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

**Source Code :**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

x = list(map(float,input().split()))

y = list(map(float,input().split()))

n = len(x)

x**=**np**.**array(x)

y**=**np**.**array(y)

xy**=**x**\***y

xsquare**=**x**\*\***2

ysum**=**sum(y)

xsum**=**sum(x)

xysum**=**sum(xy)

xsquaresum**=**sum(xsquare)

n**=**x**.**size

slope**=**round(((n**\***xysum)**-**(xsum**\***ysum))**/**((n**\***xsquaresum)**-**(xsum**\*\***2)),4)

y\_intercept**=**round(((ysum)**-**(slope**\***xsum))**/**n,4)

ycal**=**[slope**\***i**+**y\_intercept **for** i **in** x]

ycal**=**np**.**array(ycal)

print(y)

print(ycal)

plt.scatter(x,y)

plt.plot(x,ycal,marker='\_',linestyle="--",c='green')

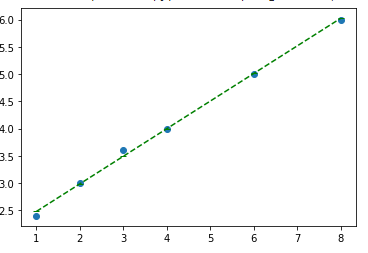
plt.show

**Output 1:**

X : 1 2 3 4 5

Y :1.8 5.1 8.9 14.1 19.8

Y CAL :



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

**Source Code :**

x = np.array(x)

y = np.array(y)

sigma\_x = sum(x)

sigma\_y = sum(y)

sigma\_x2 = sum(np.multiply(x,x))

sigma\_x3 = sum(np.multiply(x,np.multiply(x,x)))

sigma\_x4 = sum(np.multiply(np.multiply(x,x),np.multiply(x,x)))

sigma\_xy = sum(np.multiply(x,y))

sigma\_x2y = sum(np.multiply(np.multiply(x,x),y))

Det = det([[n,sigma\_x,sigma\_x2],[sigma\_x,sigma\_x2,sigma\_x3],[sigma\_x2,sigma\_x3,sigma\_x4]])

l = [sigma\_y,sigma\_xy,sigma\_x2y]

det1 = det([l,[sigma\_x,sigma\_x2,sigma\_x3],[sigma\_x2,sigma\_x3,sigma\_x4]])

det2 = det([[n,sigma\_x,sigma\_x2],l,[sigma\_x2,sigma\_x3,sigma\_x4]])

det3 = det([[n,sigma\_x,sigma\_x2],[sigma\_x,sigma\_x2,sigma\_x3],l])

a = np.round(det1/Det,2)

b =np.round( det2/Det,2)

c =np.round( det3/Det,2)

print("y = {0} + {1} X + {2} X^2".format(a,b,c))

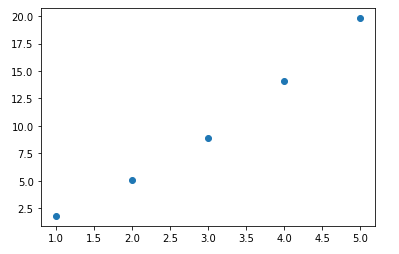
import matplotlib.pyplot as plt

plt.scatter(x,y)

plt.show()

**Output 1:**

y = -0.46 + 1.84 X + 0.44 X^2



**3.Aim : Write a python program to find Karl Pearson’s correlation coefficient.**

**Source Code:**

import numpy as np

import statistics as st

import math

x\_input=list(map(int,input("enter x values:").split()))

y\_input=list(map(int,input("enter y values:").split()))

x=np.array(x\_input)

y=np.array(y\_input)

print(x)

print(y)

xsum=sum(x)

ysum=sum(y)

xy=x\*y

xysum=sum(xy)

xsquare=x\*\*2

ysquare=y\*\*2

xsquaresum=sum(xsquare)

ysquaresum=sum(ysquare)

xbar=st.mean(x\_input)

ybar=st.mean(y\_input)

cov\_xy=(xysum/np.size(x))-(xbar\*ybar)

sigma\_x=math.sqrt((xsquaresum/np.size(x))-(xbar\*\*2))

sigma\_y=math.sqrt((ysquaresum/np.size(y))-(ybar\*\*2))

r=round(cov\_xy/(sigma\_x\*sigma\_y),4)

print('the correlation coefficient of given data is:',r)

r\_builtin=np.corrcoef(x, y)

print('the correlation coefficient of given data is:',round(r\_builtin[1,0],4))

**Output 1:**

enter x values:3 7 4 2 0 4 1 2

enter y values:11 18 9 4 7 6 3 8

[3 7 4 2 0 4 1 2]

[11 18 9 4 7 6 3 8]

the correlation coefficient of given data is: 0.7867

the correlation coefficient of given data is: 0.7867

**Output 2:**

enter x values:1 5 2 5 1

enter y values:9 8 5 9 1

[1 5 2 5 1]

[9 8 5 9 1]

the correlation coefficient of given data is: 0.5469

the correlation coefficient of given data is: 0.5469

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

**Source Code:**

import numpy as np

import pandas as pd

x = np.array(list(map(int,input().split())))

y = np.array(list(map(int,input().split())))

rank\_x = []

rank\_y = []

x\_copy = np.copy(x)

y\_copy = np.copy(y)

x\_copy = np.sort(x\_copy)[::-1]

y\_copy = np.sort(y\_copy)[::-1]

for i in x:

result = (np.where(x\_copy==i))

rank\_x.append((result[0][0]+1))

for i in y:

result = (np.where(y\_copy==i))

rank\_y.append((result[0][0]+1))

new\_rank\_x = []

new\_rank\_y = []

cf = {}

for i in rank\_x:

c = rank\_x.count(i)

if c>1:

if i not in cf:

cf[i] = (c\*(c\*\*2-1)/12)

new\_rank\_x.append(sum(range(i,i+c))/c)

else:

new\_rank\_x.append(i)

for i in rank\_y:

c = rank\_y.count(i)

if c>1:

if i not in cf:

cf[i] = (c\*(c\*\*2-1)/12)

new\_rank\_y.append(sum(range(i,i+c))/c)

else:

new\_rank\_y.append(i)

di = np.subtract(new\_rank\_x,new\_rank\_y)

di2 = di\*\*2

di2\_sum = sum(di2)

n = len(x)

row = 1-(6\*di2\_sum)/(n\*(n\*\*2-1))

di2\_sum = di2\_sum+sum(cf.values())

row = 1-(6\*di2\_sum)/(n\*(n\*\*2-1))

print(row)

**Output 1:**

81 78 73 73 69 68 62 58

10 12 18 18 18 22 20 24

-0.9285714285714286

**Output 2:**

3 7 4 2 0 4 1 2

11 18 9 4 7 6 3 8

0.6190476190476191

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

**Source Code:**

import scipy.stats as sci

n = int(input("Enter number of samples : "))

samples = []

for i in range(n):

sample = list(map(int,input().split()))

samples.append(sample)

los = int(input("Enter level of significance : "))

rss = 0

for i in samples:

for j in i:

rss+= j\*\*2

rss

Ti = []

Ti2\_ni = []

N = 0

for i in samples:

Ti.append(sum(i))

Ti2\_ni.append((sum(i)\*\*2)/len(i))

N += len(i)

sum\_Ti2\_ni = sum(Ti2\_ni)

G= sum(Ti)

cf = G\*\*2/N

sst = rss-cf

sstr = sum\_Ti2\_ni-cf

sse = sst-sstr

treatment = sstr/(n-1)

error = sse/(N-n)

f = treatment/error

print("RSS : "+str(rss)+"\n"+"CF : "+str(cf)+"\n"+"SST : "+str(sst)+"\n"+"SStr : "+str(sstr)+"\n"+"SSe : "+str(sse))

if f<1:

f = 1/f

f\_table = sci.f.ppf(1-los/100,N-n,n-1)

else:

f\_table = sci.f.ppf(1-los/100,n-1,N-n)

#inferance

if f>= f\_table:

print("Reject")

else:

print("Accept")

Output1 :

Enter number of samples : 4

6 4 5

13 10 13 12

7 9 11

3 6 1 4 1

[[6, 4, 5], [13, 10, 13, 12], [7, 9, 11], [3, 6, 1, 4, 1]]

Enter level of significance : 5

RSS : 973

CF : 735.0

SST : 238.0

SStr : 204.0

SSe : 34.0

Reject

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

**Source Code:**

import numpy as np

import scipy.stats as sci

numberOfTreatments= int(input("Enter number of treatments : "))

numberOfBlocks = int(input("Enter number of blocks : "))

los = int(input("Enter level of significance : "))

print("Enter treatments : ")

matrix = []

for i in range(numberOfTreatments):

print("Enter treatment {} data : ".format(i+1))

matrix.append(list(map(int,input().split())))

Ti = []

Ti2 = []

Bi = []

Bi2 = []

for i in range(numberOfTreatments):

Ti.append(sum(matrix[i]))

Ti2.append(sum(matrix[i])\*\*2)

for i in range(numberOfBlocks):

total = 0

for j in matrix:

total +=j[i]

Bi.append(total)

Bi2.append(total\*\*2)

G = sum(Ti)

sigmaTi2 = sum(Ti2)

sigmaBi2 = sum(Bi2)

RSS = 0

for i in matrix:

for j in i:

RSS += j\*\*2

CF = G\*\*2/(numberOfTreatments\*numberOfBlocks)

SST = RSS-CF

SStr = (1/numberOfBlocks)\*sigmaTi2 - CF

SSb = (1/numberOfTreatments)\*sigmaBi2 - CF

SSe = SST - SStr - SSb

print("RSS : {}\nCF : {}\nSST : {}\nSStr : {}\nSSb : {}\nSSe : {}".format(RSS,CF,SST,SStr,SSb,SSe))

degreeOfFreedomForTreatments = numberOfTreatments - 1

degreeOfFreedomForBlocks = numberOfBlocks - 1

degreeOfFreedomForErrors = degreeOfFreedomForTreatments\*degreeOfFreedomForBlocks

MSStr = SStr/degreeOfFreedomForTreatments

MSSb = SSb/degreeOfFreedomForBlocks

MSSe = SSe/degreeOfFreedomForErrors

print("degreeOfFreedomForTreatments : {}\ndegreeOfFreedomForBlocks : {}\ndegreeOfFreedomForErrors : {}\nMSStr : {}\nMSSb : {}\nMSSe : {}".format(degreeOfFreedomForTreatments,degreeOfFreedomForBlocks,degreeOfFreedomForErrors,MSStr,MSSb,MSSe))

Ftr = MSStr/MSSe

Fb = MSSb/MSSe

print("========F Cal ===============\nF value due to treatments : {}\nF-value due to blocks : {}".format(Ftr,Fb))

Ftr\_table = sci.f.ppf(1-los/100,degreeOfFreedomForTreatments,degreeOfFreedomForErrors)

Fb\_table = sci.f.ppf(1-los/100,degreeOfFreedomForBlocks,degreeOfFreedomForErrors)

print("========F Table ===============\nF value due to treatments : {}\nF-value due to blocks : {}".format(Ftr\_table,Fb\_table))

if Ftr >= Ftr\_table:

print("Reject")

else:

print("Accept")

if Fb >= Fb\_table:

print("Reject")

else:

print("Accept")

**Output 1:**

Enter number of treatments : 4

Enter number of blocks : 5

Enter level of significance : 1

Enter treatments :

Enter treatment 1 data :

6 14 10 8 11

Enter treatment 2 data :

14 9 12 10 14

Enter treatment 3 data :

10 12 7 15 11

Enter treatment 4 data :

9 12 8 10 11

RSS : 2383

CF : 2268.45

SST : 114.55000000000018

SStr : 12.950000000000273

SSb : 20.800000000000182

SSe : 80.79999999999973

degreeOfFreedomForTreatments : 3

degreeOfFreedomForBlocks : 4

degreeOfFreedomForErrors : 12

MSStr : 4.316666666666758

MSSb : 5.2000000000000455

MSSe : 6.73333333333331

========F Cal ===============

F value due to treatments : 0.6410891089109069

F-value due to blocks : 0.7722772277227816

========F Table ===============

F value due to treatments : 5.952544681545868

F-value due to blocks : 5.4119514344731385

Accept

Accept

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

**Source Code:**

import numpy as np

import scipy.stats as ss

import math

y = np.array(list(map(int,input('enter y').split())))

x1 = np.array(list(map(int,input('enter x1').split())))

x2 = np.array(list(map(int,input('enter x2').split())))

x0 = np.ones(len(x1),int)

xt = np.array([x0,x1,x2])

x = xt.transpose()

y = y.reshape([len(y),1])

los = 0.5

b1 = np.dot(xt,x)

b1inv = np.linalg.inv(b1)

cjj = [b1inv[i][i] for i in range(3)]

b2 = np.dot(xt,y)

b = np.dot(b1inv,b2)

ans = b.reshape([len(b)])

yExp = []

for i,j in zip(x1,x2):

yExp.append((ans[0]+ans[1]\*i+ans[2]\*j))

yExp = np.array(yExp)

yExp = yExp.reshape([len(yExp),1])

epsil = y - yExp

ybar = np.mean(y)

sst = sum((y-ybar)\*\*2)[0]

sse = sum(epsil\*\*2)[0]

ssr = sst-sse

R2 = ssr/sst

import pandas as pd

val={

'x1':x1,

'x2':x2,

'observed y':y.transpose()[0],

'expected y':yExp.transpose()[0],

'error':epsil.transpose()[0]

}

print(f'The model is : y = {ans[0]}+({ans[1]})\*x1+({ans[2]})\*x2')

line()

df = pd.DataFrame(val)

print(df)

line()

print(f'mean sum of squares due to \n -->regression {ssr}\n -->total {sst} \n -->error {sse}')

line()

print(f'The coefficient of Determination is R2 : {R2}')

line()

if(R2>0.9):

print('As R2 is greater than 0.9, the model is a goodfit')

else:

print('As R2 is less than 0.9, the model is not a goodfit')

Output 1:

The model is : y = 133.46048242804682+(-1.2485034569591562)\*x1+(-0.351008371805392)\*x2

-------------------------------------------------------------------------------------

x1 x2 observed y expected y error

0 9 62 100 100.461432 -0.461432

1 8 58 110 103.113969 6.886031

2 7 64 105 102.256422 2.743578

3 14 60 94 94.920932 -0.920932

4 12 63 95 96.364914 -1.364914

5 10 57 99 100.967971 -1.967971

6 7 55 104 105.415498 -1.415498

7 4 56 108 108.810000 -0.810000

8 6 59 105 105.259968 -0.259968

9 5 61 98 105.806454 -7.806454

10 7 57 105 104.713481 0.286519

11 6 60 110 104.908959 5.091041

-------------------------------------------------------------------------------------

mean sum of squares due to

-->regression 178.83959377968236

-->total 330.25

-->error 151.41040622031764

-------------------------------------------------------------------------------------

The coefficient of Determination is R2 : 0.5415279145486218

-------------------------------------------------------------------------------------

As R2 is less than 0.9, the model is not a goodfit

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

**Source code :**

import numpy as np

import math

n = int(input("Enter number of variants : "))

means = []

for i in range(n):

means.append( int(input()))

sigma = []

for i in range(n):

print("Enter the {} row : ".format(i+1))

sigma.append( list(map(float,input().split())))

sigma = np.array(sigma)

sigma\_det = np.linalg.det(sigma)

sigma\_inverse = np.linalg.inv(sigma)

a = sigma\_inverse[0][0]

b = sigma\_inverse[1][0]

c = sigma\_inverse[1][1]

squared\_distance = "{:.2f}x1^2 + {:.2f}x2^2 + {:.2f}x1 + {:.2f}x2 + {:.2f}x1x2 + {:.2f}".format(a,c,2\*a\*means[0]-2\*c\*means[1],-2\*b\*means[0]-2\*c\*means[1],2\*b,a\*means[0]\*\*2+c\*means[1]\*\*2+2\*b\*means[1]\*means[0])

print(squared\_distance)

a= np.round(1/(2\*math.pi\*math.sqrt(sigma\_det)),2)

result = "{} e ^ {}".format(a,squared\_distance)

result

Output 1:

Enter number of variants : 2

1

3

Enter the 1 row :

2 -1.131

Enter the 2 row :

-1.131 1

1.39x1^2 + 2.77x2^2 + -13.87x1 + -19.79x2 + 3.14x1x2 + 35.77

0.19 e ^ 1.39x1^2 + 2.77x2^2 + -13.87x1 + -19.79x2 + 3.14x1x2 + 35.77

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

**Source Code:**

data = []

k = int(input("Enter number of treatments : "))

for i in range(k):

num = int(input("Enter the number of groups in treatment {} : ".format(i+1)))

*#taking the data in y1,y2*

print("Enter treatment {} data : ".format(i+1))

treatment = []

for j in range(num):

treatment.append(list(map(int,input().split())))

data.append(treatment)

(s,s3,s4,n) = (0,0,0,0)

mean\_yi = []

for treatment in data:

s1 = 0

s2 = 0

for group in treatment:

s1 += group[0]

s2 += group[1]

n+=1

s3 += s1

s4 += s2

mean\_yi.append([s1/len(treatment),s2/len(treatment)])

mean\_yii= [s3/n,s4/n]

sse\_y1,sst\_y1,sst\_y2,sse\_y2,sse\_y12,sst\_y12 = 0,0,0,0,0,0

for i in range(k):

for j in data[i]:

sse\_y1 += (j[0]-mean\_yi[i][0])\*\*2

sst\_y1 += (j[0]-mean\_yii[0])\*\*2

sse\_y2 += (j[1]-mean\_yi[i][1])\*\*2

sst\_y2 += (j[1]-mean\_yii[1])\*\*2

sse\_y12 += (j[0]\*j[1]-mean\_yi[i][0]\*mean\_yi[i][1])

sst\_y12 += (j[0]\*j[1]-mean\_yii[0]\*mean\_yii[1])

ssr\_y1 = sst\_y1 - sse\_y1

ssr\_y2 = sst\_y2 - sse\_y2

ssr\_y12 = sst\_y12 - sse\_y12

import numpy as np

import math

B = np.array([[ssr\_y1,ssr\_y12],[ssr\_y12,ssr\_y2]])

W = np.array([[sse\_y1,sse\_y12],[sse\_y12,sse\_y2]])

T = np.array([[sst\_y1,sst\_y12],[sst\_y12,sst\_y2]])

degreeOfFreedom\_regression **=** k **-** 1

degreeOfFreedom\_error **=** n **-** k

degreeOfFreedom\_total **=** n **–** 1

det\_W **=** np**.**linalg**.**det(W)

det\_T **=** np**.**linalg**.**det(T)

lamda **=** abs(det\_W)**/**abs(det\_T)

f\_cal **=** ((n**-**k**-**1)**/**(k**-**1))**\***((1**-**math**.**sqrt(lamda))**/**math**.**sqrt(lamda))

**import** scipy.stats **as** sci

f\_table **=** sci**.**f**.**ppf(0.95,2**\***(k**-**1),2**\***(n**-**k**-**1))

table **=** {

'Source of variation ':['Regression','Error','Total'],

'Sum of squares ':[B,W,T],

'degree of freedom ':[degreeOfFreedom\_regression,degreeOfFreedom\_error,degreeOfFreedom\_total],

'Wills value':['',lamda,''],

'f-statistic':['',f\_cal,'']

}

import pandas as pd

pd.DataFrame(table)

if f\_cal>=f\_table:

print("Reject H0")

else:

print("Accept\_H0")

**Output1 :**

Enter number of treatments : 3

Enter the number of groups in treatment 1 : 3

Enter treatment 1 data :

9 3

6 2

9 7

Enter the number of groups in treatment 2 : 2

Enter treatment 2 data :

0 4

2 0

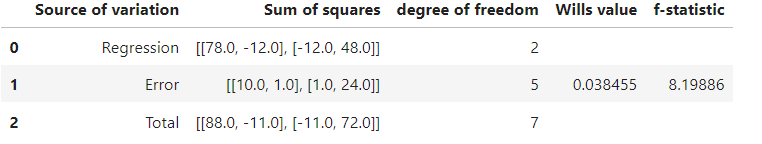
Enter the number of groups in treatment 3 : 3

Enter treatment 3 data :

3 8

1 9

2 7



Reject H0

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

**Source Code:**

n = int(input("Enter number of atributes : "))

x = []

for i in range(n-1):

print("Enter attribute {} data : ".format(i+1))

x.append(list(map(float,input().split())))

print("Enter the status of y : ")

y = list(map(int,input().split()))

print("Enter data to predict : ")

xk = np.array(list(map(float,input().split())))

import numpy as np

X = np.array(x)

X = np.transpose(X)

Y = np.array(y)

Y = np.transpose(Y)

mean = []

count = 0

for i in range(n-1):

s = 0

count = 0

for j in X:

s += j[i]

count += 1

mean.append(np.round(s/count,2))

cat = []

for i in y:

if i not in cat:

cat.append(i)

x1 = []

x2 = []

for i in range(n-1):

row1 = []

row2 = []

for j in range(count):

if Y[j] == cat[0]:

row1.append(X[j][i])

else:

row2.append(X[j][i])

x1.append(row1)

x2.append(row2)

mean1 = []

mean2 = []

for i in x1:

c = 0

s = 0

for j in i:

s += j

c += 1

mean1.append(np.round(s/c,2))

for i in x2:

c = 0

s = 0

for j in i:

s += j

c += 1

mean2.append(np.round(s/c,2))

mean1 = np.array(mean1)

mean2 = np.array(mean2)

x\_mean = X- mean

c = np.dot(np.transpose(x\_mean),x\_mean)/count

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

xk

f = []

mu = [mean1,mean2]

for i in range(n-1):

f.append(np.dot(np.dot(mu[i],c\_inv),np.transpose(xk)) - 0.5\*np.dot(np.dot(mu[i],c\_inv),np.transpose(mu[i]))+np.log(y.count(cat[i])/count))

f = np.round(f,2)

if f[0]>f[1]:

print("The entries belongs to first catogory")

else:

print("The entries belongs to second catogory")

**Ouput1 :**

Enter number of atributes : 3

Enter attribute 1 data :

1 2 3 4 5 4 5 5 3 5 6

Enter attribute 2 data :

2 3 3 5 5 2 0 2 2 3 3

Enter the status of y :

1 1 1 1 1 2 2 2 2 2 2

Enter data to predict :

5.1 3.2

The entries belongs to second category.

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

**Source Code:**

import numpy as np

import pandas as pd

n = int(input("Enter number of attributes : "))

x = []

for i in range(n):

x.append(list(map(float,input().split())))

X = np.array(x)

X = np.transpose(X)

meaw = np.mean(X,axis=0)

X\_meaw = X- meaw

X\_meaw

c = np.dot(np.transpose(X\_meaw),X\_meaw)/len(X)

eigenValues,eigenVectors = np.linalg.eig(c)

sort\_id = eigenValues.argsort()[::-1]

eigenValues = eigenValues[sort\_id]

eigenVectors = eigenVectors[:,sort\_id]

eigenVectors = np.transpose(eigenVectors)

print(eigenValues,eigenVectors)

thresHold = int(input("Enter threshold percentage : "))

f = eigenValues[0]

s = sum(eigenValues)

k = [np.round((f/s)\*100,2)]

count = 1

flag = 0

for i in range(1,n):

f += eigenValues[i]

k.append(np.round((f/s)\*100,2))

print(k,thresHold)

if k[-1] <= thresHold or flag == 0:

flag = 1

count += 1

dataFrame = {

"Principle component" : ['z'+str(i+1) for i in range(n)],

"Variance explained" : eigenValues,

"Cumulative proportion of total Vaiance " : k

}

pca\_table = pd.DataFrame(dataFrame)

pca\_table

z = np.dot(eigenVectors[:count],X.T)

for i in range(len(z)):

neg = 0

for j in z[i]:

if j<0:

neg+=1

if neg >= len(z[i])//2:

z[i] = z[i]\*(-1)

break

x\_ = X.T

result = {}

for i in range(len(x\_)):

result[f'x{i+1}'] = x\_[i]

for i in range(len(z)):

result[f'z{i+1}'] = z[i]

In [ ]:

table2 = pd.DataFrame(result)

print(table2)

Output1 :

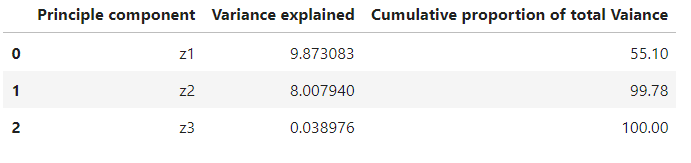
Enter number of attributes : 3

3 7 10 3 10

6 3 9 9 6

5 3 8 7 5

Enter threshold percentage : 95





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

**Source Code :**

n=int(input("enter no of variables"))

x=[]

for i in range(n):

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

x.append(p)

print(x)

mu=[]

for i in range(n):

mu.append(np.mean(x[i],axis=0))

print(mu)

from math import sqrt

from statistics import mean

n1=len(x[0])

si=[]

x=np.array(x)

for i in range(n):

k=sum((x[i]-mu[i])\*\*2)

si.append(round(sqrt(k/(n1-1)),4))

print(si)

a=[]

for i in range(n):

a.append((x[i]-mu[i])/si[i])

print(a)

A=np.transpose(a)

#variance covariance matrix

VarCov=np.dot(np.transpose(A),A)/n1

print('\nVariance Covariance Matrix is :\n {}'.format(VarCov))

#calculating eigen values and vectors

eigenValues, eigenVectors = np.linalg.eig(VarCov)

idx = eigenValues.argsort()[::-1]

eigenValues = eigenValues[idx]

eigenVectors = eigenVectors[:,idx]

print('\nEigen Values are {}'.format(eigenValues))

print('\nEigen Vectors are {}'.format(eigenVectors))

#no of principal components to be retained

k=int(input("enter threshold limit :"))

s=sum(eigenValues)

t=[]

stoppoint=1

for i in range(len(eigenValues)):

numerator=sum(eigenValues[0:i+1])

z=(numerator/s)\*100

if z<=k:

stoppoint=stoppoint+1

t.append(z)

print('\nThreshold Table :\n{}'.format(t))

eigenValues=eigenValues[0:stoppoint]

eigenVectors=eigenVectors[:,0:stoppoint]

#retaining eigen values and vectors

print('\nRetained Eigen Values :\n{}'.format(eigenValues))

print('\nRetained Eigen Vectors: \n{}'.format(eigenVectors))

egv=np.transpose(eigenVectors)

f=[]

for i in range(len(egv)):

o=[]

for j in range(n):

k=sqrt(eigenValues[i])\*egv[i][j]

o.append(k)

f.append(o)

print(f)

h=[]

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

h.append(f[0][i]\*2+f[1][i]\*2)

print(h)

sumh=sum(h)

print(sumh)

pve=[]

for i in range(len(eigenValues)):

pve.append((eigenValues[i]/sumh)\*100)

print(pve)

Output 1 :

enter number of components:3

enter no of observations:5

3 7 10 3 10

6 3 9 9 6

5 3 8 7 5

X is: [[ 3. 6. 5.]

[ 7. 3. 3.]

[10. 9. 8.]

[ 3. 9. 7.]

[10. 6. 5.]]

