sumx,sumx2],[sumx,sumx2,sumx3],[sumy,sumxy,sumx2y]])

a=round(q/p,3)

b=round(r/p,3)

c=round(s/p,3)

print("The equation of parabola is y={}+{}x+{}x2".format(a,b,c))

import matplotlib.pyplot as plt

plt.scatter(x,y)

plt.xlabel("x values")

plt.ylabel("y values")

plt.title("Data")

plt.show()

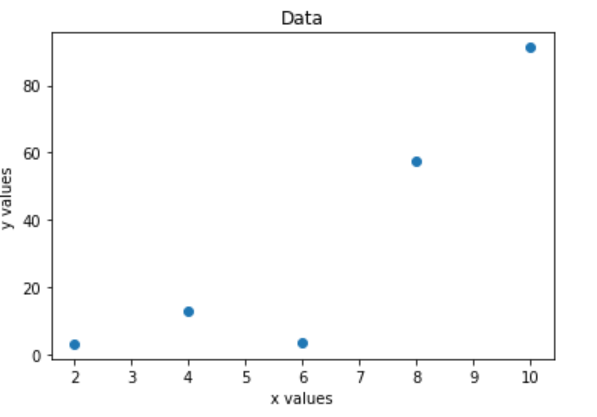
**Input 1:**

2 4 6 8 10

3.07 12.85 3.47 57.38 91.29

**Output 1:**

The equation of parabola is y=23.096+-12.855x+1.992x2



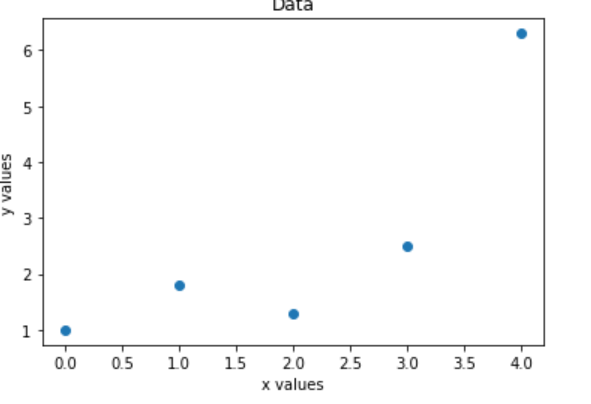
**Input 2:**

0 1 2 3 4

1 1.8 1.3 2.5 6.3

**Output 2:**

The equation of parabola is y=1.42+-1.07x+0.55x2



**3) Aim: Write a python program to find karl Pearson’s correlation coefficient between x and y variables.**

**Source Code:**

from math import sqrt

import numpy as np

print("Enter x :")

x=[int(x) for x in input().split()]

print("Enter y :")

y=[int(x) for x in input().split()]

x=np.array(x)

y=np.array(y)

xy=x\*y

x2=x\*\*2

y2=y\*\*2

n=len(x)

num=n\*sum(xy)-(sum(x)\*sum(y))

den=sqrt((n\*sum(x2)-sum(x)\*\*2)\*(n\*sum(y2)-sum(y)\*\*2))

r=num/den

print("Corelation coefficient is :",round(r,4))

**Output 1**

Enter x :

3 7 4 2 0 4 1 2

Enter y :

11 18 9 4 7 6 3 8

Corelation coefficient is : 0.7867

**Output 2**

Enter x :

65 66 67 67 68 69 70 72

Enter y :

67 68 65 68 72 72 69 71

Corelation coefficient is : 0.603

**4) Aim: Write a python program to find the Spearman’s correlation coefficient between x and y variables.**

**Source Code:**

import pandas as pd

s=input()

a=[float(i) for i in s.split(" ")]

s1=input()

b=[float(i) for i in s1.split(" ")]

n=len(a)

data=pd.DataFrame({'A':a,'B':b})

def rank(a):

  s=sorted(a)

  n=len(a)

  s=s[::-1]

  i=0

  d=[]

  count=[]

  while i<n:

    k=s.count(s[i])

    if k==1:

      d.append(i+1)

      i=i+1

    else:

      m=0

      for j in range(i+1,i+k+1):

        m=m+j

      m=m/k

      for j in range(k):

        d.append(m)

      i=i+k

      count.append(k)

  r=[]

  for i in range(n):

    j=s.index(a[i])

    r.append(d[j])

  return r,count

r\_x,c\_x=rank(a)

r\_y,c\_y=rank(b)

data['Rank of x']=r\_x

data['Rank of y']=r\_y

di=[]

di2=[]

for i in range(len(a)):

  k=r\_x[i]-r\_y[i]

  di.append(k)

  di2.append(k\*\*2)

data['di']=di

data['di2']=di2

print(data)

def correction\_factor(c):

  if len(c)!=0:

    m=c[0]

    cf=(m\*(m\*\*2-1))/12

    return cf

  else:

    return 0

cf\_x=correction\_factor(c\_x)

cf\_y=correction\_factor(c\_y)

sum\_di2=sum(di2)+cf\_x+cf\_y

print("Correction factor of a",cf\_x)

print("Correction factor of b",cf\_y)

print("di2 after correction factor is added",sum\_di2)

r=1-((6\*sum\_di2)/(n\*(n\*\*2-1)))

print("Rank Corelation coefficint : ",round(r,4))

**Input 1**

68 64 75 50 64 80 75 40 55 64

62 58 68 45 81 60 68 48 50 70

**Output 1:**

A B Rank of x Rank of y di di2

0 68.0 62.0 4.0 5.0 -1.0 1.0

1 64.0 58.0 6.0 7.0 -1.0 1.0

2 75.0 68.0 2.5 3.5 -1.0 1.0

3 50.0 45.0 9.0 10.0 -1.0 1.0

4 64.0 81.0 6.0 1.0 5.0 25.0

5 80.0 60.0 1.0 6.0 -5.0 25.0

6 75.0 68.0 2.5 3.5 -1.0 1.0

7 40.0 48.0 10.0 9.0 1.0 1.0

8 55.0 50.0 8.0 8.0 0.0 0.0

9 64.0 70.0 6.0 2.0 4.0 16.0

Correction factor of a 0.5

Correction factor of b 0.5

di2 after correction factor is added 73.0

Rank Corelation coefficint : 0.5576

**Input 2:**

115 109 112 87 98 120 98 100 98 118

75 73 85 70 76 82 65 73 68 80

**Output 2:**

A B Rank of x Rank of y di di2

0 115.0 75.0 3.0 5.0 -2.0 4.00

1 109.0 73.0 5.0 6.5 -1.5 2.25

2 112.0 85.0 4.0 1.0 3.0 9.00

3 87.0 70.0 10.0 8.0 2.0 4.00

4 98.0 76.0 8.0 4.0 4.0 16.00

5 120.0 82.0 1.0 2.0 -1.0 1.00

6 98.0 65.0 8.0 10.0 -2.0 4.00

7 100.0 73.0 6.0 6.5 -0.5 0.25

8 98.0 68.0 8.0 9.0 -1.0 1.00

9 118.0 80.0 2.0 3.0 -1.0 1.00

Correction factor of a 2.0

Correction factor of b 0.5

di2 after correction factor is added 45.0

Rank Corelation coefficint : 0.7273

**5) Aim: Write a python program to classify the data based on one way Anova.**

**Source Code:**

import numpy as np

import scipy.stats as stats

print("Enter treatment 1:")

a=[int(x) for x in input().split()]

print("Enter treatment 2:")

b=[int(x) for x in input().split()]

print("Enter treatment 3:")

c=[int(x) for x in input().split()]

#read level of significance

print("Level of significance :")

alpha=float(input())

fa=np.array(a)

fb=np.array(b)

fc=np.array(c)

print("Treatment 1 :",fa)

print("Treatment 2 :",fb)

print("Treatment 3 :",fc)

#calculate rss,cf,sst,sstr,sse

N=np.size(fa)+np.size(fb)+np.size(fc)

rss=np.sum(fa\*\*2)+np.sum(fb\*\*2)+np.sum(fc\*\*2)

cf=(np.sum(fa)+np.sum(fb)+np.sum(fc))\*\*2/(N)

sst=rss-cf

sstr=(np.sum(fa)\*\*2/np.size(fa)+np.sum(fb)\*\*2/np.size(fb)+np.sum(fc)\*\*2/np.size(fc))-cf

sse=sst-sstr

print("rss=",rss)

print("cf=",cf)

print("sst=",sst)

print("sstr=",sstr)

print("sse=",sse)

#degree of freedom

k=3

d1=k-1

d2=N-k

print("Degree of freedom of treatments =",d1)

print("Degree of freedom of error =",d2)

#calculate f

msstr=sstr/d1

msse=sse/d2

F=msstr/msse

if F<1:

  F=msse/msstr

print("Calculated value :",F)

#table value

tablevalue=stats.f.ppf(1-alpha,d1,d2)

print("Table value :",round(tablevalue,4))

 #testing

if tablevalue>F:

  print("H0 is accepted.")

else:

  print("H0 is rejected.")

**Input 1:**

Enter treatment 1:

13 10 8 11 8

Enter treatment 2:

13 11 14 14

Enter treatment 3:

4 1 3 4 2 4

Level of significance :

0.05

**Output 1:**

Treatment 1 : [13 10 8 11 8]

Treatment 2 : [13 11 14 14]

Treatment 3 : [4 1 3 4 2 4]

rss= 1262

cf= 960.0

sst= 302.0

sstr= 270.0

sse= 32.0

Degree of freedom of treatments = 2

Degree of freedom of error = 12

Calculated value : 50.625

Table value : 3.8853

H0 is rejected.

**Input 2:**

Enter treatment 1:

90 82 79 98 83 91

Enter treatment 2:

105 89 93 104 89 95 86

Enter treatment 3:

83 89 80 94

Level of significance :

0.05

**Output 2:**

Treatment 1 : [90 82 79 98 83 91]

Treatment 2 : [105 89 93 104 89 95 86]

Treatment 3 : [83 89 80 94]

rss= 138638

cf= 137700.0

sst= 938.0

sstr= 234.4523809523671

sse= 703.5476190476329

Degree of freedom of treatments = 2

Degree of freedom of error = 14

Calculated value : 2.3327016142676427

Table value : 3.7389

H0 is accepted.

**6) Aim: Write a python program to classify the data based on two way Anova**

**Source Code:**

import scipy.stats as stats

import pandas as pd

def input\_data(k):

l=[]

  for i in range(k):

    print("Enter treatment ",i+1)

    s=[int(x) for x in input().split()]

    l.append(s)

  return l

print("Enter number of treatments :")

k=int(input())

print("Enter number of blocks :")

h=int(input( ))

l=input\_data(k)

def dataframe(l):

  df=pd.DataFrame(l)

  col=[]

  for i in range(h):

    col.append("B"+str(i+1))

  df.columns=col

  index=[]

  for i in range(k):

    index.append("T"+str(i+1))

  df.index=index

  print("Given data :")

  print(df)

dataframe(l)

def calculations(l):

  G=0

  flag\_ftr=0

  flag\_fb=0

  Ti2=0

  for i in range(k):

    G=G+sum(l[i])

    Ti2=Ti2+sum(l[i])\*\*2

  bj2=0

  rss=0

  for j in range(h):

    bj=0

    for i in range(k):

      bj=bj+l[i][j]

      rss=rss+l[i][j]\*\*2

    bj2=bj2+bj\*\*2

  cf=(G\*\*2)/(k\*h)

  st2=rss-cf

  str2=Ti2\*(1/h)-cf

  sb2=bj2\*(1/k)-cf

  se2=st2-str2-sb2

  print("Row sum of squares =",rss)

  print("Correction factor =",cf)

  print("Sum of squares due to total =",st2)

  print("Sum of squares due to treatments =",str2)

  print("Sum of squares due to blocks =",sb2)

  print("Sum of squares due to error =",se2)

  mstr=str2/(k-1)

  msb=sb2/(h-1)

  mse=se2/((k-1)\*(h-1))

  ftr=mstr/mse

  fb=msb/mse

  if ftr<1:

    ftr=mse/mstr

    flag\_ftr=1

  if fb<1:

    fb=mse/msb

    flag\_fb=1

  return ftr,fb,flag\_ftr,flag\_fb

ftr,fb,flag\_ftr,flag\_fb=calculations(l)

print("Caluclated values")

print("Treatments :",round(ftr,4))

print("Blocks :",round(fb,4))

if flag\_ftr==1:

  ft\_tr=stats.f.ppf(0.95,(k-1)\*(h-1),(k-1))

else:

  ft\_tr=stats.f.ppf(0.95,(k-1),(k-1)\*(h-1))

if flag\_fb==1:

  ft\_b=stats.f.ppf(0.95,(k-1)\*(h-1),(h-1))

else:

  ft\_b=stats.f.ppf(0.95,(h-1),(k-1)\*(h-1))

print("Table values")

print("Treatments :",round(ft\_tr,4))

print("Blocks :",round(ft\_b,4))

if ftr>ft\_tr:

  print("H0(tr) is rejected.")

else:

  print("H0(tr) is accepeted.")

if fb>ft\_b:

  print("H0(b) is rejected.")

else:

  print("H0(b) is accepeted.")

**Input 1:**

Enter number of treatments :

3

Enter number of blocks :

4

13 7 9 3

6 6 3 1

11 5 15 5

**Output 1:**

Given data :

B1 B2 B3 B4

T1 13 7 9 3

T2 6 6 3 1

T3 11 5 15 5

Row sum of squares = 786

Correction factor = 588.0

Sum of squares due to total = 198.0

Sum of squares due to treatments = 56.0

Sum of squares due to blocks = 90.0

Sum of squares due to error = 52.0

Caluclated values

Treatments : 3.2308

Blocks : 3.4615

Table values

Treatments : 5.1433

Blocks : 4.7571

H0(tr) is accepeted.

H0(b) is accepeted.

**Input 2:**

Enter number of treatments :

4

Enter number of blocks :

5

Enter treatment 1

75 73 59 69 84

Enter treatment 2

83 72 56 70 92

Enter treatment 3

86 61 53 72 88

Enter treatment 4

73 67 62 79 95

**Output 2:**

Given data :

B1 B2 B3 B4 B5

T1 75 73 59 69 84

T2 83 72 56 70 92

T3 86 61 53 72 88

T4 73 67 62 79 95

Row sum of squares = 110607

Correction factor = 107898.05

Sum of squares due to total = 2708.949999999997

Sum of squares due to treatments = 42.94999999999709

Sum of squares due to blocks = 2326.699999999997

Sum of squares due to error = 339.3000000000029

Caluclated values

Treatments : 1.975

Blocks : 20.5721

Table values

Treatments : 8.7446

Blocks : 3.2592

H0(tr) is accepeted.

H0(b) is rejected.

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

**Source Code:**

#transpose

def transpose(arr):

t=[]

for i in range(len(arr[0])):

T=[]

for j in range(len(arr)):

T.append(arr[j][i])

t.append(T)

return t

#multipliction

def mul(a,b):

c=[]

n=len(a)

m=len(a[0])

q=len(b[0])

for i in range(n):

C=[]

for j in range(q):

multi=0

for k in range(m):

multi=multi+a[i][k]\*b[k][j]

C.append(round(multi,4))

c.append(C)

return c

#inverse

def inverse(a):

n=len(a)

m=len(a[0])

cofactor=[]

det=0

if n==2 and m==2:

cofactor.append([a[1][1],-1\*a[0][1]])

cofactor.append(-1\*[a[1][0],a[0][0]])

det=cofactor[0][0]\*cofactor[1][1]-cofactor[0][1]\*cofactor[1][0]

else:

for i in range(n):

co=[]

for j in range(m):

c=[]

for k in range(n):

for o in range(m):

if i!=k and j!=o:

c.append(a[k][o])

if (i+j)%2!=0:

q=-1\*(c[0]\*c[3]-c[1]\*c[2])

else:

q=(c[0]\*c[3]-c[1]\*c[2])

co.append(q)

if i==0:

det=det+a[i][j]\*q

cofactor.append(co)

inv=transpose(cofactor)

ine=inv/det

return ine

import numpy as np

import pandas as pd

import scipy.stats as s

df=pd.read\_csv('mutliple.csv')

print(df)

#beta\_hat=(x'x)-1(x'y)

df['x0']=[1]\*len(df)

x=df[['x0','x1','x2']].to\_numpy()

y=df[['y']].to\_numpy()

#calculate x'x and x'y and (x'x)(x'y)

x1=transpose(x)

x1x=mul(x1,x)

inv=inverse(x1x)

x1y=mul(x1,y)

beta\_hat=mul(inv,x1y)

print("y =",beta\_hat[0][0],"+ x1",beta\_hat[1][0],"+ x2",beta\_hat[2][0])

#test of goodness of fit using coefficient of determination

y\_hat=[]

error=[]

for i in range(len(x)):

s=beta\_hat[0][0]+x[i][1]\*beta\_hat[1][0]+x[i][2]\*beta\_hat[2][0]

y\_hat.append(round(s,4))

error.append(round(y[i][0]-s,4))

d=pd.DataFrame({'y\_hat':y\_hat,'error':error})

print(d)

#calculations

sse=sum(np.array(error)\*\*2)

y\_bar=sum(y)/len(y)

sst=sum((y-y\_bar)\*\*2)

ssr=sst-sse

R2=ssr/sst

print(R2)

if R2<0.9:

print("The Regression model is not good fit")

else:

print("The Regression model is good fit")

#degree of freedom

n1=len(x[0])-1

n2=len(x)-len(x[0])

#to test goodness of fit using anova

msr=ssr/n1

mse=sse/n2

f=msr/mse

f=f[0]

if f<1:

f=mse/msr

print("Calculated value",round(f,4))

#table value

f\_tab=s.f.ppf(0.95,n1,n2)

print("Table value",round(f\_tab,4))

if f<f\_tab:

print("H0 is Accepted")

print("Hence we conclude that there is no regression parameter that influence in the model.")

else:

print("H0 is Rejected")

print("Hence we conclude that there is atleast one regression parameter that influence in the model.")

#test of individual variables

cij=[inv[0][0],inv[1][1],inv[2][2]]

t\_cal=[]

print("Calculated value :")

for i in range(len(cij)):

se=sqrt(mse\*cij[i])

t=beta\_hat[i][0]

t\_cal.append(round(t/se,4))

print("beta",i,"=",t\_cal[i])

#table value

t\_tab=stats.t.ppf(1-0.05/2,n2)

print("Table value",round(t\_tab,4))

weak\_variable=-1

for i in range(len(t\_cal)):

if t\_tab>t\_cal[i]:

print("H0 is accepted.\nHence the parameter beta",i,"is not influencing the model")

weak\_variable=i

else:

print("H0 is rejected.\nHence the parameter beta",i,"is influencing the model")

print("Therefore,The weak variable is beta",weak\_variable)

**Output 1:**

x1 x2 y

0 -5 5 11

1 -4 4 11

2 -1 1 8

3 2 -3 2

4 2 -2 5

5 3 -2 5

6 3 -3 4

y = 6.5714 + 1.0 x1 + 2.0 x2

y\_hat error

0 11.5714 -0.5714

1 10.5714 0.4286

2 7.5714 0.4286

3 2.5714 -0.5714

4 4.5714 0.4286

5 5.5714 -0.5714

6 3.5714 0.4286

#R2

R2= 0.97674419

The Regression model is good fit

#anova

Calculated value 84.0

Table value 6.9443

H0 is Rejected

Hence we conclude that there is atleast one regression parameter that influence in the model.

#testing weak variables

Calculated value :

beta 0 = 26.558

beta 1 = 2.1523

beta 2 = 4.3046

Table value 2.7764

H0 is rejected.

Hence the parameter beta 0 is influencing the model

H0 is accepted.

Hence the parameter beta 1 is not influencing the model

H0 is rejected.

Hence the parameter beta 2 is influencing the model

Therefore, The weak variable is beta 1

**Output 2**:

month y x1 x2

0 1 100 9 62

1 2 110 8 58

2 3 105 7 64

3 4 94 14 60

4 5 95 12 63

5 6 99 10 57

6 7 104 7 55

7 8 108 4 56

8 9 105 6 59

9 10 98 5 61

10 11 105 7 57

11 12 110 6 60

y = 133.4605 + -1.2485 x1 + -0.351 x2

y\_hat error

0 100.4620 -0.4620

1 103.1145 6.8855

2 102.2570 2.7430

3 94.9215 -0.9215

4 96.3655 -1.3655

5 100.9685 -1.9685

6 105.4160 -1.4160

7 108.8105 -0.8105

8 105.2605 -0.2605

9 105.8070 -7.8070

10 104.7140 0.2860

11 104.9095 5.0905

[0.5415279]

The Regression model is not good fit

Calculated value 5.3152

Table value 4.2565

H0 is Rejected

Hence we conclude that there is atleast one regression parameter that influence in the model.

Calculated value :

beta 0 = 5.0882

beta 1 = -2.8079

beta 2 = -0.7711

Table value 2.2622

H0 is rejected.

Hence the parameter beta 0 is influencing the model

H0 is accepted.

Hence the parameter beta 1 is not influencing the model

H0 is accepted.

Hence the parameter beta 2 is not influencing the model

Therefore,The weak variable is beta 2

**8) Aim: Write a python program to fit a multivariate regression model for the given data.**

**Source Code:**

import numpy as np

import pandas as pd

import scipy.stats as stats

df=pd.read\_csv('multivariate.csv')

print(df)

#beta ground=(x'x)-1\*(x'y)

df['x0']=[1]\*len(df)

x=df[['x0','x1','x2','x3']].to\_numpy()

y=df[['y1','y2']].to\_numpy()

x1x=mul(transpose(x),x)

x1y=mul(transpose(x),y)

inv=np.linalg.inv(x1x)

beta=np.array(mul(inv,x1y))

print("y1 =",beta[0][0],"+",beta[1][0],"x1 +",beta[2][0],"x2 +",beta[3][0],"x3")

print("y2 =",beta[0][1],"+",beta[1][1],"x1 +",beta[2][1],"x2 +",beta[3][1],"x3")

#test of goodness of fit using coefficient of determination

y\_hat=x@beta

error=y-y\_hat

sse=np.sum(error\*\*2,axis=0)

mean=np.sum(y,axis=0)/len(y)

sst=np.sum((y-mean)\*\*2,axis=0)

ssr=sst-sse

R2=ssr/sst

def test(r2):

if r2<0.9:

print("The Regression model is not good fit")

else:

print("The Regression model is good fit")

print("For y1 R2 is:",round(R2[0],4))

test(R2[0])

print("For y2 R2 is:",round(R2[1],4))

test(R2[1])

#to test goodness of fit using anova

def cal\_value(f\_cal):

if f\_cal<1:

f\_cal=mse/msr

print("Calculated value",round(f\_cal,4))

#degree of freedom

n1=len(x[0])-1

n2=len(x)-len(x[0])

msr=ssr/n1

mse=sse/n2

f=msr/mse

cal\_value(f[0])

cal\_value(f[1])

#table value

f\_tab=stats.f.ppf(0.95,n1,n2)

print("Table value",round(f\_tab,4))

def test(f\_cal):

if f\_cal<f\_tab:

print("H0 is Accepted")

print("Hence we conclude that there is no regression parameter that influence in the model.")

else:

print("H0 is Rejected")

print("Hence we conclude that there is atleast one regression parameter that influence in the model.")

print("For y1 :")

test(f[0])

print("For y2 :")

test(f[1])

#testing individual parameters

t=[]

for i in range(len(inv)):

se1=sqrt(mse[0]\*inv[i][i])

se2=sqrt(mse[1]\*inv[i][i])

t1=round(beta[i][0]/se1,4)

t2=round(beta[i][1]/se2,4)

t.append([t1,t2])

for i in range(len(t[0])):

print("t calculated value for y"+str(i+1))

for j in range(len(t)):

print("beta"+str(j)+":",t[j][i])

t\_tab=round(stats.t.ppf(1-0.05/2,n2),4)

print("Table Value :",t\_tab)

#testing the weak parameters

for i in range(len(t[0])):

print("For y"+str(i+1)+" :")

weak\_variable=[]

for j in range(len(t)):

if t\_tab>t[j][i]:

print("H0 is accepted.\nHence the parameter beta",i,"is not influencing the model")

weak\_variable.append(j)

else:

print("H0 is rejected.\nHence the parameter beta",i,"is influencing the model")

print("\nTherefore,The weak variable is beta",weak\_variable)

**Output 1:**

#output

month y1 y2 x1 x2 x3

0 1 10 100 9 62 1.0

1 2 12 110 8 58 1.3

2 3 11 105 7 64 1.2

3 4 9 94 14 60 0.8

4 5 9 95 12 63 0.8

5 6 10 99 10 57 0.9

6 7 11 104 7 55 1.0

7 8 12 108 4 56 1.2

8 9 11 105 6 59 1.1

9 10 10 98 5 61 1.0

10 11 11 103 7 57 1.2

11 12 12 110 6 60 1.2

y1 = 10.897 + -0.0449 x1 + -0.0877 x2 + 5.0355 x3

y2 = 91.0972 + -0.064 x1 + -0.2944 x2 + 27.8353 x3

For y1 R2 is: 0.9238

The Regression model is good fit

For y2 R2 is: 0.8655

The Regression model is not good fit

Calculated value 32.3272

Calculated value 17.1613

Table value 4.0662

For y1 :

H0 is Rejected

Hence we conclude that there is atleast one regression parameter that influence in the model.

For y2 :

H0 is Rejected

Hence we conclude that there is atleast one regression parameter that influence in the model.

t calculated value for y1

beta0: 4.2373

beta1: -0.8276

beta2: -2.2751

beta3: 5.4618

t calculated value for y2

beta0: 5.2648

beta1: -0.1753

beta2: -1.1351

beta3: 4.4872

Table Value : 2.306

For y1 :

H0 is rejected.

Hence the parameter beta 0 is influencing the model

H0 is accepted.

Hence the parameter beta 0 is not influencing the model

H0 is accepted.

Hence the parameter beta 0 is not influencing the model

H0 is rejected.

Hence the parameter beta 0 is influencing the model

Therefore,The weak variable is beta [1, 2]

For y2 :

H0 is rejected.

Hence the parameter beta 1 is influencing the model

H0 is accepted.

Hence the parameter beta 1 is not influencing the model

H0 is accepted.

Hence the parameter beta 1 is not influencing the model

H0 is rejected.

Hence the parameter beta 1 is influencing the model

Therefore, The weak variable is beta [1, 2]

**Output 2:**

y1 y2 x1 x2 x3

0 9 150 3 62 1.5

1 2 98 8 58 0.7

2 5 75 3 64 3.6

3 2 24 14 60 1.8

4 4 95 15 63 1.8

5 13 39 18 57 6.9

6 10 14 23 55 1.1

7 16 88 4 56 1.0

8 6 15 6 59 1.0

9 2 48 12 61 1.6

10 1 73 9 57 1.3

11 12 91 10 60 1.2

y1 = 59.8734 + -0.2111 x1 + -0.8916 x2 + 1.0523 x3

y2 = -84.0353 + -3.0545 x1 + 3.1483 x2 + -1.7594 x3

For y1 R2 is: 0.2755

The Regression model is not good fit

For y2 R2 is: 0.3696

The Regression model is not good fit

Calculated value 1.014

Calculated value 1.5632

Table value 4.0662

For y1 :

H0 is Accepted

Hence we conclude that there is no regression parameter that influence in the model.

For y2 :

H0 is Accepted

Hence we conclude that there is no regression parameter that influence in the model.

t calculated value for y1

beta0: 1.6625

beta1: -0.7635

beta2: -1.5111

beta3: 1.1341

t calculated value for y2

beta0: -0.3096

beta1: -1.4659

beta2: 0.708

beta3: -0.2516

Table Value : 2.306

For y1 :

H0 is accepted.

Hence the parameter beta 0 is not influencing the model

H0 is accepted.

Hence the parameter beta 0 is not influencing the model

H0 is accepted.

Hence the parameter beta 0 is not influencing the model

H0 is accepted.

Hence the parameter beta 0 is not influencing the model

Therefore,The weak variable is beta [0, 1, 2, 3]

For y2 :

H0 is accepted.

Hence the parameter beta 1 is not influencing the model

H0 is accepted.

Hence the parameter beta 1 is not influencing the model

H0 is accepted.

Hence the parameter beta 1 is not influencing the model

H0 is accepted.

Hence the parameter beta 1 is not influencing the model

Therefore, The weak variable is beta [0, 1, 2, 3]

**9) Aim: Write a python program to classify the treatments based on MANOVA Test.**

**Source Code:**

import numpy as np

import scipy.stats as stats

from math import sqrt

m=int(input('Enter number of treatments : '))

t=[]

for i in range(m):

print("Enter Treatment",(i+1))

y1=[int(x) for x in input().split()]

y2=[int(x) for x in input().split()]

t.append([y1,y2])

for i in range(m):

print("Treatment",i+1)

print(t[i][0])

print(t[i][1])

t\_mean=[]

total=[0,0]

t\_size=0

for i in range(m):

y1=np.array(t[i][0])

y2=np.array(t[i][1])

t\_mean.append([sum(y1)/len(y1),sum(y2)/len(y2)])

total[0]+=sum(y1)

total[1]+=sum(y2)

t\_size+=len(y1)

total[0]=total[0]/t\_size

total[1]=total[1]/t\_size

print("Yi mean")

for i in range(m):

print(t\_mean[i])

print("Y mean")

print(total)

def calculations(t,t\_mean,total,y):

sse=0

sst=0

for i in range(m):

for j in range(len(t[i][0])):

if y==-1:

sse+=t[i][0][j]\*t[i][1][j]-t\_mean[i][0]\*t\_mean[i][1]

sst+=t[i][0][j]\*t[i][1][j]-total[0]\*total[1]

else:

sse+=(t[i][y][j]-t\_mean[i][y])\*\*2

sst+=(t[i][y][j]-total[y])\*\*2

return sse,sst

sse\_y1,sst\_y1=calculations(t,t\_mean,total,0)

ssr\_y1=sst\_y1-sse\_y1

sse\_y2,sst\_y2=calculations(t,t\_mean,total,1)

ssr\_y2=sst\_y2-sse\_y2

sse\_y,sst\_y=calculations(t,t\_mean,total,-1)

ssr\_y=sst\_y-sse\_y

print("For y1 :")

print("sse =",sse\_y1,end=" , ")

print("sst =",sst\_y1,end=" , ")

print("ssr =",ssr\_y1)

print("For y2 :")

print("sse =",sse\_y2,end=" , ")

print("sst =",sst\_y2,end=" , ")

print("ssr =",ssr\_y2)

print("Cross product values of y1 and y2 :")

print("sse =",sse\_y,end=" , ")

print("sst =",sst\_y,end=" , ")

print("ssr =",ssr\_y)

#sum of squares

B=np.array([ssr\_y1,ssr\_y,ssr\_y,ssr\_y2]).reshape(2,2)

W=np.array([sse\_y1,sse\_y,sse\_y,sse\_y2]).reshape(2,2)

T=np.array([sst\_y1,sst\_y,sst\_y,sst\_y2]).reshape(2,2)

print("Regression :\n",B)

print("Error :\n",W)

print("Total :\n",T)

#Degree of Freedom

d1=m-1

n=0

for i in range(m):

n=n+len(t[i][0])

d2=n-m

print("Degree of Freedom :",d1,",",d2)

def det(A):

return A[0][0]\*A[1][1]-A[0][1]\*A[1][0]

wilks=det(W)/det(T)

print("Wilk's Value :",round(wilks,4))

f=((n-m-1)/(m-1))\*(1-sqrt(wilks))/sqrt(wilks)

print("Calculate value :",round(f,4))

tab=stats.f.ppf(0.95,2\*(m-1),2\*(n-m-1))

print("Table Value :",round(tab,4))

if f>tab:

print("H0 is Rejected.Hence we conclude that there is no homogenity among regression model")

else:

print("H0 is Accepted.Hence we conclude that there is homogenity among regression model")

**Input 1:**

Enter number of treatments : 3

Enter Treatment 1

2 3 5 2

3 4 4 5

Enter Treatment 2

4 5 6

8 6 7

Enter Treatment 3

7 8 10 9 7

6 7 8 5 6

**Output 1:**

Treatment 1

[2, 3, 5, 2]

[3, 4, 4, 5]

Treatment 2

[4, 5, 6]

[8, 6, 7]

Treatment 3

[7, 8, 10, 9, 7]

[6, 7, 8, 5, 6]

Yi mean

[3.0, 4.0]

[5.0, 7.0]

[8.2, 6.4]

Y mean

[5.666666666666667, 5.75]

For y1 :

sse = 14.799999999999997 , sst = 76.66666666666667 , ssr = 61.866666666666674

For y2 :

sse = 9.2 , sst = 28.25 , ssr = 19.05

Cross product values of y1 and y2 :

sse = 1.6000000000000156 , sst = 25.999999999999943 , ssr = 24.399999999999928

Regression :

[[61.86666667 24.4 ]

[24.4 19.05 ]]

Error :

[[14.8 1.6]

[ 1.6 9.2]]

Total :

[[76.66666667 26. ]

[26. 28.25 ]]

Degree of Freedom : 2 , 9

Wilk's Value : 0.0897

Calculate value : 9.3575

Table Value : 3.0069

H0 is Rejected.Hence we conclude that there is no homogenity among regression model

**Input 2:**

Enter number of treatments : 3

Enter Treatment 1

9 6 9

3 2 7

Enter Treatment 2

0 2

4 0

Enter Treatment 3

3 1 2

8 9 7

**Output 2:**

Treatment 1

[9, 6, 9]

[3, 2, 7]

Treatment 2

[0, 2]

[4, 0]

Treatment 3

[3, 1, 2]

[8, 9, 7]

Yi mean

[8.0, 4.0]

[1.0, 2.0]

[2.0, 8.0]

Y mean

[4.0, 5.0]

For y1 :

sse = 10.0 , sst = 88.0 , ssr = 78.0

For y2 :

sse = 24.0 , sst = 72.0 , ssr = 48.0

Cross product values of y1 and y2 :

sse = 1.0 , sst = -11.0 , ssr = -12.0

Regression :

[[ 78. -12.]

[-12. 48.]]

Error :

[[10. 1.]

[ 1. 24.]]

Total :

[[ 88. -11.]

[-11. 72.]]

Degree of Freedom : 2 , 5

Wilk's Value : 0.0385

Calculate value : 8.1989

Table Value : 3.8379

H0 is Rejected.Hence we conclude that there is no homogenity among regression model

**10) Aim: Write a python program to classify the given observation using Linear Discriminant Analysis.**

**Source Code:**

import pandas as pd

import numpy as np

sat=int(input('Enter incoming student SAT : '))

gpa=float(input('Enter incoming student GPA : '))

df=pd.read\_csv('discriminant.csv')

print(df)

df['x0']=[1]\*len(df)

x=df[['x0','x1','x2']]

x=np.array(x,dtype=float)

y=df[['y']].replace('yes',1)

y=df[['y']].replace('no',0)

y=np.array(y)

#beta hat

x1x=mul(transpose(x),x)

inv=np.linalg.inv(x1x)

x1y=mul(transpose(x),y)

beta\_hat=mul(inv,x1y)

print("y = ",beta\_hat[0][0],"+",beta\_hat[1][0],"x1 +",beta\_hat[2][0],"x2")

new\_y=beta\_hat[0][0]+beta\_hat[1][0]\*sat+beta\_hat[2][0]\*gpa

print("new\_y =",new\_y)

if round(new\_y)==0:

print("The value is nearer to zero.Therefore, the candidate will not graduated.")

else:

print("The value is nearer to one.Therefore, the candidate will graduated.")

**Input 1:**

Enter incoming student SAT : 1000

Enter incoming student GPA : 2.9

**Output 1:**

x1 x2 y

0 1300 2.7 yes

1 1260 3.7 yes

2 1220 2.9 yes

3 1180 2.5 yes

4 1060 3.9 yes

5 1140 2.1 no

6 1100 3.5 no

7 1020 3.3 no

8 980 2.3 no

9 940 3.1 no

y = -3.8392 + 0.0032 x1 + 0.2395 x2

new\_y = 0.05535000000000023

The value is nearer to zero.Therefore, the candidate will not graduated.

**Output 2:**

Enter incoming student SAT : 1500

Enter incoming student GPA : 1.3

x1 x2 y

0 1150 2.7 yes

1 1060 3.7 yes

2 1220 2.9 yes

3 1980 2.5 yes

4 1980 3.9 no

5 1840 2.1 no

y = 2.264495 + -0.00080765 x1 + -0.11979125 x2

new\_y = 0.89727898

The value is nearer to one.Therefore, the candidate will graduated.

**#Fishers’ Linear Discrminant**

import pandas as pd

import numpy as np

from math import log

print("Enter new matrix")

l=[float(x) for x in input().split()]

df=pd.read\_csv('fishers.csv')

print(df)

x\_1=[]

x\_2=[]

for i in range(len(df)):

if df['y'][i]==1:

x\_1.extend([df['x1'][i],df['x2'][i]])

else:

x\_2.extend([df['x1'][i],df['x2'][i]])

x=df[['x1','x2']].to\_numpy()

x\_1=np.array(x\_1).reshape(len(x\_1)//2,2)

x\_2=np.array(x\_2).reshape(len(x\_2)//2,2)

print("x :\n",x)

print("\nx1 :\n",x\_1)

print("\nx2 :\n",x\_2)

mean=np.mean(x,axis=0)

mean\_1=np.mean(x\_1,axis=0)

mean\_2=np.mean(x\_2,axis=0)

print("mean for x :",mean)

print("mean for x\_1 :",mean\_1)

print("mean for x\_2 :",mean\_2)

xm=x-mean

c=mul(transpose(xm),xm)

c=np.array(c)/len(x)

c\_inv=np.linalg.inv(c)

print("Pooled covariance matrix :\n",c)

def fisherEquation(mean,c\_inv,x,p):

m\_c=mul([mean],c\_inv)

f=mul(m\_c,x)[0][0]-0.5\*mul(m\_c,transpose([mean]))[0][0]+log(p)

return f

f1=fisherEquation(mean\_1,c\_inv,transpose([l]),len(x\_1)/len(x))

f2=fisherEquation(mean\_2,c\_inv,transpose([l]),len(x\_2)/len(x))

print("f1 =",f1)

print("f2 =",f2)

if f1>f2:

print("The new observation",l,"is classified into group 1")

else:

print("The new observation",l,"is classified into group 2")

**Input 1:**

Enter new matrix

5.1 3.2

**Output 1:**

x1 x2 y

0 1 2 1

1 2 3 1

2 3 3 1

3 4 5 1

4 5 5 1

5 4 2 0

6 5 0 0

7 5 2 0

8 3 2 0

9 5 3 0

10 6 3 0

x :

[[1 2]

[2 3]

[3 3]

[4 5]

[5 5]

[4 2]

[5 0]

[5 2]

[3 2]

[5 3]

[6 3]]

x1 :

[[1 2]

[2 3]

[3 3]

[4 5]

[5 5]]

x2 :

[[4 2]

[5 0]

[5 2]

[3 2]

[5 3]

[6 3]]

mean for x : [3.90909091 2.72727273]

mean for x\_1 : [3. 3.6]

mean for x\_2 : [4.66666667 2. ]

Pooled covariance matrix :

[[2.08264545 0.15702727]

[0.15702727 1.83470909]]

f1 = 6.48594263963573

f2 = 7.393764196429685

The new observation [5.1, 3.2] is classified into group 2

**Input 2:**

Enter new matrix

5 6

**Output 2:**

x1 x2 y

0 4 2 1

1 2 4 1

2 2 3 1

3 3 6 1

4 4 4 1

5 9 10 0

6 6 8 0

7 9 5 0

8 8 7 0

9 10 8 0

x :

[[4 2]

[2 4]

[2 3]

[3 6]

[4 4]

[9 10]

[6 8]

[9 5]

[8 7]

[10 8]]

x1 :

[[4 2]

[2 4]

[2 3]

[3 6]

[4 4]]

x2 :

[[9 10]

[6 8]

[9 5]

[8 7]

[10 8]]

mean for x : [5.7 5.7]

mean for x\_1 : [3. 3.8]

mean for x\_2 : [8.4 7.6 ]

Pooled covariance matrix :

[[86.1 50.1]

[50.1 58.1]]

f1 = 1.9839

f2 = 1.7051

The new observation [5, 6] is classified into group 1

**11) Aim: Write a python program to find Principal Components for the given variables.**

**Source Code:**

import numpy as np

import pandas as pd

n=int(input('Enter number of components :'))

l=[]

for i in range(n):

k=[float(x) for x in input().split()]

l.extend(k)

m=len(k)

x=np.array(l).reshape(n,m)

x=transpose(x)

print("x\n",x)

mean=np.sum(x,axis=0)/m

print("Mean\n",mean)

x\_mean=x-mean

c=mul(transpose(x\_mean),x\_mean)/m

print("c\n",c)

e\_values,e\_vectors=np.linalg.eig(c)

e1=np.argsort(e\_values)[::-1]

e\_values=e\_values[e1]

e\_vectors=e\_vectors[:,e1]

print('Eigen values:\n',e\_values)

print('Eigen vectors:\n',e\_vectors)

z=[]

z\_name=[]

sum=0

t\_sum=np.sum(e\_values)

for i in range(len(e\_values)):

sum=sum+e\_values[i]

z\_name.append('z'+str(i+1))

z.append(round(sum\*100/t\_sum,2))

print("Principal Components :",z)

threshold=int(input('Enter Threshold value :'))

c=0

for x in z:

if x<=threshold+2:

c=c+1

d={'principle components':z\_name,'variance explained':e\_values,'cummulative proportion of total variance':z}

df=pd.DataFrame(d)

print(df)

print("Principal Components are:")

for i in range(c):

print("z"+str(i+1)+" =",end='')

for y in range(n):

if y==n-1:

print(round(e\_vectors[y][i],4),"x"+str(y+1))

else:

print(round(e\_vectors[y][i],4),"x"+str(y+1)+" + ",end='')

p=mul(x,transpose(e\_vectors[:c]))

p=transpose(p)

d={}

for i in range(c):

d.update({'z'+str(i+1):p[i]})

df=pd.DataFrame(d)

df

**Input 1**:

Enter number of components :2

2 1 0 -1

4 3 1 0.5

**Output 1:**

x

[[ 2. 4. ]

[ 1. 3. ]

[ 0. 1. ]

[-1. 0.5]]

Mean

[0.5 2.125]

c

[[1.25 1.5625 ]

[1.5625 2.046875]]

Eigen values:

[3.26093826 0.03593674]

Eigen vectors:

[[-0.6135581 -0.78964958]

[-0.78964958 0.6135581 ]]

Principal Components : [98.91, 100.0]

Enter Threshold value :99

|  |  |  |  |
| --- | --- | --- | --- |
|  | principle components | variance explained | cummulative proportion of total variance |
| 0 | z1 | 3.260938 | 98.91 |
| 1 | z2 | 0.035937 | 100.00 |

Principal Components are:

z1 =-0.6136 x1 + -0.7896 x2

z2 =-0.7896 x1 + 0.6136 x2

|  | **z1** | **z2** |
| --- | --- | --- |
| **0** | -4.385715 | 0.874933 |
| **1** | -2.982507 | 1.051025 |
| **2** | -0.789650 | 0.613558 |
| **3** | 0.218733 | 1.096429 |

**Input 2:**

Enter number of components :3

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

**Output 2:**

x

[[7. 4. 3.]

[4. 1. 8.]

[6. 3. 5.]

[8. 6. 1.]

[8. 5. 7.]

[7. 2. 9.]

[5. 3. 3.]

[9. 5. 8.]

[7. 4. 5.]

[8. 2. 2.]]

Mean

[6.9 3.5 5.1]

c

[[ 2.09 1.45 -0.39]

[ 1.45 2.25 -1.15]

[-0.39 -1.15 7.09]]

Eigen values:

[7.44654832 3.30851634 0.67493534]

Eigen vectors:

[[-0.1375708 0.69903712 -0.70172743]

[-0.25045969 0.66088917 0.70745703]

[ 0.95830278 0.27307986 0.08416157]]

Principal Components : [65.15, 94.1, 100.0]

Enter Threshold value :93

Principle components variance explained \

0 z1 7.446548

1 z2 3.308516

2 z3 0.674935

cumulative proportion of total variance

0 65.15

1 94.10

2 100.00

Principal Components are:

z1 =-0.1376 x1 + -0.2505 x2 + 0.9583 x3

z2 =0.699 x1 + 0.6609 x2 + 0.2731 x3

|  | **z1** | **z2** |
| --- | --- | --- |
| **0** | -0.272029 | 3.012710 |
| **1** | -5.465065 | 5.318707 |
| **2** | -2.236951 | 4.017195 |
| **3** | 2.391929 | 2.669115 |
| **4** | -2.517473 | 6.252968 |
| **5** | -5.880468 | 5.935674 |
| **6** | -0.695925 | 2.852740 |
| **7** | -3.356771 | 6.709965 |
| **8** | -1.675484 | 4.427624 |
| **9** | -1.105947 | 0.733015 |

**12) Aim: Write a python program to group the given variables using Factor Analysis.**

**Source Code:**

import numpy as np

import pandas as pd

from math import floor

from math import sqrt

n=int(input('Enter number of components :'))

l=[]

for i in range(n):

  k=[float(x) for x in input().split()]

  l.extend(k)

  m=len(k)

x=np.array(l).reshape(n,m)

x=transpose(x)

print("x\n",x)

mean=np.sum(x,axis=0)/m

x\_mean=x-mean

s\_d=[]

s\_d=np.sum((x\_mean)\*\*2,axis=0)/(m-1)

for i in range(n):

  s\_d[i]=sqrt(s\_d[i])

x=x\_mean/s\_d

c=mul(transpose(x),x)/m

e\_values,e\_vectors=np.linalg.eig(c)

e1=np.argsort(e\_values)[::-1]

e\_values=e\_values[e1]

e\_vectors=e\_vectors[:,e1]

print('Eigen values:\n',e\_values)

print('Eigen vectors:\n',e\_vectors)

z=[]

z\_name=[]

sum=0

t\_sum=np.sum(e\_values)

for i in range(len(e\_values)):

  sum=sum+e\_values[i]

  z\_name.append('z'+str(i+1))

  z.append(round(sum\*100/t\_sum,2))

print("Principal Components :",z)

threshold=int(input('Enter Threshold value :'))

c=0

for x in z:

  if floor(x)<=threshold+2:

    c=c+1

print("Principal Components are:")

for i in range(c):

  print("z"+str(i+1)+" =",end='')

  for y in range(n):

    if y==n-1:

      print(round(e\_vectors[y][i],4),"x"+str(y+1))

    else:

      print(round(e\_vectors[y][i],4),"x"+str(y+1)+" + ",end='')

f=[]

for i in range(c):

  f1=[]

  for j in range(n):

    f1.append(sqrt(e\_values[i])\*e\_vectors[j][i])

  f.append(f1)

f1=f

f=transpose(np.array(f))

h2=np.sum(f\*\*2,axis=1)

s=np.sum(f,axis=0)

t\_s=np.sum(h2)

f1[0].extend([s[0],s[0]\*100/t\_s])

f1[1].extend([s[1],s[1]\*100/t\_s])

h2=list(h2)

h2.extend([t\_s,s[0]+s[1]])

df=pd.DataFrame({'variables':['Finance','Marketing','Business Policy','variance explained','%of variance explained'],'F1':f1[0],'F2':f1[1],'h^2':h2})

df

**Input 1:**

Enter number of components :3

3 7 10 3 10

6 3 9 9 6

5 3 8 7 5

**Output 1:**

x

[[ 3. 6. 5.]

[ 7. 3. 3.]

[10. 9. 8.]

[ 3. 9. 7.]

[10. 6. 5.]]

Eigen values:

[1.5851705 0.80664506 0.00818443]

Eigen vectors:

[[ 0.0212208 -0.99538347 -0.09360247]

[ 0.70623585 0.08119286 -0.70330551]

[ 0.70765853 -0.05118071 0.70469847]]

Principal Components : [66.05, 99.66, 100.0]

Enter Threshold value :97

Principal Components are:

z1 =0.0212 x1 + 0.7062 x2 + 0.7077 x3

z2 =-0.9954 x1 + 0.0812 x2 + -0.0512 x3

|  | **variables** | **F1** | **F2** | **h^2** |
| --- | --- | --- | --- | --- |
| **0** | Finance | 0.026718 | -0.893988 | 0.799928 |
| **1** | Marketing | 0.889176 | 0.072922 | 0.795952 |
| **2** | Business Policy | 0.890967 | -0.045967 | 0.795936 |
| **3** | variance explained | 1.806861 | -0.867033 | 2.391816 |
| **4** | %of variance explained | 75.543493 | -36.249995 | 0.939828 |

**Input 2:**

Enter number of components :3

2 4 1 5

2 8 4 0

8 3 1 2

**Output 2:**

x

[[2. 2. 8.]

[4. 8. 3.]

[1. 4. 1.]

[5. 0. 2.]]

Eigen values:

[0.89208647 0.82928548 0.52862805]

Eigen vectors:

[[-0.5267742 0.63311916 0.56715876]

[-0.34594212 -0.76916788 0.53731259]

[ 0.7764232 0.08683831 0.62420039]]

Principal Components : [39.65, 76.51, 100.0]

Enter Threshold value :75

Principal Components are:

z1 =-0.5268 x1 + -0.3459 x2 + 0.7764 x3

z2 =0.6331 x1 + -0.7692 x2 + 0.0868 x3

|  | **variables** | **F1** | **F2** | **h^2** |
| --- | --- | --- | --- | --- |
| **0** | Finance | -0.497540 | 0.576551 | 0.579957 |
| **1** | Marketing | -0.326743 | -0.700444 | 0.597383 |
| **2** | Business Policy | 0.733334 | 0.079079 | 0.544033 |
| **3** | variance explained | -0.090949 | -0.044814 | 1.721372 |
| **4** | %of variance explained | -5.283528 | -2.603361 | -0.135763 |