```
# Homework 1
# Author: Diego Martin Crespo
# Id: A20432558
# Fall 2018, CS-584

### Question 1

## Q1-A

#necesary libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.neighbors import NearestNeighbors as kNN
from sklearn.neighbors import KNeighborsClassifier
from numpy import linalg as LA
from scipy import linalg as LA2

#import the dataset
data = np.genfromtxt('NormalSample.csv', dtype=float, delimiter=',')
#remove the unecesary labels
datar = np.delete(data, 0, 0)
#save de data into array
xInit = []
groupInit = []
for j in datar:
    xInit.append(j[0])
    groupInit.append(j[1])

#the x array is set in order
x = sorted(xInit)
#the median in found
medianQ2=np.median(x)
#the qarra1 and qarray3 are found, the array is converted to numpy array to use
 split function
x = np.asarray(x, dtype="float", order="C")
qarra1, qarray3 = np.split(x, 2)
#the IQR= Q3-Q1 is found
q1=np.median(qarra1)
q3=np.median(qarray3)
iqr= q3 - q1
#find the right bin-width
n=len(x)
h= 2*iqr*np.power(n,-1.0/3)
print("h*=",h)

## Q1-B

#beautification of h
u=np.log10(h)
v= np.sign(u)*np.ceil(np.abs(u))
```

```
hb = np.power(10,v)
print("hb= ",hb)

## Q1-C

#plot histogram with h=0,5 of x with min=45 and max=55
xHistogram = np.histogram(x, bins=int(len(x)/0.5), range=(45,55))
ocurrencesN, valuesOcurr = np.histogram(x)
plt.hist(x, bins=int(len(x)/0.5), range=(45,55))
plt.show()
print("Histogram with h*=0.5")

## Q1-D

#plot histogram with h=1 of x with min=45 and max=55
plt.hist(x, bins=int(len(x)/1), range=(45,55))
plt.show()
print("Histogram with h*=1")

## Q1-E

#plot histogram with h=2 of x with min=45 and max=55
plt.hist(x, bins=int(len(x)/2), range=(45,55))
plt.show()
print("Histogram with h*=1")

### Question 2

## Q2-A

#definition of a plot box as pb= Q1- 1.5*IQR for the lower wisker and Q3+1.5IQR
 as the upper wisker
min=q1-1.5*iqr
max=q3+1.5*iqr
print("The five-number summary of x are:")
print(" Min= "+str(min))
print(" Q1= "+str(q1))
print(" Q2= "+str(medianQ2))
print(" Q3= "+str(q3))
print(" Max= "+str(max))

## Q2-B

##definition of a plot box by groups 0 and 1

#first the data set of normalsample needs to be separated into 2 groups
xGroup0=[]
xGroup1=[]
i=0
for n in xInit:
    if groupInit[i]==0:
```

```
        xGroup0.append(n)
        i=i+1
    else:
        xGroup1.append(n)
        i=i+1

#funtion to calculate the five-number summary
def histogramValues(xGroup):
    # the x array is set in order
    x = sorted(xGroup)
    # the median in found
    medianQ2 = np.median(x)
    # the qarra1 and qarray3 are found, the array is converted to numpy array
     to use split function
    x = np.asarray(x, dtype="float", order="C")
    qarra1= x[:np.int(np.floor(len(x)/2))]
    qarra3 = x[np.int(np.floor(len(x)/2)):]
    # the IQR= Q3-Q1 is found
    q1 = np.median(qarra1)
    q3 = np.median(qarra3)
    iqr = q3 - q1
    # definition of a plot box as pb= Q1- 1.5*IQR for the lower wisker and
     Q3+1.5IQR as the upper wisker
    min = q1 - 1.5 * iqr
    max = q3 + 1.5 * iqr

    result= [min, q1, medianQ2, q3, max]

    return result

result= []
result= histogramValues(xGroup0)
print("The five-number summary of x with group 0 are:")
print(" Min= " + str(result[0]))
print(" Q1= " + str(result[1]))
print(" Q2= " + str(result[2]))
print(" Q3= " + str(result[3]))
print(" Max= " + str(result[4]))
result= histogramValues(xGroup1)
print("The five-number summary of x with group 1 are:")
print(" Min= " + str(result[0]))
print(" Q1= " + str(result[1]))
print(" Q2= " + str(result[2]))
print(" Q3= " + str(result[3]))
print(" Max= " + str(result[4]))

## Q2-C

#drawing a plotbox of x without group
plotX =plt.boxplot(x)
plt.show()
```

```
print("Plot box x")

## Q2-D

xGroup0 = np.asarray(xGroup0, dtype="float", order="C")
xGroup1 = np.asarray(xGroup1, dtype="float", order="C")
data=[x, xGroup0, xGroup1]
plotXandGroups= plt.boxplot(data)
plt.show()
print("Plot box x, group0 and group1")


### Question 3

## Q3-A

frauds = pd.read_csv('Fraud.csv', delimiter=',')
df=pd.DataFrame(frauds)
fraud1=df.loc[df['FRAUD'] == 1]
fraud0=df.loc[df['FRAUD'] == 0]
#calculation of the percentage of fraudulation
fraudPer=round((len(fraud1)/len(frauds))*100,4)
print("The "+str(fraudPer)+"% of the investigations are fraudulent")

## Q3-B

#Box-plot of variable: 'TOTAL_SPEND'
fraudSpend=[fraud0['TOTAL_SPEND'],fraud1['TOTAL_SPEND']]
plotFraudSpend=plt.boxplot(fraudSpend, vert=False)
plt.show()
print("Plot box TOTAL_SPEND")
#Box-plot of variable: 'DOCTOR_VISITS'
fraudVisits=[fraud0['DOCTOR_VISITS'],fraud1['DOCTOR_VISITS']]
plotFraudVisits=plt.boxplot(fraudVisits, vert=False)
plt.show()
print("Plot box DOCTOR_VISITS")
#Box-plot of variable: 'NUM_CLAIMS'
fraudClaims=[fraud0['NUM_CLAIMS'],fraud1['NUM_CLAIMS']]
plotFraudClaims=plt.boxplot(fraudClaims, vert=False)
plt.show()
print("Plot box NUM_CLAIMS")
#Box-plot of variable: 'MEMBER_DURATION'
fraudMemberDur=[fraud0['MEMBER_DURATION'],fraud1['MEMBER_DURATION']]
plotFraudMemberDur=plt.boxplot(fraudMemberDur, vert=False)
plt.show()
print("Plot box MEMBER_DURATION")
#Box-plot of variable: 'OPTOM_PRESC'
fraudOption=[fraud0['OPTOM_PRESC'],fraud1['OPTOM_PRESC']]
plotFraudOption=plt.boxplot(fraudOption, vert=False)
plt.show()
print("Plot box OPTOM_PRESC")
```

```
#Box-plot of variable: 'NUM_MEMBERS'
fraudMemberDur=[fraud0['NUM_MEMBERS'],fraud1['NUM_MEMBERS']]
plotFraudMemberDur=plt.boxplot(fraudMemberDur, vert=False)
plt.show()
print("Plot box NUM_MEMBERS")

## Q3-C

print("All data Matrix = \n", df)
#from the data we take only a matrix with the variables
print(type(df))
xFraud = np.matrix(df)
xFraud = xFraud[:,[2,3,4,5,6,7]]
print("Variables Matrix = \n", xFraud)
xFraudx = xFraud.transpose() * xFraud
# Eigenvalue decomposition
evals, evecs = LA.eigh(xFraudx)
print("Eigenvalues of x = \n", evals)
print("Eigenvectors of x = \n",evecs)
#transformation matrix
transf = evecs * LA.inv(np.sqrt(np.diagflat(evals)))
print("Transformation Matrix = \n", transf)
# Here is the transformed X
transf_x = xFraud * transf;
print("Transformed X= \n", transf_x)
# Check columns of transformed X
xFraudx = transf_x.transpose() * transf_x;
print("Expect an Identity Matrix = \n", xFraudx.round())
# Orthonormalize using the orth function
orthx = LA2.orth(xFraud)
print("The orthonormalize x = \n", orthx)
# Check columns of the ORTH function
check = orthx.transpose().dot(orthx)
print("Also Expect an Identity Matrix = \n", check.round())

## Q3-D

kNNSpec = kNN(n_neighbors = 5)
trainData =
 df[['TOTAL_SPEND','DOCTOR_VISITS','NUM_CLAIMS','MEMBER_DURATION','OPTOM_PRESC'
 ,'NUM_MEMBERS']]
trainData.describe()
target = np.array(df[['FRAUD']]).ravel()
print(target)
# Build nearest neighbors
neigh = KNeighborsClassifier(n_neighbors=5)
nbrs = neigh.fit(trainData, target)
score = neigh.score(trainData,target)
print("Score = \n", score)

## Q3-E
```

```
observation = [[7500,15,3,127,2,2]]
myNeighbors = nbrs.kneighbors(observation, return_distance = False)
print("My Neighbors = \n", myNeighbors)


# See the classification result
class_result = nbrs.predict(observation)
print("Class Observation = \n",class_result)

# See the classification probabilities
class_prob = nbrs.predict_proba(observation)
print("Probability Observation= \n",class_prob)

accuracy = nbrs.score(trainData, target)

print("Accuracy = \n",accuracy)

## Q3-F

print("The given observation is fraud and the predicted is also fraud. The
 predicted probability of the observation is 20% of being fraud. Therefore, as
 it is greater than the probability calculated in a) the observation is well
 classified.")
```