Practical File

Submitted by: Divija

102018056

2020-2024

COmputational Statistics

***LAB-BASIC STATISTICS***

import numpy as np

import pandas as pd

from bisect import bisect\_right as upper\_bound

import statistics

import math

#random.randit is used to generate random numbers in a particular range

data = np.random.randint(0, 100, size=(5, 3))

#pd.dataframe prints data in tabular form ,thus easy to access

df = pd.DataFrame(data, columns=['random\_1', 'random\_2', 'random\_3'])

print(df)

#iloc helps to extract rows and col to make x array

x=df.iloc[:,:].values

#.shape gives the dimensions of the dataframe

print(x.shape)

n=x.shape[0]

m=x.shape[1]

#for harmonic mean

sumh=0

for i in range(n):

sumh=sumh+1/x[i][0]

hm=(n)/sumh

print('harmonic mean by formula is :', round(hm, 2))

print('harmonic mean by inbuilt function is :' ,round(statistics.harmonic\_mean(df.random\_1), 2))

#for geometric mean

product=1

for i in range(n):

product=product \* x[i][0]

gm = (float)(math.pow(product, (1 / n)))

print('geometeric mean by formula is :', round(gm, 2))

print('geometeric mean by inbuilt function is :' ,round(statistics.geometric\_mean(df.random\_1), 2))

#for mean of data

sum =0

for i in range(n):

for j in range(m):

sum=sum+x[i][j]

me=sum/(n\*m)

print('mean by formula is :', round(me, 2))

print('mean by inbuilt function is :' ,round(np.mean(x), 2))

#for median of data

#first we have to sort the data

a=np.array(x)

def sortRowWise(a):

# One by one sort individual rows.

for i in range(len(a)):

a[i].sort()

return 0

MAX = 100;

# Function to find median in the matrix

def median\_using\_binary(a, n, m):

mi = a[0][0]

mx = 0

for i in range(n):

#finding min

if a[i][0] < mi:

mi = a[i][0]

#finding max

if a[i][m-1] > mx :

mx = a[i][m-1]

#stores the value of median if matrix is put in array format

desired = (n \* m + 1) // 2

while (mi < mx):

mid = mi + (mx - mi) // 2

#place stores the number of elements less than mid in the matrix

place = [0];

# Find count of elements smaller than mid

for i in range(n):

j = upper\_bound(a[i], mid)

place[0] = place[0] + j

if place[0] < desired:

mi = mid + 1

else:

mx = mid

return mi

#calling of functions

sortRowWise(a)

mi=median\_using\_binary(a, n, m)

print ("Median by formula is", mi)

print('median by inbuilt function is :' ,round(np.median(x), 2))

var=0

for i in range(n):

for j in range(m):

var=var+(x[i][j]-sum/(n\*m))\*\*2

#for variance

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

def variance(a, n, m, me):

sum1 = 0;

for i in range(n):

for j in range(m):

# subtracting mean from elements

a[i][j] -= me;

# squaring each terms

a[i][j] \*= a[i][j];

# taking sum

for i in range(n):

for j in range(m):

sum1 += a[i][j];

return sum1/(n\*m)

variance(x,n,m,me)

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

def covariance\_data():

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

csum=[]

sum=0

for i in range(data.shape[1]):

for j in range(data.shape[0]):

sum+=data[j][i]

csum.append(sum)

sum=0

cmean=[]

for i in csum:

mean=0

mean=i/data.shape[0]

cmean.append(mean)

cov\_mat=np.empty((data.shape[0],data.shape[1]))

cov=0

for i in range (data.shape[1]):

for j in range (data.shape[1]):

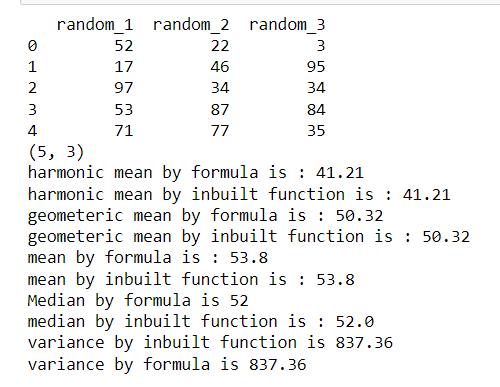
for k in range (data.shape[0]):

cov+=(data[k][i]-cmean[i])\*(data[k][j]-cmean[j])

cov=cov/(data.shape[0]-1)

cov\_mat[i][j]=cov

print(cov\_mat)

******

***LAB-GRAPH PLOTTING***

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

df = pd.read\_csv(r'C:\Users\hp\Desktop\Iris.csv')

print(df.head(5))

def simpleline():

#straight line graph

plt.title('Linear graph')

x = np.linspace(0, 10, 50)

y=np.linspace(0, 5, 50)

plt.plot(x,"o",y,"r")

plt.xlabel('x-axis')

plt.ylabel('Y-axis')

plt.show()

def trigno():

x= np.linspace(0,10,50)

y= np.linspace(0,5,50)

sin\_x = np.sin(x)

cos\_y =np.cos(y)

z=sin\_x + cos\_y

w=sin\_x\*\*2

plt.title('Trignometry')

plt.plot(z,label ='z=sin\_x+cos\_y',color ='green',linestyle ='-')

plt.plot(sin\_x, label='sinwave', color ='red', linestyle = '--')

plt.plot(cos\_y, label='coswave', color ='blue', linestyle = '-')

plt.plot(w, label='sin\_x\*\*2', color ='yellow', linestyle = '-')

plt.legend()

plt.show()

def subplots():

names = ['group\_1', 'group\_2', 'group\_3']

values = [10, 50, 200]

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

#bar-graph

# 1x3 grid for 1st plot

plt.subplot(131)

plt.bar(names, values)

#scatter plot

plt.subplot(132)

plt.scatter(names, values,c=['red'])

#normal plot

plt.subplot(133)

plt.plot(names, values,"g",)

plt.suptitle('Categorical Plotting')

plt.show()

def symbols():

#plot using diff symbols

#arrange takes values start,stop and step

t = np.arange(0, 5, 0.2)

plt.plot(t, t, 'g--', label='linear' )

plt.plot(t, t\*\*2, 'rs',label='square')

plt.plot(t, t\*\*3, 'b^',label='cube')

plt.legend()

plt.show()

def random\_data():

data1 = np.random.randn(2, 10)

print(data1)

print(data1.shape)

plt.plot(data1)

plt.show()

def pie\_chart():

species = ['SETOSA', 'VERSICOlOR', 'VIRGINIA']

data = [50,50,50]

# Creating plot

fig = plt.figure(figsize =(10, 7))

plt.pie(data, labels = species)

plt.title('Different species of Irirs')

# show plot

plt.show()

def scatterplot\_fileread():

print('Enter any two attributes of the data(SepalWidthCm,SepalLengthCm ,PetalLengthCm,PetalWidthCm )')

atr1=input()

atr2=input()

colors = {'Iris-setosa':'r', 'Iris-virginica':'g', 'Iris-versicolor':'y'}

if atr1 == 'SepalWidthCm' and atr2 == 'SepalLengthCm' :

plt.xlabel('SepalWidthCm')

plt.ylabel('SepalLengthCm')

plt.scatter(df.SepalWidthCm, df.SepalLengthCm, c=df['Species'].apply(lambda col\_vector: colors[col\_vector]), s = 100)

elif atr1 == 'SepalWidthCm' and atr2 == 'PetalLengthCm' :

plt.xlabel('SepalWidthCm')

plt.ylabel('PetalLengthCm')

plt.scatter(df.SepalWidthCm, df.PetalLengthCm, c=df['Species'].apply(lambda col\_vector: colors[col\_vector]), s = 100)

elif atr1 =='PetalWidthCm' and atr2 == 'PetalLengthCm' :

plt.xlabel('PetalWidthCm')

plt.ylabel('PetalLengthCm')

plt.scatter(df.PetalWidthCm, df.PetalLengthCm, c=df['Species'].apply(lambda col\_vector: colors[col\_vector]), s = 100)

elif atr1 == 'PetalWidthCm' and atr2 =='SepalLengthCm' :

plt.xlabel('PetalWidthCm')

plt.ylabel('SepalLengthCm')

plt.scatter(df.PetalWidthCm, df.SepalLengthCm, c=df['Species'].apply(lambda col\_vector: colors[col\_vector]), s = 100)

print("\n------------------------------------------------\n")

print('1 Simpleline')

print('2 Trigno\_functions')

print('3 Subplots')

print('4 Random\_data\_plot')

print('5 pie chart')

print('6 scatterplot\_fileread')

print("\n------------------------------------------------\n")

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

if x==1:

simpleline()

elif x==2:

trigno()

elif x==3:

subplots()

elif x==4:

random\_data()

elif x==5:

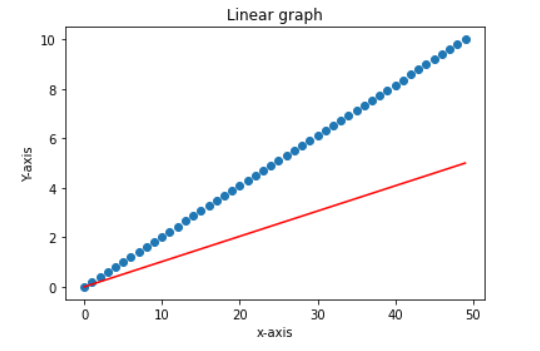
pie\_chart()

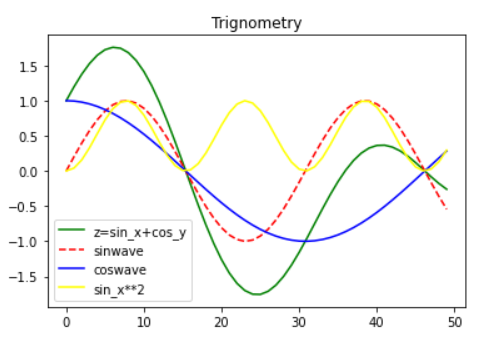
elif x==6:

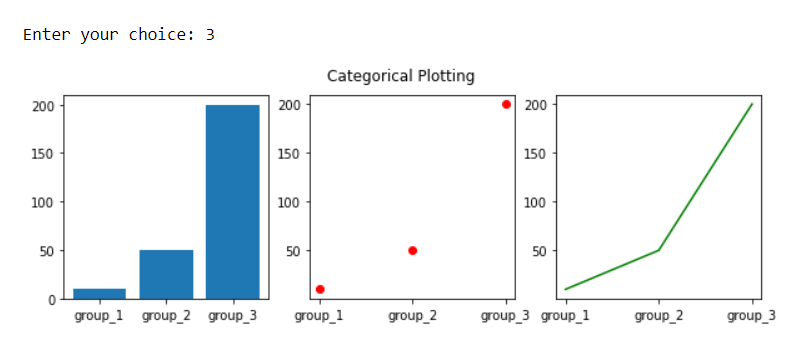
scatterplot\_fileread()

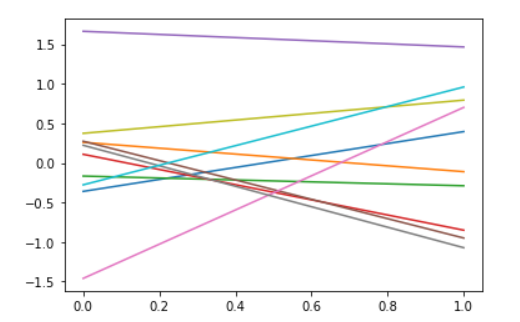
else:

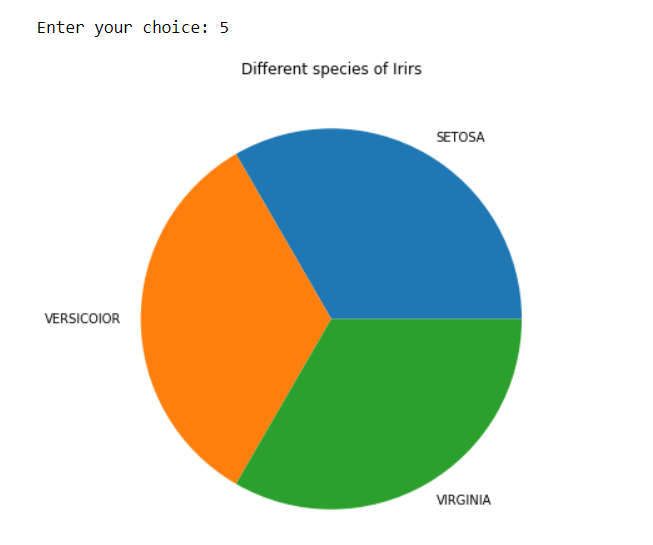
print("Please enter valid choice")

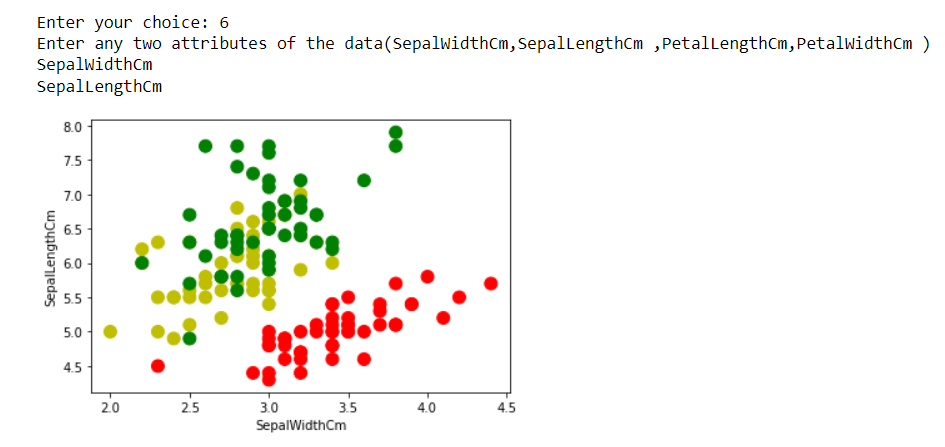












***LAB- LINEAR REGRESSION***

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

data = pd.read\_csv("headbrain.csv")

#head size is independent

#brain weight is dependent

#values() returns a list of all the values available in given data

x = data['Head Size(cm^3)'].values

y = data['Brain Weight(grams)'].values

print(data.head())

#237 input values

# total no.of input values

l = len(x)

sum\_x =0

sum\_y =0

# calculating the mean of x and y

for i in range(l):

    sum\_x += x[i]

mean\_x = sum\_x/l

for i in range(l):

    sum\_y += y[i]

mean\_y = sum\_y/l

# using the formula to calculate m & c

numer = 0

denom = 0

for i in range(l):

    numer += (x[i] - mean\_x) \* (y[i] - mean\_y)

    denom += (x[i] - mean\_x) \*\* 2

m = numer / denom

c = mean\_y - (m \* mean\_x)

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

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

#m is b1 and c is b0

# y=b0 +b1\*x

y\_reg=[]

y\_reg= c+(m\*x)

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

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

plt.plot(x,y\_reg ,color ="red" , label ="regression line")

plt.xlabel("Head Size")

plt.ylabel("Brain Weight")

plt.legend()

plt.title("Plot of linear Regression")

plt.show()

from sklearn.linear\_model import LinearRegression

Reg\_model = LinearRegression()

x=np.array(x)

y=np.array(y)

#array will get reshaped in such a way that the resulting array has only 1 column

#no reshaping,error while fitting it in the model.

x=x.reshape(-1,1)

y=y.reshape(-1,1)

#takes array as an input

Reg\_model.fit(x,y)

Reg\_model.coef\_

Reg\_model.intercept\_

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

ss\_t = 0 #total sum of squares

ss\_r = 0 #total sum of square of residuals

for i in range(len(x)):

    y\_pred = c + m \* x[i]

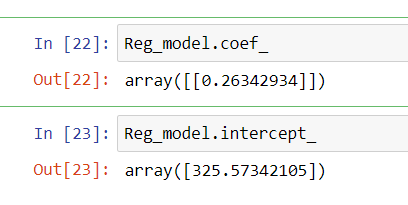
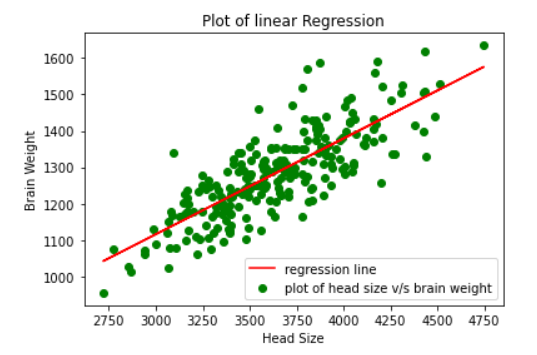
    ss\_t += (y[i] - mean\_y) \*\* 2

    ss\_r += (y[i] - y\_pred) \*\* 2

r2 = 1 - (ss\_r/ss\_t)

print(r2)





***LAB MULTICOLLINEARITY AND VIF***

import pandas as pd

import math

data=pd.read\_csv(r'C:\Users\hp\Desktop\cs\_data.csv')

data.head()

df=data.iloc[:,0:4]

print(df)

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

# print(data)

#using karl pearson method to calculate VIF

def calculate(x1,x2):

    n = len(x1)

    sum\_x1 = 0

    sum\_sq\_x1 = 0

    sum\_x2 = 0

    sum\_sq\_x2 = 0

    sum\_x1\_x2=0

    for i in range(n):

        sum\_x1 += x1[i]

        sum\_sq\_x1 +=(x1[i]\*x1[i])

        sum\_x2 += x2[i]

        sum\_sq\_x2 +=(x2[i]\*x2[i])

        sum\_x1\_x2 +=x1[i]\*x2[i]

    r=0

    num =0

    den= 0

    num = (n\*sum\_x1\_x2) - (sum\_x1\*sum\_x2)

    den = math.sqrt((n\*sum\_sq\_x1)-(sum\_x1\*\*2))\*math.sqrt((n\*sum\_sq\_x2)-(sum\_x2\*\*2))

    r=num/den

    r\_sq = r\*r

    print('r\_sq=', r\_sq)

    vif=1/(1-r\_sq)

    print('vif =' ,vif)

    for i in range(df.shape[1]):

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

            calculate(df.iloc[:,i],df.iloc[:,j])

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

from statsmodels.tools.tools import add\_constant

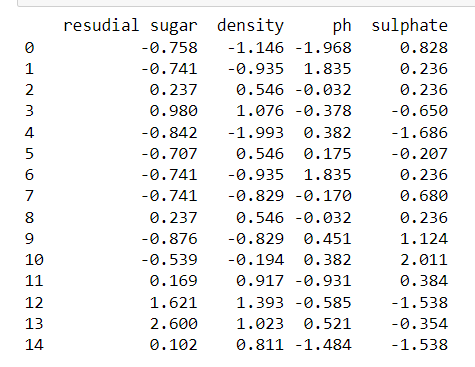
vif = pd.DataFrame()

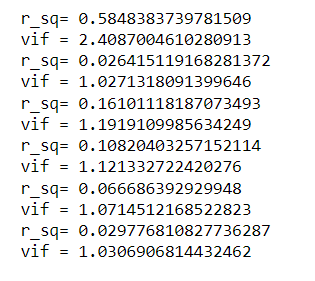
vif["features"] = df.columns

vif["VIF Factor"] = [variance\_inflation\_factor(df.values, i)

                     for i in range(df.shape[1])]

print(vif)





***LAB MULTIPLE LINEAR REGRESSION***

import pandas as pd

import math

import numpy as np

from statistics import stdev

data=pd.read\_csv('./bodyfat.csv')

data.head()

df=data.iloc[:,:]

print(df.head())

df\_normalized = df.copy()

for i in df\_normalized.columns:

    df\_normalized[i] = (df\_normalized[i] - df\_normalized[i].mean()) / df\_normalized[i].std()

df=df\_normalized

df1=df\_normalized.iloc[:,1:6]

X = df1.to\_numpy()

print(len(X))

ALL\_ONE=[1]\*len(X)

X1=np.insert(X,0,ALL\_ONE,axis=1)

print(X1)

y=df\_normalized.iloc[:,0]

Y=y.to\_numpy()

X1\_T = X1.T

print(X1\_T.shape[0])

X1\_mul = np.matmul(X1\_T,X1)

print(X1\_mul.shape[0])

print(X1\_mul.shape[1])

X\_inv = np.linalg.inv(X1\_mul)

print(X\_inv.shape[0])

print(X\_inv.shape[1])

X\_final = np.matmul(X\_inv, X1\_T)

b=np.matmul(X\_final,Y)

Y\_pred = []

for i in range(len(Y)):

    Y\_pred=b[0] + b[1]\*df\_normalized.iloc[:,1][i]+b[2]\*df\_normalized.iloc[:,2][i]+b[3]\*df\_normalized.iloc[:,3][i]+b[4]\*df\_normalized.iloc[:,4][i]+b[5]\*df\_normalized.iloc[:,5]

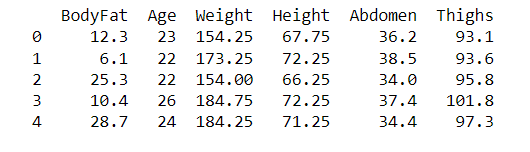
print(Y\_pred)

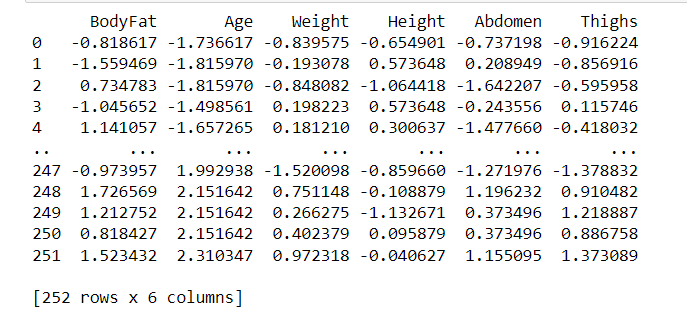
e=0

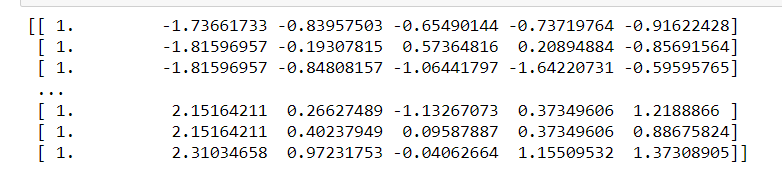
for i in range(len(Y)):

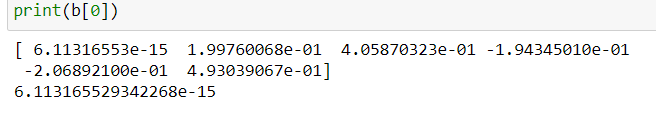
    e+=(Y[i]-Y\_pred[i])\*\*2

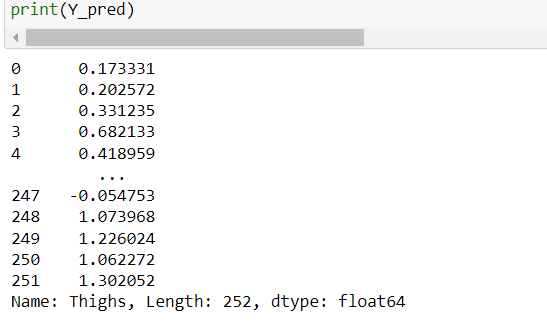
print(e/len(Y))











Error:

