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

**Source code:-**

import numpy as np

import matplotlib.pyplot as plt

def mean(x):

    len=0

    sum=0

    for i in x:

        len+=1

        sum+=i

    return sum/len

x=np.array(list(map(float, input("Enter x values : ").split(","))))

y=np.array(list(map(float, input("Enter y values : ").split(","))))

x\_bar=mean(x)

y\_bar=mean(y)

num=0

den=0

length=len(x)

for i in range(length):

    num+=((x[i]-x\_bar)\*(y[i]-y\_bar))

    den+=(x[i]-x\_bar)\*\*2

slope=num/den

slope=slope.round(4)

print(f"slope={slope}")

y\_intercept=y\_bar-(slope\*x\_bar)

y\_intercept=y\_intercept.round(4)

print(f"y\_intercept={y\_intercept}")

print(f"the required line is y={slope}x+{y\_intercept}")

y1=[]

for i in range(len(x)):

  y1.append(slope\*x[i] + y\_intercept)

plt.scatter(y,x)

plt.scatter(y1,x)

plt.xlabel("X-axis")

plt.ylabel("y-axis")

plt.title("linear regression")

plt.plot(y,x)

plt.plot(y1,x,'r')

plt.show()

sse=0

sst=0

ssr=0

for i in range(length):

    sse=sse+(y[i]-(slope\*x[i]+y\_intercept))\*\*2

    sst+=(y\_bar-y[i])\*\*2

    ssr+=((slope\*x[i]+y\_intercept)-y\_bar)\*\*2

print("y = ",y)

for i in range(length):

    print(slope\*x[i]+y\_intercept)

sse=sse.round(4)

ssr=ssr.round(4)

r\_square=1-(sse/sst)

r\_square=r\_square.round(4)

print(f"cost function of the line is {sse}")

print(f"The regression is {ssr}")

print(f"total error is {sst}")

print(f"The Goodness of fit model {r\_square}")

**Output 1:-**

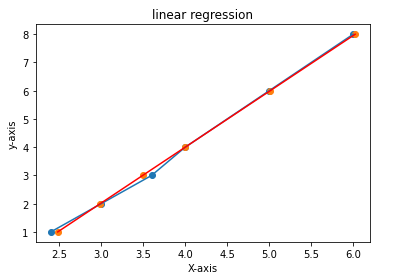
Enter x values : 1,2,3,4,6,8

Enter y values : 2.4,3,3.6,4,5,6

slope=0.5059

y\_intercept=1.9764

the required line is y=0.5059x+1.9764



y = [2.4 3. 3.6 4. 5. 6. ]

2.4823

2.9882

3.4941

4.0

5.0118

6.0236

cost function of the line is 0.0188

The regression is 8.7018

total error is 8.72

The Goodness of fit model 0.9978

**Output 2:-**

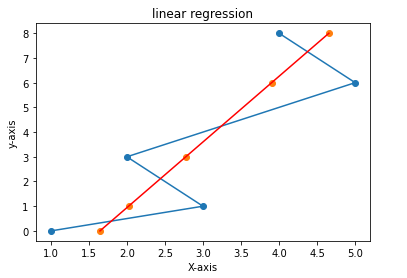
Enter x values : 0,1,3,6,8

Enter y values : 1,3,2,5,4

slope=0.3761

y\_intercept=1.646

the required line is y=0.3761x+1.646



y = [1. 3. 2. 5. 4.]

1.646

2.0221

2.7742999999999998

3.9025999999999996

4.6548

cost function of the line is 3.6062

The regression is 6.3936

total error is 10.0

The Goodness of fit model 0.6394

**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:-**

import numpy as np

import matplotlib.pyplot as plt

x=list(map(float, input().split(",")))

y=list(map(float, input().split(",")))

def mean(x):

    len=0

    sum=0

    for i in x:

        len+=1

        sum+=i

    return sum/len

xy=[]

x2y=[]

x2=[]

x3=[]

x4=[]

for i in range(len(x)):

  xy.append(x[i]\*y[i])

  x2y.append(x[i]\*x[i]\*y[i])

  x2.append(x[i]\*x[i])

  x3.append(x2[i]\*x[i])

  x4.append(x2[i]\*x2[i])

#print(x,y,xy,x2y,x2,x3,x4)

print("Equations to be solved are : ")

print("{} = {}a + {}b + {}c".format(sum(y),len(x),sum(x),sum(x2)))

print("{} = {}a + {}b + {}c".format(sum(xy),sum(x),sum(x2),sum(x3)))

print("{} = {}a + {}b + {}c".format(sum(x2y),sum(x2),sum(x3),sum(x4)))

def getMinor(m,i,j):

  return [p[:j]+p[j+1:] for p in m[:i]+m[i+1:]]

def getDeterminant(a):

  if len(a)==1:

    return a[0][0]

  if len(a)==2:

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

  det=0

  for c in range(len(a)):

    det+=(-1)\*\*c\*a[0][c]\*getDeterminant(getMinor(a,0,c))

  return det

d=[[len(x),sum(x),sum(x2)],[sum(x),sum(x2),sum(x3)],[sum(x2),sum(x3),sum(x4)]]

d1=[[sum(y),sum(xy),sum(x2y)],[sum(x),sum(x2),sum(x3)],[sum(x2),sum(x3),sum(x4)]]

d2=[[len(x),sum(x),sum(x2)],[sum(y),sum(xy),sum(x2y)],[sum(x2),sum(x3),sum(x4)]]

d3=[[len(x),sum(x),sum(x2)],[sum(x),sum(x2),sum(x3)],[sum(y),sum(xy),sum(x2y)]]

a=getDeterminant(d1)/getDeterminant(d)

b=getDeterminant(d2)/getDeterminant(d)

c=getDeterminant(d3)/getDeterminant(d)

print()

print('a = {},b = {},c = {}'.format(a,b,c))

print('final equation is :')

print('y = {} + {}x + {}x^2'.format(a,b,c))

print()

y1=[]

for i in range(len(y)):

  y1.append(a + b\*x[i] + c\*x[i]\*x[i])

plt.xlabel('x')

plt.ylabel('y')

plt.plot(y,x)

plt.plot(y1,x,'r')

plt.show()

**Output 1:-**

0,1,2,3,4

1,1.8,1.3,2.5,6.3

Equations to be solved are :

12.899999999999999 = 5a + 10.0b + 30.0c

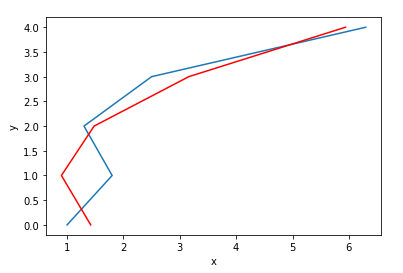
37.1 = 10.0a + 30.0b + 100.0c

130.3 = 30.0a + 100.0b + 354.0c

a = 1.4200000000000026,b = -1.0700000000000116,c = 0.5500000000000025

final equation is :

y = 1.4200000000000026 + -1.0700000000000116x + 0.5500000000000025x^2



**Output 2:-**

2,4,6,8,10

3.07,12.85,31.47,57.38,91.29

Equations to be solved are :

196.06 = 5a + 30.0b + 220.0c

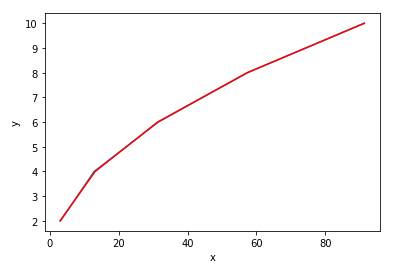
1618.3000000000002 = 30.0a + 220.0b + 1800.0c

14152.12 = 220.0a + 1800.0b + 15664.0c

a = 0.6959999999996008,b = -0.8550714285710326,c = 0.9919642857143065

final equation is :

y = 0.6959999999996008 + -0.8550714285710326x + 0.9919642857143065x^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)

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 by builtin function 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 by builtin function is: 0.7867

**Output 2:-**

enter x values:35 39 45 53 63 75 89 105 117

enter y values:181 151 126 106 91 81 76 68 65

[ 35 39 45 53 63 75 89 105 117]

[181 151 126 106 91 81 76 68 65]

the correlation coefficient of given data is: -0.8834

the correlation coefficient of given data by builtin function is: -0.8834

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

**Source code:-**

import numpy as np

cf=[]

def find\_rank(l):

    r={}

    s=list(set(l))

    w=l.copy()

    s.sort(reverse=True)

    w.sort(reverse=True)

    a=1

    for i in s:

        m=l.count(i)

        if m==1:

            r[i]=a

            a+=1

        else:

            q=0

            for j in range(m):

                q+=a

                a+=1

            r[i]=q/m

            cf.append(m\*(m\*\*2-1)/12)

    return r

x=list(map(int, input().split(",")))

y=list(map(int, input().split(",")))

x\_rank=find\_rank(x)

y\_rank=find\_rank(y)

n=len(x)

d=[]

for i in range(n):

    d.append(x\_rank[x[i]]-y\_rank[y[i]])

d\_2=[i\*i for i in d]

if len(cf)!=0:

    sum\_d\_2=sum(d\_2)+sum(cf)

else:

    sum\_d\_2=sum(d\_2)

print(d)

print(d\_2)

print(x\_rank)

print(cf)

print(sum\_d\_2)

num=6\*sum\_d\_2

den=n\*(n\*\*2 - 1)

print(num)

print(den)

rankcorelation=1 - num/den

print("Rank corelation is {:.5f}".format(rankcorelation))

**Output 1:-**

81,78,73,73,69,68,62,58

10,12,18,18,18,22,20,24

[-7, -5, -1.5, -1.5, 0.0, 4, 4, 7]

[49, 25, 2.25, 2.25, 0.0, 16, 16, 49]

{81: 1, 78: 2, 73: 3.5, 69: 5, 68: 6, 62: 7, 58: 8}

[0.5, 2.0]

162.0

972.0

504

Rank corelation is -0.92857

**Output 2:-**

68,64,75,50,64,80,75,40,55,64

62,58,68,45,81,60,68,48,50,70

[-1, -1.0, -1.0, -1, 5.0, -5, -1.0, 1, 0, 4.0]

[1, 1.0, 1.0, 1, 25.0, 25, 1.0, 1, 0, 16.0]

{80: 1, 75: 2.5, 68: 4, 64: 6.0, 55: 8, 50: 9, 40: 10}

[0.5, 2.0, 0.5]

75.0

450.0

990

Rank corelation is 0.54545

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

**Source code:-**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

import scipy.stats as s

print("ANOVA ONE WAY CLASSIFICATION")

deg=float(input("Enter the degrees of freedom:"))

tn=int(input("Enter the no.of Treatments: "))

f\_list=[]

d=[]

for i in range(tn):

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

  d.append(len(l))

  f\_list.append(l)

#print(f\_list)

ti=[]

for i in f\_list:

  ti.append(sum(i))

#print(ti)

ti\_sum=sum(ti)

#print(ti\_sum)

ti2\_ni=[]

for i in range(len(ti)):

  ti2\_ni.append((ti[i]\*\*2)/d[i])

#print(ti2\_ni)

#print(sum(ti2\_ni))

rss=0.0

for i in f\_list:

  for j in range(len(i)):

    rss+=(i[j]\*\*2)

cf=(ti\_sum\*\*2)/sum(d)

sst=rss-cf

sstr=sum(ti2\_ni)-cf

sse=sst-sstr

print("The row sum of squares: ",rss)

print("The correction factor:",cf)

print("sum of squares due to total:",sst)

print("sum of squares due to treatments:",sstr)

print("sum of squares due to error:",sse)

#print("sst:{},sstr:{},sse:{}".format(sst,sstr,sse));

#Constructing a dataframe

df=pd.DataFrame()

df["source of variation"]=["treatments","errors","total"]

df["sum of squares"]=[sstr,sse,sst]

x=(tn-1)

y=sum(d)-tn

z=sum(d)-1

df["Degrees of freedom"]=[x,y,z]

x1=sstr/x

y1=sse/y

df["Mean sum of Squares"]=[x1,y1,"\*"]

f=x1/y1

df["Variance ratio"]=["\*",f,"\*"]

print(df)

if f<1:

  f=1/f

  tv=s.f.ppf(1-deg,y,x)

else:

  tv=s.f.ppf(1-deg,x,y)

print("f is :",f)

print("The table value is:",tv)

#Inference or decision

if f>=tv:

  print("Reject the Null Hypothesis")

else:

  print("Accept the NUll Hypothesis")

**Output 1:-**

ANOVA ONE WAY CLASSIFICATION

Enter the degrees of freedom:0.05

Enter the no.of Treatments: 3

13 10 8 11 8

13 11 14 14

4 1 3 4 2 4

The row sum of squares: 1262.0

The correction factor: 960.0

sum of squares due to total: 302.0

sum of squares due to treatments: 270.0

sum of squares due to error: 32.0

source of variation sum of squares Degrees of freedom Mean sum of Squares \

0 treatments 270.0 2 135.0

1 errors 32.0 12 2.666667

2 total 302.0 14 \*

Variance ratio

0 \*

1 50.625

2 \*

f is : 50.625

The table value is: 3.8852938346523933

Reject the Null Hypothesis

**Output 2:-**

ANOVA ONE WAY CLASSIFICATION

Enter the degrees of freedom:0.05

Enter the no.of Treatments: 3

90 82 79 98 83 91

105 89 93 104 89 95 86

83 89 80 94

The row sum of squares: 138638.0

The correction factor: 137700.0

sum of squares due to total: 938.0

sum of squares due to treatments: 234.4523809523671

sum of squares due to error: 703.5476190476329

source of variation sum of squares Degrees of freedom Mean sum of Squares \

0 treatments 234.452381 2 117.22619

1 errors 703.547619 14 50.253401

2 total 938.000000 16 \*

Variance ratio

0 \*

1 2.332702

2 \*

f is : 2.3327016142676427

The table value is: 3.738891832440735

Accept the NUll Hypothesis

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

**Source code:-**

import scipy.stats as stats

print("Enter level of significance")

alpha=float(input())

print("Enter number of treatments")

t=int(input())

print("Enter number of blocks")

b=int(input())

l=[]

ti=[]

ti2ni=[]

for i in range(t):

    p=list(map(int,input().split()))

    ti.append(sum(p))

    ti2ni.append(sum(p)\*\*2)

    l.append(p)

sti2=sum(ti2ni)

blocks=[]

squared\_blocks=[]

for i in range(b):

  s=0

  for k in range(t):

    s+=l[k][i]

  blocks.append(s)

  squared\_blocks.append(s\*\*2)

G=sum(blocks)

sbi2=sum(squared\_blocks)

m=[]

for i in l:

  m.extend(i)

rss=0

for i in m:

  rss+=i\*\*2

cf=(G\*\*2) / len(m)

st2=rss-cf

str2=(sti2/b) - cf

sb2=(sbi2/t) - cf

se2=st2 - str2 - sb2

dft=t-1

dfb=b-1

dfe=dft\*dfb

mst2=str2/dft

msb2=sb2/dfb

mse2=se2/dfe

ftr=mst2/mse2

fb=msb2/mse2

print("rss : {} ,cf : {} ,st2 : {} ,str2 : {} ,sb2 : {} ,se2 : {} ,ftr : {} ,fb : {}".format(rss,cf,st2,str2,sb2,se2,ftr,fb))

if ftr<1.0:

  ftr=(1.0/ftr)

  t\_ftr=stats.f.ppf(1-alpha,dfe,dft)

  print("table value for treatments")

  print(t\_ftr)

  if ftr>t\_ftr:

    print("we reject treatments")

  else:

    print("we accept treatments")

else:

  t\_ftr=stats.f.ppf(1-alpha,dft,dfe)

  print("table value for treatments")

  print(t\_ftr)

  if ftr>t\_ftr:

    print("we reject treatments")

  else:

    print("we accept treatments")

if fb<1.0:

  fb=(1.0/fb)

  t\_fb=stats.f.ppf(1-alpha,dfe,dfb)

  print("Table value for blocks")

  print(t\_fb)

  if fb>t\_fb:

    print("we reject blocks")

  else:

    print("we accept blocks")

else:

  print("Table value for blocks")

  t\_fb=stats.f.ppf(1-alpha,dfb,dfe)

  print(t\_fb)

  if fb>t\_fb:

    print("we reject blocks")

  else:

    print("we accept blocks")

**Output 1:-**

Enter level of significance

0.05

Enter number of treatments

3

Enter number of blocks

4

13 7 9 3

6 6 3 1

11 5 15 5

rss : 786 ,cf : 588.0 ,st2 : 198.0 ,str2 : 56.0 ,sb2 : 90.0 ,se2 : 52.0 ,ftr : 3.230769230769231 ,fb : 3.4615384615384617

table value for treatments

5.143252849784718

we accept treatments

Table value for blocks

4.757062663089414

we accept blocks

**Output 2:-**

Enter level of significance

0.01

Enter number of treatments

4

Enter number of blocks

3

45 43 51

47 46 52

48 50 55

42 37 49

rss : 26867 ,cf : 26602.083333333332 ,st2 : 264.9166666666679 ,str2 : 110.91666666666788 ,sb2 : 135.16666666666788 ,se2 : 18.83333333333212 ,ftr : 11.77876106194779 ,fb : 21.530973451329015

table value for treatments

9.779538240923273

we reject treatments

Table value for blocks

10.92476650083833

we reject blocks

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

**Source code:-**

import scipy.stats as stats

def transportMatrix(a):

  res = [[a[j][i] for j in range(len(a))] for i in range(len(a[0]))]

  return res

import numpy as np

no\_ind=int(input("Enter no of independent variables"))

indep=[]

for i in range(no\_ind):

  print(f"Enter the values for independent variable {i+1}")

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

  indep.append(li)

print("Enter the values of output")

y=[float(x) for x in input().split()]

alpha=float(input("Enter level of significance"))

x\_trans=[]

x\_trans.append(np.ones(len(indep[0])))

for i in range(len(indep)):

  x\_trans.append(indep[i])

s1=np.dot(x\_trans,transportMatrix(x\_trans))

s2=np.linalg.inv(s1)

s3=np.dot(x\_trans,y)

s4=np.dot(s2,s3)

s4=s4.round(4)

y\_equ=[x.round(4) for x in s4]

y\_equ=str(s4[0])

for i in range(1,no\_ind+1):

    y\_equ+='+'+str(s4[i])+'x'+str(i)

print()

print(f"The required multi linear regression is y={y\_equ}")

print()

y\_fit=[float(0) for i in range(len(indep[0]))]

for i in range(len(indep[0])):

    y\_fit[i]=s4[0]

    k=0

    for j in range(1,no\_ind+1):

        y\_fit[i]+=s4[j]\*indep[k][i]

        k+=1

y\_fit=[x.round(4) for x in y\_fit]

print("Fitted values are")

print(y\_fit)

print()

sse=0

for i in range(len(y\_fit)):

    sse+=(y[i]-(y\_fit[i]))\*\*2

sse=sse.round(4)

print(f"The sum of squares due to error is {sse}")

y\_mean=np.mean(y).round(4)

sst=0

for i in range(len(y\_fit)):

    sst+=(y[i]-y\_mean)\*\*2

sst=sst.round(4)

print(f"The sum of squares due to total is {sst}")

ssr=sst-sse

print(f"The sum of squares due to total is {ssr}")

print()

r\_square=(ssr/sst).round(4)

print(f"The R^2 value is {r\_square}")

print()

print("By using R^2 test")

if(r\_square>=0.90):

    print("We accept the model")

else:

    print("We reject the model")

print()

m\_ssr=(ssr/no\_ind).round(4)

m\_sse=(sse/(len(y\_fit)-no\_ind-1))

f\_cal=(m\_ssr/m\_sse).round(4)

dfe=len(y\_fit)-no\_ind-1

print(f"The calucalted value of F is {f\_cal}")

f\_tab=stats.f.ppf(1-alpha,no\_ind,dfe).round(4)

print(f"The table value of F is {f\_tab}")

print()

print("By using ANOVA test")

if(f\_cal>f\_tab):

    print("We accept the model")

else:

    print("We reject the model")

print()

param\_value=[]

for i in range(no\_ind+1):

    temp=(s4[i]/((m\_sse\*s2[i][i])\*\*0.5)).round(4)

    param\_value.append(temp)

print("t values parameters are :")

print(param\_value)

t\_table=stats.t.ppf(1-alpha/2,len(y\_fit)-no\_ind-1)

print(f"The table value of t : {t\_table}")

i=0

for a in param\_value:

    if(abs(a))>t\_table:

        print("Beta"+str(i)+" is contributing to the model")

    else:

        print("Beta"+str(i)+" is not contributing to the model")

    i+=1

**Output 1:-**

Enter no of independent variables2

Enter the values for independent variable 1

9 8 7 14 12 10 7 4 6 5 7 6

Enter the values for independent variable 2

62 58 64 60 63 57 55 56 59 61 57 60

Enter the values of output

100 110 105 94 95 99 104 108 105 98 105 110

Enter level of significance.05

The required multi linear regression is y=133.4605+-1.2485x1+-0.351x2

Fitted values are

[100.462, 103.1145, 102.257, 94.9215, 96.3655, 100.9685, 105.416, 108.8105, 105.2605, 105.807, 104.714, 104.9095]

The sum of squares due to error is 151.4104

The sum of squares due to total is 330.25

The sum of squares due to total is 178.8396

The R^2 value is 0.5415

By using R^2 test

We reject the model

The calucalted value of F is 5.3152

The table value of F is 4.2565

By using ANOVA test

We accept the model

t values parameters are :

[5.0882, -2.8079, -0.7711]

The table value of t : 2.2621571627409915

Beta0 is contributing to the model

Beta1 is contributing to the model

Beta2 is not contributing to the model

**Output 2:-**

Enter no of independent variables2

Enter the values for independent variable 1

-5 -4 -1 2 2 3 3

Enter the values for independent variable 2

5 4 1 -3 -2 -2 -3

Enter the values of output

11 11 8 2 5 5 4

Enter level of significance0.05

The required multi linear regression is y=6.5714+1.0x1+2.0x2

Fitted values are

[11.5714, 10.5714, 7.5714, 2.5714, 4.5714, 5.5714, 3.5714]

The sum of squares due to error is 1.7143

The sum of squares due to total is 73.7143

The sum of squares due to total is 72.0

The R^2 value is 0.9767

By using R^2 test

We accept the model

The calucalted value of F is 83.9993

The table value of F is 6.9443

By using ANOVA test

We accept the model

t values parameters are :

[26.5579, 2.1523, 4.3046]

The table value of t : 2.7764451051977987

Beta0 is contributing to the model

Beta1 is not contributing to the model

Beta2 is contributing to the model

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

**Source code:-**

import numpy as np

q=int(input('Enter no. of dependent variables:'))

yt=[]

for i in range(q):

  print('Enter y'+str(i+1),'data')

  yt.append([float(j) for j in input().split()])

n=len(yt[0])

p=int(input('Enter no. of independent variables:'))

xt=[]

xt.append([1]\*n)

for i in range(p):

  print('Enter x'+str(i+1),'data')

  xt.append([float(j) for j in input().split()])

p=p+1

g=float(input('Enter level of  significance: '))

def transpose(m):

    return [[m[j][i] for j in range(len(m))] for i in range(len(m[0]))]

def getMinor(m,i,j):

  return [p[:j]+p[j+1:] for p in m[:i]+m[i+1:]]

def getDeterminant(a):

  if len(a)==1:

    return a[0][0]

  if len(a)==2:

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

  det=0

  for c in range(len(a)):

    det+=(-1)\*\*c\*a[0][c]\*getDeterminant(getMinor(a,0,c))

  return det

def coFactor(m):

  return [[(-1)\*\*(i+j)\*getDeterminant(getMinor(m,i,j))for j in range(len(m))] for i in range(len(m))]

def Inverse(m):

  det=getDeterminant(m)

  if det==0: print('det=0,Inverse not posssible');exit()

  adj=transpose(coFactor(m))

  return [[adj[i][j]/det for j in range(len(adj[0]))] for i in range(len(adj))]

def Multiply(a,b):

    r1,c1,r2,c2=len(a),len(a[0]),len(b),len(b[0])

    if c1!=r2: print('Multiplication not possible');exit()

    mm=[]

    for i in range(0,r1):

        c=[]

        for j in range(0,c2):

            p=0

            for k in range(0,c1):

                p=p+(a[i][k]\*b[k][j])

            c.append(p)

        mm.append(c)

    return mm

x=transpose(xt)

y=transpose(yt)

xtxi=Inverse(Multiply(xt,x))

xty=Multiply(xt,y)

b=Multiply(xtxi,xty)

yt=np.array(yt)

xt=np.array(xt)

for i in range(q):

  print('\nFOR THE EQUATION y'+str(i+1),':')

  ye='y'+str(i+1)+' = '

  ye=ye+str('%.3f'%b[0][i])+' + '

  y1=np.array(b[0][i]\*xt[0])

  f=0

  for j in range(1,p):

      ye=ye+str('(%.3f)'%b[j][i])+'x'+str(j);y1=y1+(b[j][i]\*xt[j])

      if j!=p-1:ye=ye+' + '

  print('Model fit equation is ',ye)

  sse=np.sum((yt[i]-y1)\*\*2)

  sst=np.sum((yt[i]-np.mean(yt[i]))\*\*2)

  ssr=sst-sse

  print('\nUsing Coefficient of Determination Method: ')

  r2=ssr/sst

  print('R^2 = %.5f'%(r2))

  if(r2>0.90):print('Model fits good (R^2>0.90)')

  else:print('Model does not fit good (R^2<=0.90)')

  print('\nUSing ANOVA Test:')

  mssr=ssr/(p-1)

  msse=sse/(n-p)

  f=mssr/msse

  import scipy.stats as s

  if f>=1: ft=s.f.ppf(1-g,p-1,n-p)

  else: f=1/f;ft=s.f.ppf(1-g,n-p,p-1)

  print('Calculated value of f is %.5f'%(f))

  print('Table value of f is %.5f'%(ft))

  if f>ft: print('Model fits good')

  else: print('Model does not fit good')

  print('\nTest of Individual Parameters:')

  cjj=[xtxi[j][j] for j in range(len(xtxi))]

  tt=s.t.ppf(1-g/2,n-p)

  print('Table value of t is %.5f'%(tt))

  for j in range(p):

      e=np.sqrt(msse\*cjj[j])

      t=b[j][i]/e

      q='b['+str(j)+']'

      print('t(cal) for',q,'is %.5f'%(t))

      if abs(t)>tt: print(q,'is contributing to the model')

      else: print(q,'is not contributing to the model')

**Output 1:-**

Enter no. of dependent variables:2

Enter y1 data

10 12 11 9 9 10 11 12 11 10 11 12

Enter y2 data

100 110 105 94 95 99 104 108 105 98 103 110

Enter no. of independent variables:3

Enter x1 data

9 8 7 14 12 10 7 4 6 5 7 6

Enter x2 data

62 58 64 60 63 57 55 56 59 61 57 60

Enter x3 data

1 1.3 1.2 0.8 0.8 0.9 1 1.2 1.1 1.0 1.2 1.2

Enter level of significance: 0.05

FOR THE EQUATION y1 :

Model fit equation is y1 = 10.897 + (-0.045)x1 + (-0.088)x2 + (5.035)x3

Using Coefficient of Determination Method:

R^2 = 0.92380

Model fits good (R^2>0.90)

USing ANOVA Test:

Calculated value of f is 32.32738

Table value of f is 4.06618

Model fits good

Test of Individual Parameters:

Table value of t is 2.30600

t(cal) for b[0] is 4.23735

b[0] is contributing to the model

t(cal) for b[1] is -0.82835

b[1] is not contributing to the model

t(cal) for b[2] is -2.27524

b[2] is not contributing to the model

t(cal) for b[3] is 5.46181

b[3] is contributing to the model

FOR THE EQUATION y2 :

Model fit equation is y2 = 91.097 + (-0.064)x1 + (-0.294)x2 + (27.835)x3

Using Coefficient of Determination Method:

R^2 = 0.86551

Model does not fit good (R^2<=0.90)

USing ANOVA Test:

Calculated value of f is 17.16130

Table value of f is 4.06618

Model fits good

Test of Individual Parameters:

Table value of t is 2.30600

t(cal) for b[0] is 5.26478

b[0] is contributing to the model

t(cal) for b[1] is -0.17534

b[1] is not contributing to the model

t(cal) for b[2] is -1.13500

b[2] is not contributing to the model

t(cal) for b[3] is 4.48725

b[3] is contributing to the model

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

**Source code:-**

import numpy as np

import scipy.stats as s

k = int(input("Enter no.of treatments : "))

p = int(input("Enter no.of observations in each subgroup : "))

T = []

n = []

alp = float(input('Enter level of significance : '))

for i in range(k):

    n.append(int(input("Enter no.of sub groups in treatment :")))

    t = []

    for j in range(n[i]):

        a = list(map(int, input("Enter rowwise: ").split()))

        a = np.matrix(a).reshape(p,1)

        t.append(a)

    T.append(t)

print()

for i in range(k):

    print()

    print("Treatment ",i+1)

    print(T[i])

print()

yi\_ = []

y\_\_ = 0

for i in range(k):

    yi\_.append(sum(T[i])/n[i])

    y\_\_ += sum(T[i])

y\_\_ = y\_\_/sum(n)

print("yi\_ = ",yi\_)

print()

print("y\_\_ = ",y\_\_)

print()

# For y1

ssey1 = 0

ssty1 = 0

for i in range(k):

    for j in range(n[i]):

        ssey1 = ssey1 + (float(T[i][j][0])-float(yi\_[i][0]))\*\*2

        ssty1 = ssty1 + (float(T[i][j][0])-float(y\_\_[0]))\*\*2

ssry1 = ssty1-ssey1

print("ssey1 = ",ssey1)

print("ssty1 = ",ssty1)

print("ssrt1 = ",ssry1)

print()

# For y2

ssey2 = 0

ssty2 = 0

for i in range(k):

    for j in range(n[i]):

        ssey2 = ssey2 + (float(T[i][j][1])-float(yi\_[i][1]))\*\*2

        ssty2 = ssty2 + (float(T[i][j][1])-float(y\_\_[1]))\*\*2

ssry2 = ssty2-ssey2

print("ssey2 = ",ssey2)

print("ssty2 = ",ssty2)

print("ssry2 = ",ssry2)

print()

# For y1 and y2

ssey12 = 0

ssty12 = 0

for i in range(k):

    for j in range(n[i]):

        ssey12 = ssey12 + float((T[i][j][0]\*T[i][j][1]) - (yi\_[i][0]\*yi\_[i][1]))

        ssty12 = ssty12 + float((T[i][j][0]\*T[i][j][1]) - (y\_\_[0]\*y\_\_[1]))

ssry12 = ssty12-ssey12

print("ssey12 = ",ssey12)

print("ssty12 = ",ssty12)

print("ssry12 = ",ssry12)

print()

B = np.matrix([[ssry1, ssry12],[ssry12, ssry2]])

W = np.matrix([[ssey1, ssey12],[ssey12, ssey2]])

T = np.matrix([[ssty1, ssty12],[ssty12, ssty2]])

print("B = ",B)

print()

print("W = ",W)

print()

print("T = ",T)

print()

def determinent(matrix):

    no\_rows=len(matrix)

    for row in matrix:

        if len(row)!=no\_rows:

            print("not a square matrix")

            return None

    if len(matrix)==2:

        simple\_det=(matrix[0][0]\*matrix[1][1])-(matrix[1][0]\*matrix[0][1])

        return simple\_det

    else:

        #cofactor expression

        answer=0

        no\_columns=no\_rows

        for j in range(no\_columns):

            cofactor=((-1) \*\* (0+j) \* matrix[0][j]) \* (determinent(smallermatrix(matrix,0,j)))

            answer+=cofactor

        return answer

det\_w = determinent(np.array(W))

det\_t = determinent(np.array(T))

LAMBDA = float(det\_w/det\_t)

print("LAMBDA\*=",LAMBDA)

F = ((sum(n)-k-1)/(k-1))\*((1-(LAMBDA)\*\*.5)/(LAMBDA\*\*.5))

alp = .05

cou = 0

if F<1:

    cou=1

    F=1/F

print("F cal=",F)

if cou==1:

    Ft=s.f.ppf(1-alp,p\*(sum(n)-k-1),p\*(k-1))

else:

    Ft=s.f.ppf(1-alp,p\*(k-1),p\*(sum(n)-k-1))

print("F t.v=",Ft)

if F>Ft:

    print("We reject Ho")

    print("There is no homoginity among the table")

    print("We reject the model")

else:

    print("We accept Ho")

    print("There is homoginity among the table")

    print("We accept the model")

**Output 1:-**

Enter no.of treatments : 3

Enter no.of observations in each subgroup : 2

Enter level of significance : 0.05

Enter no.of sub groups in treatment :4

Enter rowwise: 2 3

Enter rowwise: 3 4

Enter rowwise: 5 4

Enter rowwise: 2 5

Enter no.of sub groups in treatment :3

Enter rowwise: 4 8

Enter rowwise: 5 6

Enter rowwise: 6 7

Enter no.of sub groups in treatment :5

Enter rowwise: 7 6

Enter rowwise: 8 7

Enter rowwise: 10 8

Enter rowwise: 9 5

Enter rowwise: 7 6

Treatment 1

[matrix([[2],

[3]]), matrix([[3],

[4]]), matrix([[5],

[4]]), matrix([[2],

[5]])]

Treatment 2

[matrix([[4],

[8]]), matrix([[5],

[6]]), matrix([[6],

[7]])]

Treatment 3

[matrix([[7],

[6]]), matrix([[8],

[7]]), matrix([[10],

[ 8]]), matrix([[9],

[5]]), matrix([[7],

[6]])]

yi\_ = [matrix([[3.],

[4.]]), matrix([[5.],

[7.]]), matrix([[8.2],

[6.4]])]

y\_\_ = [[5.66666667]

[5.75 ]]

ssey1 = 14.799999999999997

ssty1 = 76.66666666666667

ssrt1 = 61.866666666666674

ssey2 = 9.2

ssty2 = 28.25

ssry2 = 19.05

ssey12 = 1.6000000000000156

ssty12 = 25.999999999999943

ssry12 = 24.399999999999928

B = [[61.86666667 24.4 ]

[24.4 19.05 ]]

W = [[14.8 1.6]

[ 1.6 9.2]]

T = [[76.66666667 26. ]

[26. 28.25 ]]

LAMBDA\*= 0.08967446023045059

F cal= 9.357513005519227

F t.v= 3.0069172799243438

We reject Ho

There is no homoginity among the table

We reject the model

**Output 2:-**

Enter no.of treatments : 3

Enter no.of observations in each subgroup : 2

Enter level of significance : 0.05

Enter no.of sub groups in treatment :3

Enter rowwise: 9 3

Enter rowwise: 6 2

Enter rowwise: 9 7

Enter no.of sub groups in treatment :2

Enter rowwise: 0 4

Enter rowwise: 2 0

Enter no.of sub groups in treatment :3

Enter rowwise: 3 8

Enter rowwise: 1 9

Enter rowwise: 2 7

Treatment 1

[matrix([[9],

[3]]), matrix([[6],

[2]]), matrix([[9],

[7]])]

Treatment 2

[matrix([[0],

[4]]), matrix([[2],

[0]])]

Treatment 3

[matrix([[3],

[8]]), matrix([[1],

[9]]), matrix([[2],

[7]])]

yi\_ = [matrix([[8.],

[4.]]), matrix([[1.],

[2.]]), matrix([[2.],

[8.]])]

y\_\_ = [[4.]

[5.]]

ssey1 = 10.0

ssty1 = 88.0

ssrt1 = 78.0

ssey2 = 24.0

ssty2 = 72.0

ssry2 = 48.0

ssey12 = 1.0

ssty12 = -11.0

ssry12 = -12.0

B = [[ 78. -12.]

[-12. 48.]]

W = [[10. 1.]

[ 1. 24.]]

T = [[ 88. -11.]

[-11. 72.]]

LAMBDA\*= 0.03845534995977474

F cal= 8.198859563778374

F t.v= 3.837853354555897

We reject Ho

There is no homoginity among the table

We reject the model

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

**Source code:-**

import numpy as np

import math

#l1=[2.95,2.53,3.57,3.16,2.58,2.16,3.27]

#l2=[6.63,7.79,5.65,5.47,4.46,6.22,3.52]

k=[[2.81,5.46]]

l1=[1300,1260,1220,1180,1060,1140,1100,1020,980,940]

l2=[2.7,3.7,2.9,2.5,3.9,2.1,3.5,3.3,2.3,3.1]

x=[]

for i in range(len(l1)):

  x.append([l1[i],l2[i]])

print("x=",x)

x1=[]

x2=[]

#y=[1,1,1,1,0,0,0]

y=[1,1,1,1,1,0,0,0,0,0]

for i in range(len(y)):

  if(y[i]==y[0]):

    x1.append([l1[i],l2[i]])

  else:

    x2.append([l1[i],l2[i]])

print("x1=",x1)

print("x2=",x2)

mu=[]

sum1=0

sum2=0

for i in range(len(l1)):

  sum1=sum1+l1[i]

  sum2=sum2+l2[i]

mu.append([(sum1/len(l1)),(sum2/len(l2))])

print("Mean of x=",mu)

mu1=[]

sum3=0

sum4=0

for i in range(len(x1)):

  sum3=sum3+x1[i][0]

  sum4=sum4+x1[i][1]

mu1.append([(sum3/len(x1)),(sum4/len(x1))])

print("Mean of x1=",mu1)

mu2=[]

sum5=0

sum6=0

for i in range(len(x2)):

  sum5=sum5+x2[i][0]

  sum6=sum6+x2[i][1]

mu2.append([(sum5/len(x2)),(sum6/len(x2))])

print("Mean of x2=",mu2)

def transpose(x):

    xt=[[0 for i in range(len(x))] for j in range(len(x[0]))]

    for i in range(len(x)):

        for j in range(len(x[0])):

             xt[j][i]=x[i][j]

    return xt

def mul(x,y):

    res =  [[0 for i in range(len(y[0]))] for j in range(len(x))]

    for i in range(len(x)):

        for j in range(len(y[0])):

            for k in range(len(y)):

                res[i][j]+=x[i][k]\*y[k][j]

    return res

a=[]

for i in range(len(x)):

  a.append([x[i][0]-mu[0][0],x[i][1]-mu[0][1]])

print("x-mu=",a)

for i in range(len(a)):

  c=mul(transpose(a),a)

for i in range(len(c)):

  for j in range(len(a[0])):

    c[i][j]=c[i][j]/len(a)

print("c=",c)

cin=np.linalg.inv(c)

print("c inverse=",cin)

f1=0

f2=0

f1=mul(mul(mu1,cin),transpose(k))[0][0]-((mul(mul(mu1,cin),transpose(mu1))[0][0])\*0.5)+math.log(len(x1)/len(x))

print("F1=",f1)

f2=mul(mul(mu2,cin),transpose(k))[0][0]-((mul(mul(mu2,cin),transpose(mu2))[0][0])\*0.5)+math.log(len(x2)/len(x))

print("F2=",f2)

if(f1>f2):

  print('The new observation is kept into first category i.e,passed category')

else:

  print('The new observation is kept into second category')

**Output 1:-**

x= [[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]]

x1= [[2.95, 6.63], [2.53, 7.79], [3.57, 5.65], [3.16, 5.47]]

x2= [[2.58, 4.46], [2.16, 6.22], [3.27, 3.52]]

Mean of x= [[2.888571428571429, 5.677142857142857]]

Mean of x1= [[3.0525, 6.385]]

Mean of x2= [[2.67, 4.733333333333333]]

x-mu= [[0.06142857142857139, 0.9528571428571428], [-0.358571428571429, 2.112857142857143], [0.681428571428571, -0.02714285714285669], [0.27142857142857135, -0.2071428571428573], [-0.3085714285714287, -1.217142857142857], [-0.7285714285714286, 0.5428571428571427], [0.38142857142857123, -2.157142857142857]]

c= [[0.2059836734693877, -0.23093265306122449], [-0.23093265306122449, 1.6921632653061225]]

c inverse= [[5.73171449 0.78221769]

[0.78221769 0.69771022]]

F1= 43.82818099216788

F2= 43.86302017821443

The new observation is kept into second category

**Output 2:-**

x= [[1300, 2.7], [1260, 3.7], [1220, 2.9], [1180, 2.5], [1060, 3.9], [1140, 2.1], [1100, 3.5], [1020, 3.3], [980, 2.3], [940, 3.1]]

x1= [[1300, 2.7], [1260, 3.7], [1220, 2.9], [1180, 2.5], [1060, 3.9]]

x2= [[1140, 2.1], [1100, 3.5], [1020, 3.3], [980, 2.3], [940, 3.1]]

Mean of x= [[1120.0, 3.0000000000000004]]

Mean of x1= [[1204.0, 3.14]]

Mean of x2= [[1036.0, 2.86]]

x-mu= [[180.0, -0.30000000000000027], [140.0, 0.6999999999999997], [100.0, -0.10000000000000053], [60.0, -0.5000000000000004], [-60.0, 0.8999999999999995], [20.0, -0.9000000000000004], [-20.0, 0.49999999999999956], [-100.0, 0.2999999999999994], [-140.0, -0.7000000000000006], [-180.0, 0.09999999999999964]]

c= [[13200.0, -2.799999999999991], [-2.799999999999991, 0.32999999999999996]]

c inverse= [[7.58941713e-05 6.43950545e-04]

[6.43950545e-04 3.03576685e+00]]

F1= -16.559987701607927

F2= -4.471300403992386

The new observation is kept into 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

import scipy.stats as st

import copy

xn=int(input('Enter number of components='))

x=[]

n=int(input("Enter no of observations="))

for i in range(xn):

    l=list(map(float,input().split()))

    x.append(l)

x\_copy=np.transpose(x)

print("x=",x\_copy)

def mu\_cal(x):

  n=len(x)

  l=len(x[0])

  mu=[]

  for i in range(l):

    s=0

    for j in range(n):

      s+=x[j][i]

    s=s/n

    mu.append(s)

  return mu

mu\_x=(mu\_cal(x\_copy))

print("Mean of x=",mu\_x)

xsubm=np.subtract(x\_copy,mu\_x)

print("\nx-mu=",xsubm)

#calculating pooled variance c\_inverse

pooled\_matrix=(np.dot(np.transpose(xsubm),xsubm))/n

print("pooled matrix=",pooled\_matrix)

# pooled\_matrix\_inv=np.linalg.inv(pooled\_matrix)

# print("pooled matrix inverse is:",pooled\_matrix\_inv)

eig\_value,eig\_vector=np.linalg.eig(pooled\_matrix)

eig\_value=np.sort(eig\_value)[::-1]

print("Eigen values=",eig\_value)

df=pd.DataFrame()

df['pricipal components']=[i+1 for i in range(len(eig\_value))]

df['variance explained']=[i for i in eig\_value]

sum=0

tot\_sum=np.sum(eig\_value)

l=[]

for i in eig\_value:

  sum+=i

  l.append((sum/tot\_sum)\*100)

df['cumulative tot var']=l

print(df)

thres\_value=float(input("Enter The Threshold Value For Principal Component Analysis="))

l=0

#del df['cumulative tot var']

for i in df['cumulative tot var']:

  if i>=thres\_value:

    break

  else:

    l+=1

df=df.loc[0:l]

print(df)

eig\_values=list(df['variance explained'])

print("Eigen values taken for to retain the principal components[based on the thershold is fixed]=",eig\_values)

co\_related\_coeff=[]

for m in eig\_values:

  l=copy.copy(pooled\_matrix)

  for i in range(len(l)):

    for j in range(len(l[0])):

      if i==j:

        l[i][j]-=m

  co\_related\_coeff.append(l)

co\_related\_coeff

def normalization(x):

  if len(x)==2:

    total=(x[0][0])\*\*2+(x[0][1])\*\*2

    total=total\*\*0.5

   # print(total)

   # l=[]

    l=[-x[0][1]/total,x[0][0]/total]

   # l=list(-x[0][1]/total,x[0][0]/total)

    return l

  else:

    l=[]

    copy1=list(copy.copy(x[0]))

    copy2=list(copy.copy(x[1]))

    copy1.append(x[0][0])

    copy1.append(x[0][1])

    copy2.append(x[1][0])

    copy2.append(x[1][1])

    for i in range(1,len(copy1)-1):

      l.append(copy1[i]\*copy2[i+1]-copy1[i+1]\*copy2[i])

    sum=0

    for i in range(len(l)):

      sum+=l[i]\*\*2

    sum=(sum\*\*0.5)\*\*-1

    l=[i\*sum for i in l]

    return l

z=[normalization(i) for i in co\_related\_coeff]

print("The eigen vectors of z1 and z2 are:",z)

print("The Principal component equation are:")

for i in range(len(z)):

  print("\nZ",i+1,"=",end='')

  for j in range(len(z[0])):

    if j==len(z[0])-1:

      print(z[i][j],'X',j+1)

    else:

      print(z[i][j],"X",j+1,"+",end='')

corelated\_data=[]

for i in z:

  l=[]

  for j in range(len(x\_copy)):

    sum=0

    for k in range(len(x\_copy[j])):

      sum+=x\_copy[j][k]\*i[k]

    l.append(sum)

  corelated\_data.append(l)

corelated\_data

corelated\_data\_df=pd.DataFrame()

for i in range(len(corelated\_data)):

  corelated\_data\_df[f'Z{i+1}']=corelated\_data[i]

corelated\_data\_df

**Output 1:-**

Enter number of components=2

Enter no of observations=4

2 1 0 -1

4 3 1 0.5

x= [[ 2. 4. ]

[ 1. 3. ]

[ 0. 1. ]

[-1. 0.5]]

Mean of x= [0.5, 2.125]

x-mu= [[ 1.5 1.875]

[ 0.5 0.875]

[-0.5 -1.125]

[-1.5 -1.625]]

pooled matrix= [[1.25 1.5625 ]

[1.5625 2.046875]]

Eigen values= [3.26093826 0.03593674]

pricipal components variance explained cumulative tot var

0 1 3.260938 98.909976

1 2 0.035937 100.000000

Enter The Threshold Value For Principal Component Analysis=99

pricipal components variance explained cumulative tot var

0 1 3.260938 98.909976

1 2 0.035937 100.000000

Eigen values taken for to retain the principal components[based on the thershold is fixed]= [3.2609382570250163, 0.03593674297498417]

The eigen vectors of z1 and z2 are: [[-0.6135581035026774, -0.789649576474399], [-0.7896495764743992, 0.6135581035026774]]

The Principal component equation are:

Z 1 =-0.6135581035026774 X 1 +-0.789649576474399 X 2

Z 2 =-0.7896495764743992 X 1 +0.6135581035026774 X 2

|  | **Z1** | **Z2** |
| --- | --- | --- |
| **0** | -4.385715 | 0.874933 |
| **1** | -2.982507 | 1.051025 |
| **2** | -0.789650 | 0.613558 |
| **3** | 0.218733 | 1.096429 |

**Output 2:-**

Enter number of components=3

Enter no of observations=5

90 90 60 60 30

60 90 60 60 30

90 30 60 90 30

x= [[90. 60. 90.]

[90. 90. 30.]

[60. 60. 60.]

[60. 60. 90.]

[30. 30. 30.]]

Mean of x= [66.0, 60.0, 60.0]

x-mu= [[ 24. 0. 30.]

[ 24. 30. -30.]

[ -6. 0. 0.]

[ -6. 0. 30.]

[-36. -30. -30.]]

pooled matrix= [[504. 360. 180.]

[360. 360. 0.]

[180. 0. 720.]]

Eigen values= [910.06995304 629.11038668 44.81966028]

pricipal components variance explained cumulative tot var

0 1 910.069953 57.453911

1 2 629.110387 97.170476

2 3 44.819660 100.000000

Enter The Threshold Value For Principal Component Analysis=95

pricipal components variance explained cumulative tot var

0 1 910.069953 57.453911

1 2 629.110387 97.170476

Eigen values taken for to retain the principal components[based on the thershold is fixed]= [910.0699530410365, 629.1103866763252]

The eigen vectors of z1 and z2 are: [[0.6558022549801461, 0.42919779654868706, 0.6210576895914801], [0.38599879538810006, 0.5163664177215531, -0.7644413990675454]]

The Principal component equation are:

Z 1 =0.6558022549801461 X 1 +0.42919779654868706 X 2 +0.6210576895914801 X 3

Z 2 =0.38599879538810006 X 1 +0.5163664177215531 X 2 +-0.7644413990675454 X 3

|  | **Z1** | **Z2** |
| --- | --- | --- |
| **0** | 140.669263 | -3.077849 |
| **1** | 116.281735 | 58.279627 |
| **2** | 102.363464 | 8.275429 |
| **3** | 120.995195 | -14.657813 |
| **4** | 51.181732 | 4.137714 |

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

**Source code:-**

import numpy as np

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)

print(“Variance:”)

sumh=sum(h)

print(sumh)

pve=[]

for i in range(len(eigenValues)):

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

print(pve)

**Output 1:-**

enter no of variables3

3 7 10 3 10

6 3 9 9 6

5 3 8 7 5

[[3, 7, 10, 3, 10], [6, 3, 9, 9, 6], [5, 3, 8, 7, 5]]

[6.6, 6.6, 5.6]

[3.5071, 2.51, 1.9494]

[array([-1.02648912, 0.11405435, 0.96946195, -1.02648912, 0.96946195]), array([-0.23904382, -1.43426295, 0.9561753 , 0.9561753 , -0.23904382]), array([-0.30778701, -1.33374372, 1.23114805, 0.71816969, -0.30778701])]

Variance Covariance Matrix is :

[[ 0.80001623 -0.04089598 0.06435815]

[-0.04089598 0.7999873 0.78479557]

[ 0.06435815 0.78479557 0.79996624]]

Eigen Values are [1.58512447 0.80666111 0.0081842 ]

Eigen Vectors are [[ 0.02122114 -0.99538368 -0.09360014]

[ 0.70624057 0.0811911 -0.70330098]

[ 0.70765382 -0.05117936 0.7047033 ]]

enter threshold limit :99

Threshold Table :

[66.04768471804098, 99.65898746868129, 100.0]

Retained Eigen Values :

[1.58512447 0.80666111]

Retained Eigen Vectors:

[[ 0.02122114 -0.99538368]

[ 0.70624057 0.0811911 ]

[ 0.70765382 -0.05117936]]

[[0.026717780654088277, 0.8891690658976457, 0.8909483734282817], [-0.8939970315309378, 0.07292122996398519, -0.04596639553844041]]

[-1.734558501753699, 1.9241805917232617, 1.6899639557796826]

Variance:

1.8795860457492453

[84.33370077691795, 42.916955545008435]