```
#Importing Libs
import pandas as pd
import numpy as np

from scipy import stats

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

from sklearn.metrics import confusion_matrix

from scipy import signal
from scipy.fftpack import fft, fftshift
import matplotlib.pyplot as plt

import seaborn as sns

import math
```

```
#Reading Data a)InputData
data = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/0047
print(data)
```

```
     Age  Delivery Number  Delivery Time  Blood of Pressure  Heat Problem  \
0     22                1              0                  2             0
1     26                2              0                  1             0
2     26                2              1                  1             0
3     28                1              0                  2             0
4     22                2              0                  1             0
..   ...              ...            ...                ...           ...
75    27                2              1                  1             0
76    33                4              0                  1             0
77    29                2              1                  2             0
78    25                1              2                  0             0
79    24                2              2                  1             0

     Caesarian
0            0
1            1
2            0
3            0
4            1
..         ...
75           0
76           1
77           1
78           1
79           0

[80 rows x 6 columns]
```

```
#b) Statistics
Age = [i for i in data["Age"]]
DeliveryNumber = [i for i in data["Delivery Number"]]
```

```
DeliveryTime = [i for i in data["Delivery Time"]]
BloodPressure = [i for i in data["Blood of Pressure"]]
HeatProblem = [i for i in data["Heat Problem"]]
Caesarian = [i for i in data["Caesarian"]]

#Average
print("Statistics\n\nAverage\nAge :",np.average(Age),"\nDelivery Number :",np.avera
print("Delivery Time :",np.average(DeliveryTime),"\nBlood Pressure :",np.average(Bl
print("Heat Problem :",np.average(HeatProblem),"\nCaesarian :",np.average(Caesarian

#Mean
print("\n\nMean\nAge :",np.mean(Age),"\nDelivery Number :",np.mean(DeliveryNumber)
print("Delivery Time :",np.mean(DeliveryTime),"\nBlood Pressure :",np.mean(BloodPre
print("Heat Problem :",np.mean(HeatProblem),"\nCaesarian :",np.mean(Caesarian))

#Median
print("\n\nMean\nAge :",np.median(Age),"\nDelivery Number :",np.median(DeliveryNumb
print("Delivery Time :",np.median(DeliveryTime),"\nBlood Pressure :",np.median(Bloo
print("Heat Problem :",np.median(HeatProblem),"\nCaesarian :",np.median(Caesarian)

#Mode
print("\n\nMean\nAge :",stats.mode(Age),"\nDelivery Number :",stats.mode(DeliveryNu
print("Delivery Time :",stats.mode(DeliveryTime),"\nBlood Pressure :",stats.mode(Bl
print("Heat Problem :",stats.mode(HeatProblem),"\nCaesarian :",stats.mode(Caesarian

#Standard Deviation
print("\n\nMean\nAge :",np.std(Age),"\nDelivery Number :",np.std(DeliveryNumber))
print("Delivery Time :",np.std(DeliveryTime),"\nBlood Pressure :",np.std(BloodPress
print("Heat Problem :",np.std(HeatProblem),"\nCaesarian :",np.std(Caesarian))

#Variance
print("\n\nMean\nAge :",np.var(Age),"\nDelivery Number :",np.var(DeliveryNumber))
print("Delivery Time :",np.var(DeliveryTime),"\nBlood Pressure :",np.var(BloodPress
print("Heat Problem :",np.var(HeatProblem),"\nCaesarian :",np.var(Caesarian))
```

```
Statistics

Average
Age : 27.6875
Delivery Number : 1.6625
Delivery Time : 0.6375
Blood Pressure : 1.0
Heat Problem : 0.375
Caesarian : 0.575


Mean
Age : 27.6875
Delivery Number : 1.6625
Delivery Time : 0.6375
Blood Pressure : 1.0
Heat Problem : 0.375
Caesarian : 0.575


Mean
Age : 27.0
```

```
Delivery Number : 1.0
Delivery Time : 0.0
Blood Pressure : 1.0
Heat Problem : 0.0
Caesarian : 1.0


Mean
Age : ModeResult(mode=array([26]), count=array([10]))
Delivery Number : ModeResult(mode=array([1]), count=array([41]))
Delivery Time : ModeResult(mode=array([0]), count=array([46]))
Blood Pressure : ModeResult(mode=array([1]), count=array([40]))
Heat Problem : ModeResult(mode=array([0]), count=array([50]))
Caesarian : ModeResult(mode=array([1]), count=array([46]))


Mean
Age : 4.9864660582420495
Delivery Number : 0.7896795236043543
Delivery Time : 0.8099961419661207
Blood Pressure : 0.7071067811865476
Heat Problem : 0.4841229182759271
Caesarian : 0.4943429983321297


Mean
Age : 24.86484375
Delivery Number : 0.6235937500000001
Delivery Time : 0.6560937499999999
Blood Pressure : 0.5
Heat Problem : 0.234375
Caesarian : 0.24437499999999995
```

```
#c)Covariance Matrix of each class
#we have to classes 0 and 1
i = 0
class0,class1 = [],[]
for c in Caesarian :
    row = [Age[i],DeliveryNumber[i],DeliveryTime[i],BloodPressure[i],HeatProblem[i
    i += 1
    if c == 0 :
        class0.append(row)
    else :
        class1.append(row)
coVar0,coVar1 = np.cov(class0),np.cov(class1)
s1,s2 = coVar0.shape, coVar1.shape
print("\nCovariance Matrices\n\nCovariance Matrix for class 0\n",coVar0,s1)
print("\nCovariance Matrix for class 1\n",coVar1,s2)
```

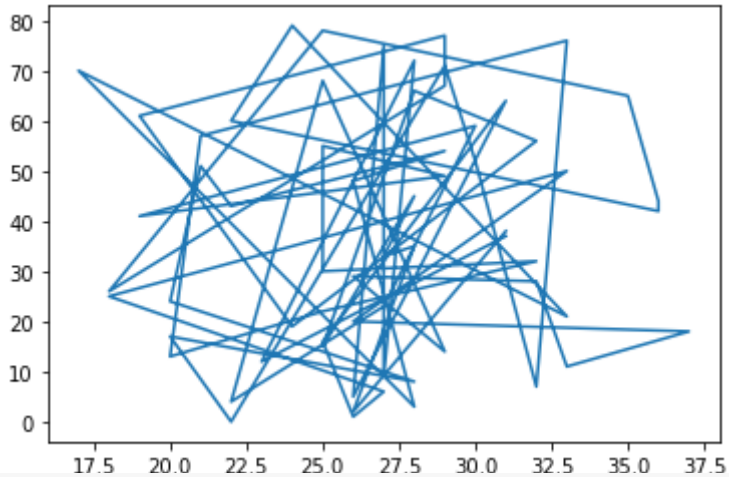
```
Covariance Matrices

Covariance Matrix for class 0
 [[ 91.   106.5  116.5  ... 131.   110.75  96.75]
 [106.5  125.5  136.5  ... 154.25 130.5  114.25]
 [116.5  136.5  149.2  ... 167.9  141.95 124.05]
 ...
 [131.   154.25 167.9  ... 190.3  160.4  140.1 ]
```

```
 [110.75 130.5  141.95 ... 160.4  135.7  118.8 ]
 [ 96.75 114.25 124.05 ... 140.1  118.8  104.2 ]] (34, 34)

Covariance Matrix for class 1
 [[128.2  108.   157.55 ... 161.65 140.7  122.4 ]
 [108.    91.   132.75 ... 136.25 118.5  103.  ]
 [157.55 132.75 193.7  ... 198.85 172.8  150.35]
 ...
 [161.65 136.25 198.85 ... 204.3  177.15 154.05]
 [140.7  118.5  172.8  ... 177.15 154.7  134.65]
 [122.4  103.   150.35 ... 154.05 134.65 118.3 ]] (46, 46)
```

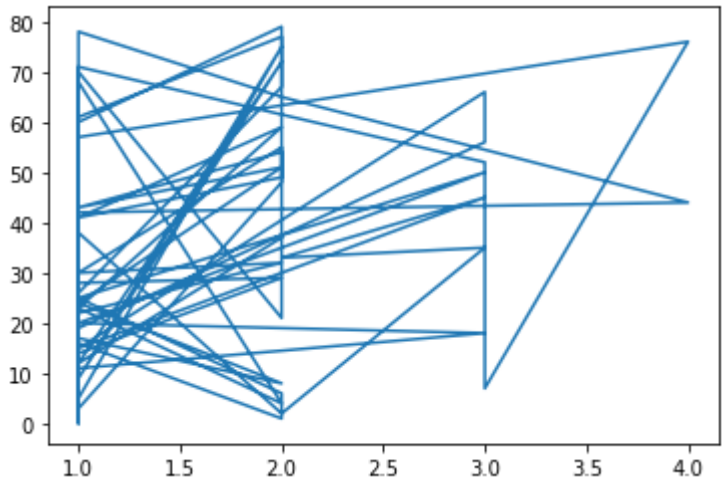
```
#d)assume gaussian distribution - ww
#For the vector consider test_size=0.5 and you get the train and test example for 
X = data.drop(['Caesarian'], axis=1)
y = data['Caesarian']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_
print("X_Train\n",X_train.head(),end="\n")
print("\nX_Test\n",X_test.head(),end="\n")
print("\nY_Train\n",y_train.head(),end="\n")
print("\nY_Test\n",y_test.head(),end="\n")
```

```
X_Train
     Age  Delivery Number  Delivery Time  Blood of Pressure  Heat Problem
38   31                1              0                  1             0
2    26                2              1                  1             0
35   28                3              0                  2             0
33   27                2              0                  1             1
45   28                3              0                  1             1

X_Test
     Age  Delivery Number  Delivery Time  Blood of Pressure  Heat Problem
63   32                2              0                  1             1
27   30                1              0                  1             0
31   40                1              0                  1             1
69   27                2              2                  0             0
46   26                1              0                  1             0

Y_Train
 38    0
2     0
35    1
33    1
45    1
Name: Caesarian, dtype: int64

Y_Test
 63    1
27    0
31    1
69    0
46    0
Name: Caesarian, dtype: int64
```

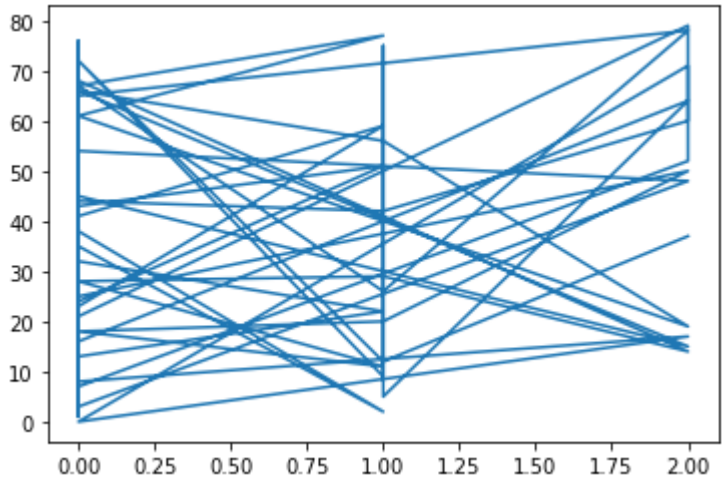
```
plt.plot(X_train["Age"],X_train.index)
plt.show()
```



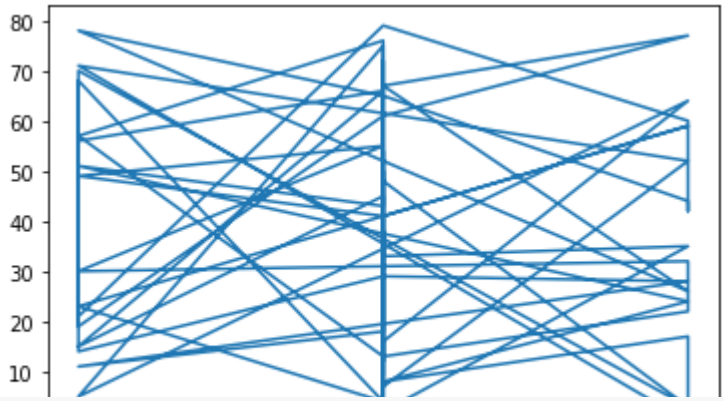

```
plt.plot(X_train["Delivery Number"],X_train.index)
plt.show()
```



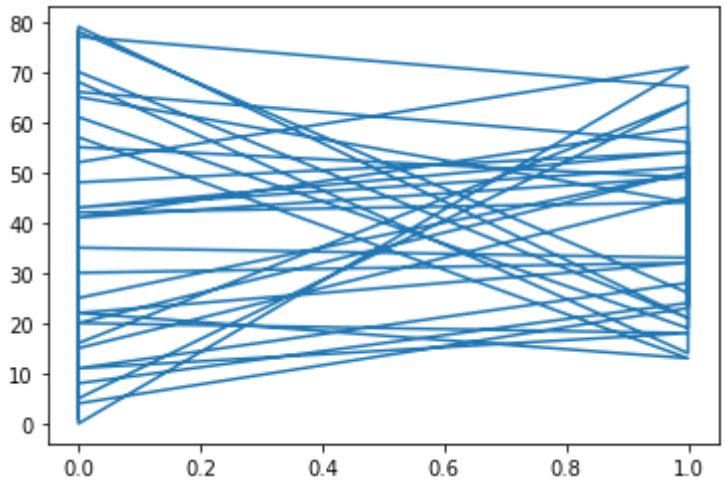

```
plt.plot(X_train["Delivery Time"],X_train.index)
plt.show()
```



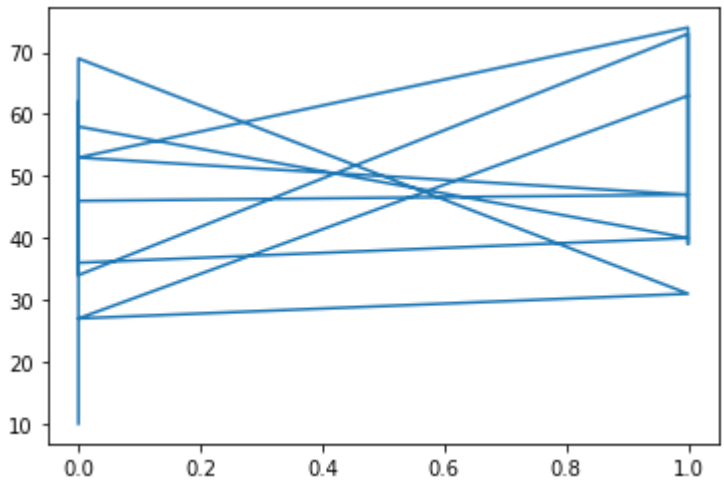

```
plt.plot(X_train["Blood of Pressure"],X_train.index)
plt.show()
```




```
plt.plot(X_train["Heat Problem"],X_train.index)
plt.show()
```

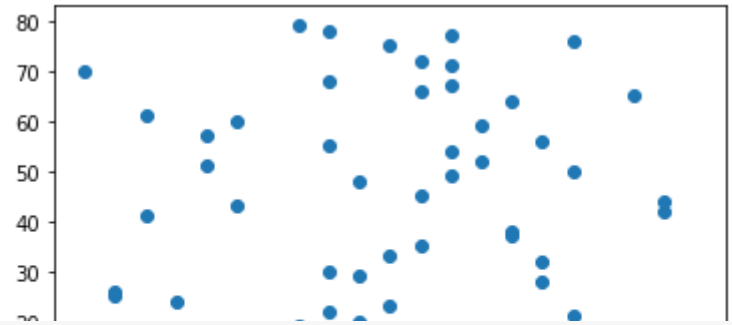



```
#Model example for train and test
plt.plot(X_test["Heat Problem"],X_test.index)
plt.show()
```




```
plt.scatter(X_train["Age"],X_train.index)
```

```
<matplotlib.collections.PathCollection at 0x7effc43328e0>
```




```
#e) confussion matrix

#geting predicted from test and actual from original database
y_predicted = y_test
y_actual = []
for i in y_test :
    y_actual.append(Caesarian[i])
print("YPred\tYactual")
for i,j in zip(y_predicted,y_actual) :
    print(i,"\t",j)

#confusion matrix
print("\nConfusion Matrix")
print(confusion_matrix(y_actual, y_predicted))
#since there are only 2 classes we get 2x2 matrix as output
```

```
YPred   Yactual
1        1
0        0
1        1
0        0
0        0
1        1
0        0
1        1
1        1
0        0
1        1
1        1
0        0
1        1
1        1
0        0

Confusion Matrix
[[7 0]
 [0 9]]
```

```
#f) Parzen Window
winSize = int(input("Enter window size : "))
#For the shake of simplicity here I'm considiring only 2 attributes Age and Delive
AgeRange = [[i-winSize,i+winSize] for i in Age]
dnRange = [[i-winSize,i+winSize] for i in DeliveryNumber]
```

```
Enter window size : 2
```

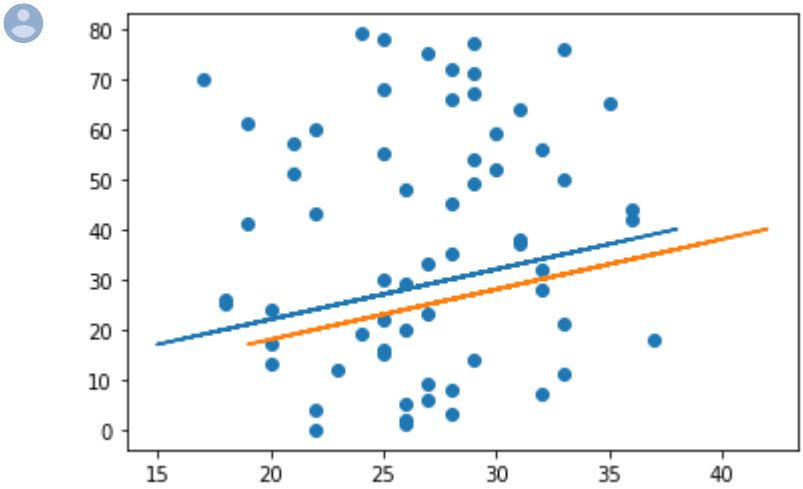
```
age = int(input("Enter age : "))
dn = int(input("Eneter Delivery Number { 1,2,3,4 } : "))
```

```
Enter age : 36
Eneter Delivery Number { 1,2,3,4 } : 3
```

```
h = round(1/(2**winSize),2)
counter,index = 0,0
cla = []
for i,j in zip(AgeRange,dnRange) :
    if (i[0] < age < i[1]) and (j[0] < dn < j[1]) :
        counter += 1
        cla.append(Caesarian[index])
    index += 1
Px = round(h * (1/len(AgeRange)) * counter,4)
if cla.count(0) > cla.count(1) :
    print("Predicted Class : 0")
else :
    print("Predicted Class : 1")
```

```
Predicted Class : 1
```

```
plt.plot(AgeRange,Age)
plt.scatter(X_train["Age"],X_train.index)
plt.show()
```




```
#g) Classifier and confussion matrix
y1 = [1,4]
y2 = [-1,-8]
w = [1,1]
e = 0.1
def computeR(y,w) :
    r = 0
    #Computing Wt*Y
    for i,j in zip(y,w) :
        r += i*j
    return r
r1 = computeR(y1,w)
r2 = computeR(y2,w)
```

```
print(r1,r2)
y = []
wn = []
while (r1 < 0) or (r2 < 0) :
    y.clear()
    if r1 < 0 :
        y = [(k*0.1) for k in y1]
    else :
        y = [(k*0.1) for k in y2]
    if not len(wn) :
        for l,m in zip(w,y) :
            wn.append(round(l+m,4))
    else :
        temp = []
        for l,m in zip(wn,y) :
            temp.append(round(l+m,4))
        wn.clear()
        wn = temp
    print(wn)
    r1 = computeR(y1,wn)
    r2 = computeR(y2,wn)
    print(r1,r2)
print(wn)
```

```
5 -9
[0.9, 0.2]
1.7000000000000002 -2.5
[0.8, -0.6]
-1.5999999999999999 4.0
[0.9, -0.2]
0.09999999999999998 0.7000000000000001
[0.9, -0.2]
```

```
#KNN city block distance
distance = []
x = []
x.append(int(input("Enter Age : ")))
x.append(int(input("Enter Delivery Number { 1,2,3,4 } : ")))
x.append(int(input("Enter Delivery Time { 0,1,2 } : ")))
x.append(int(input("Enter Blood of Pressure { 2,1,0 } : ")))
x.append(int(input("ENter Heat Problem { 1,0 } : ")))
x
```

```
Enter Age : 36
Enter Delivery Number { 1,2,3,4 } : 3
Enter Delivery Time { 0,1,2 } : 0
Enter Blood of Pressure { 2,1,0 } : 0
ENter Heat Problem { 1,0 } : 1
[36, 3, 0, 0, 1]
```

```
for i,j,k,l,m in zip(Age,DeliveryNumber,DeliveryTime,BloodPressure,HeatProblem) :
    distance.append(math.sqrt((x[0]-i)**2 + (x[1]-j)**2 + (x[2]-k)**2 + (x[3]-l)**
```

```
index = distance.index(min(distance))
```

```
index
```

```
18
```

```
Class = Caesarian[index]
print("Predicted class for entered data by KNN (City block distance) : {:}".format
```

```
Predicted class for entered data by KNN (City block distance) : 1
```

```

```