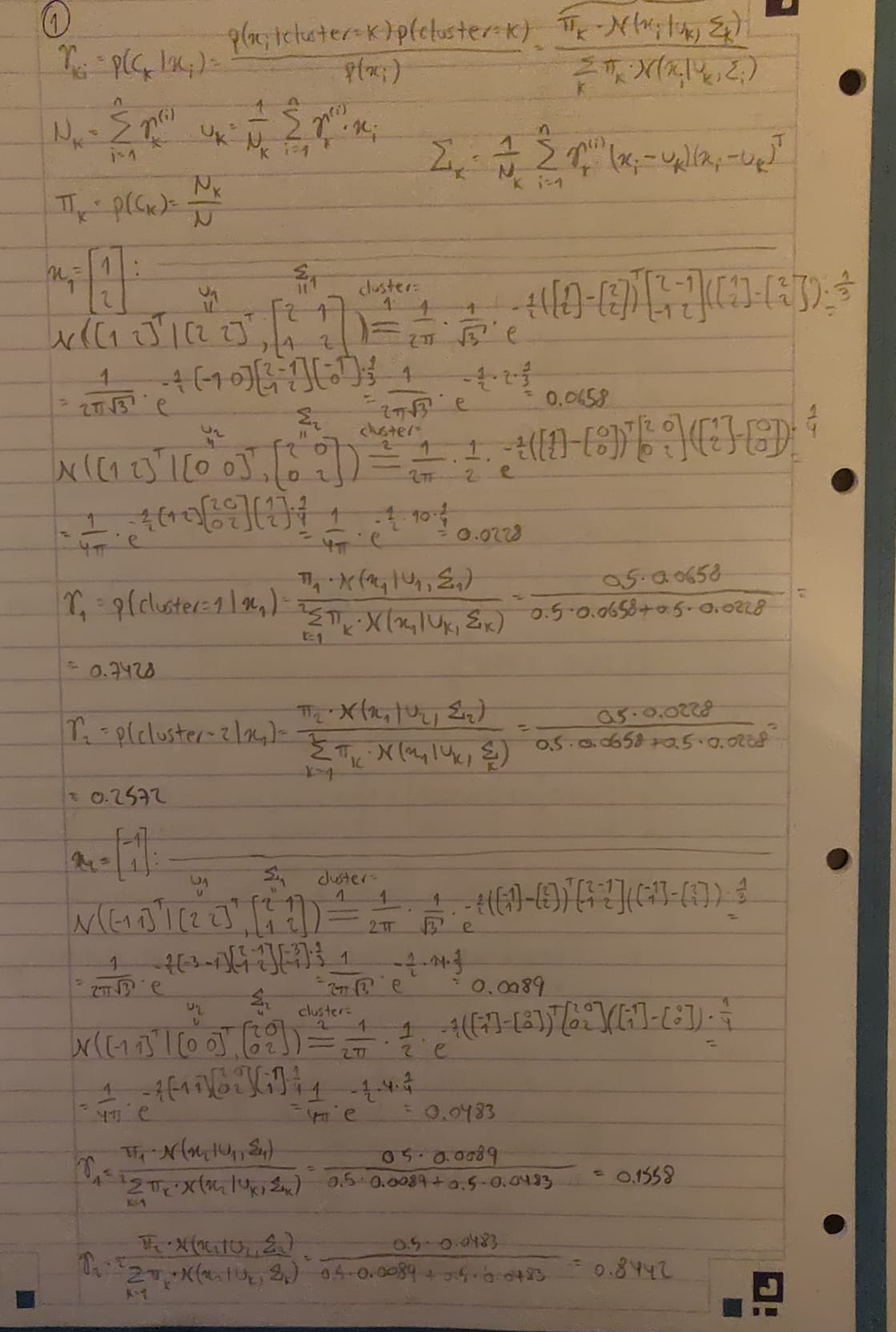
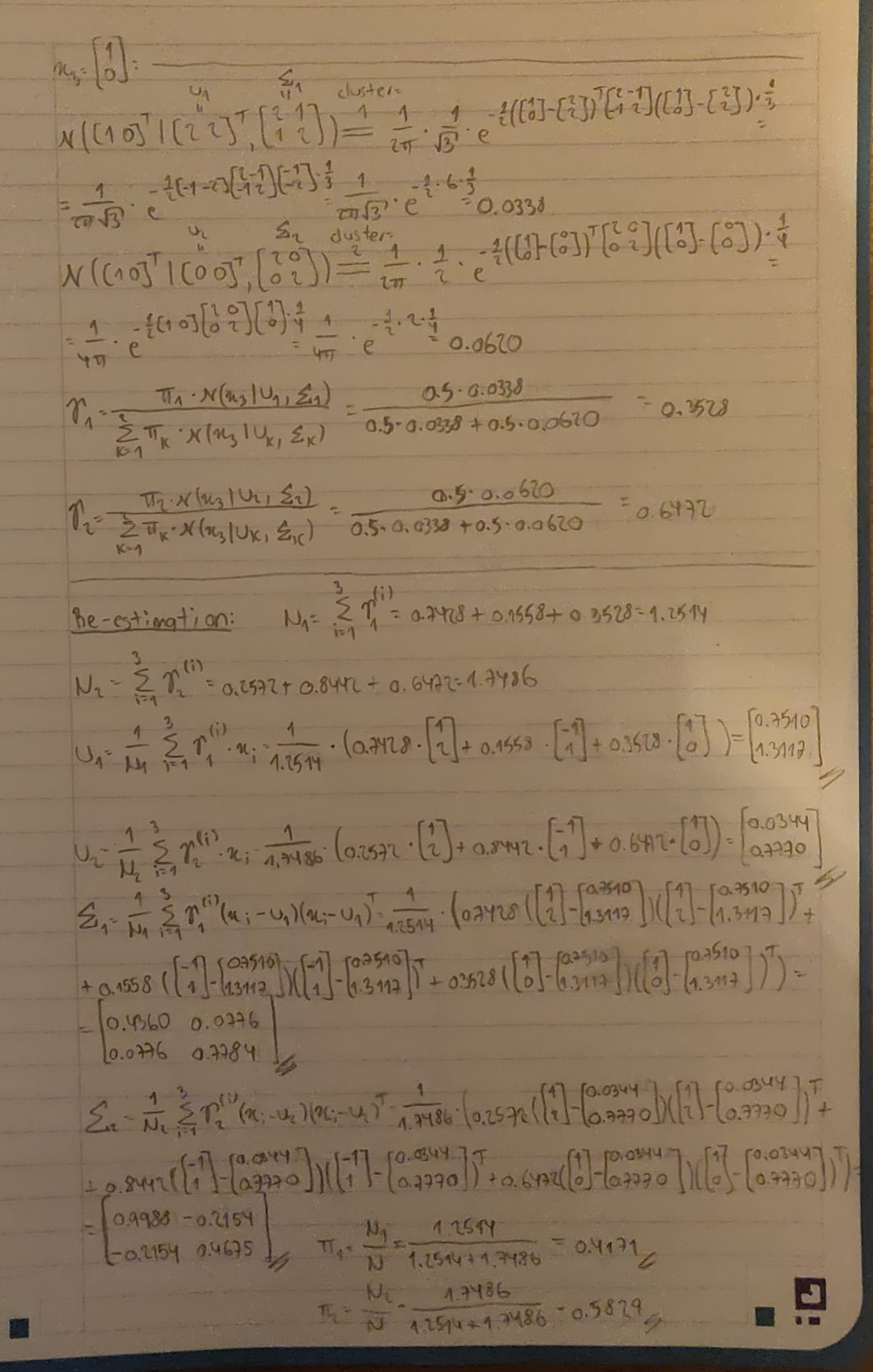
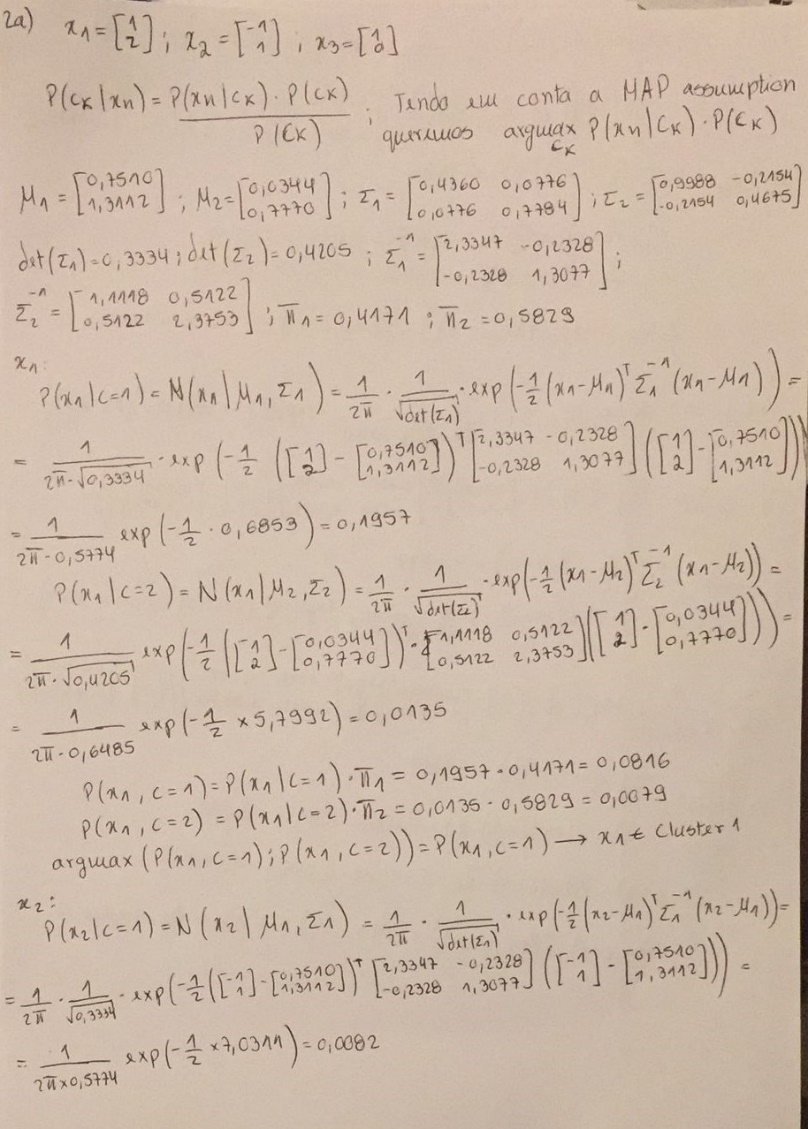
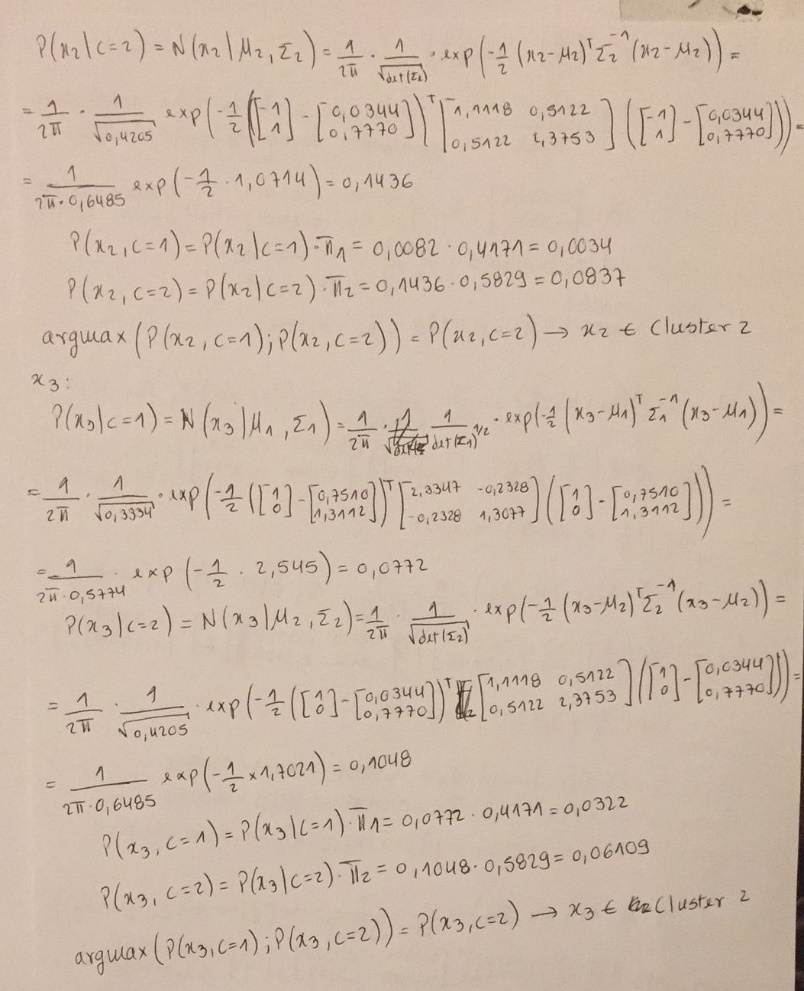
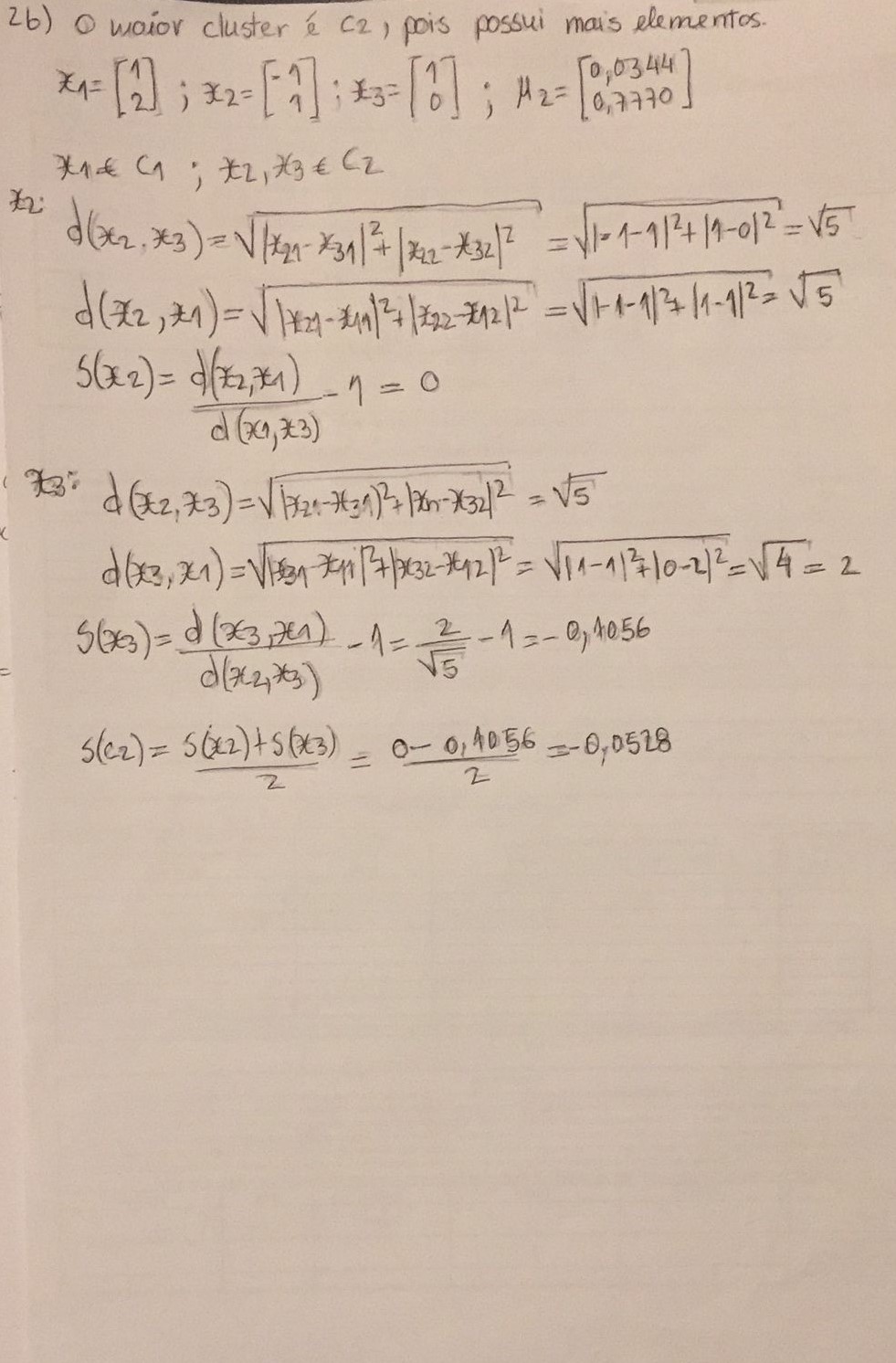
**I. Pen-and-paper**









**II. Programming and critical analysis**

1. Silhouette score of Solution 1: 0.1136;

Silhouette score of Solution 2: 0.1140;

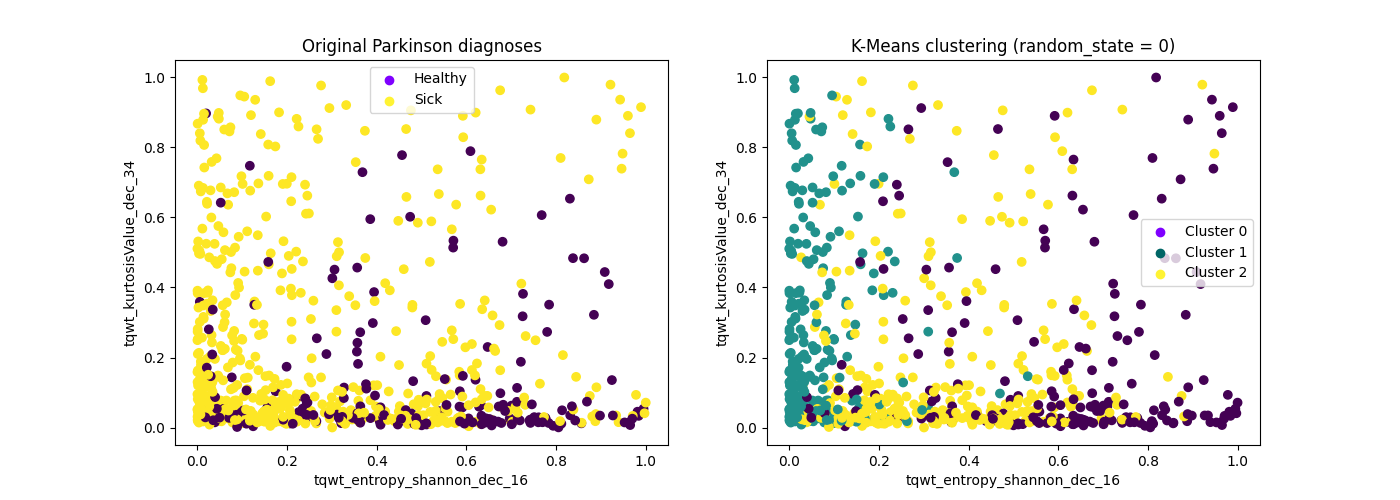
Silhouette score of Solution 3: 0.1136;

Purity of Solution 1: 0.7672;

Purity of Solution 2: 0.7632;

Purity of Solution 3: 0.7672.

A inicialização dos centróides é a principal causa de não determinismo presente neste método, daí a utilização de random\_state=n, que tem como objetivo aplicar uma determinação transformação a n de forma a atribuir pseudo-aleatoriamente determinados valores como sendo os centros de cada cluster. Como o resto do algoritmo (k-means) pode ser considerado determinista, concluimos que esta é a única causa para o aparente não-determinismo.



Number of principal components: 31.

**III. APPENDIX**

#######         Importing required libraries          #######

import pandas as pd

from scipy.io.arff import loadarff

from sklearn.preprocessing import MinMaxScaler

from sklearn import metrics, cluster

import matplotlib.pyplot as plt

import numpy as np

from sklearn.decomposition import PCA

#######           Defining Purity Function            #######

def purity\_score(y\_true, y\_pred):

    # compute contingency/confusion matrix

    confusion\_matrix = metrics.cluster.contingency\_matrix(y\_true, y\_pred)

    return round(np.sum(np.amax(confusion\_matrix, axis=0)) / np.sum(confusion\_matrix), 4)

#######             Reading the ARFF file             #######

data = loadarff('pd\_speech.arff')

df = pd.DataFrame(data[0])

df['class'] = df['class'].str.decode('utf-8')

X, y = df.drop('class', axis=1), df['class']

#######                  Normalizing                  #######

scaler = MinMaxScaler()

X\_normalized = scaler.fit\_transform(X)

#######                    K-Means                    #######

kmeans\_algo0 = cluster.KMeans(n\_clusters=3, random\_state=0)

kmeans\_algo1 = cluster.KMeans(n\_clusters=3, random\_state=1)

kmeans\_algo2 = cluster.KMeans(n\_clusters=3, random\_state=2)

# learning the model

kmeans\_model0 = kmeans\_algo0.fit(X\_normalized)

kmeans\_model1 = kmeans\_algo1.fit(X\_normalized)

kmeans\_model2 = kmeans\_algo2.fit(X\_normalized)

# getting the predicted labels

y\_pred0 = kmeans\_model0.labels\_

y\_pred1 = kmeans\_model1.labels\_

y\_pred2 = kmeans\_model2.labels\_

#######                     Ex 1                      #######

print("Silhouette score of Solution 1:", round(metrics.silhouette\_score(X\_normalized, y\_pred0), 4))

print("Purity of Solution 1:", purity\_score(y, y\_pred0))

print("Silhouette score of Solution 2:", round(metrics.silhouette\_score(X\_normalized, y\_pred1), 4))

print("Purity of Solution 2:", purity\_score(y, y\_pred1))

print("Silhouette score of Solution 3:", round(metrics.silhouette\_score(X\_normalized, y\_pred2), 4))

print("Purity of Solution 3:", purity\_score(y, y\_pred2))

#######                     Ex 3                      #######

# variance by feature of normalized data

variance = X\_normalized.var(axis=0)

# get the two highest features based on variance

two\_highest\_variance = variance.argsort()[-2:][::-1]

# plotting

plt.figure(figsize=(14, 5))

plt.subplot(121)

y\_values = np.array([int(i) for i in y.values])

plt.scatter(X\_normalized[:, two\_highest\_variance[0]], X\_normalized[:, two\_highest\_variance[1]], c=y\_values)

plt.legend(handles=[plt.scatter([], [], label='Healthy', c='#7F00FF'),

                    plt.scatter([], [], label='Sick', c='#FFF333')])

plt.title("Original Parkinson diagnoses")

plt.xlabel(X.columns[two\_highest\_variance[0]])

plt.ylabel(X.columns[two\_highest\_variance[1]])

plt.subplot(122)

plt.scatter(X\_normalized[:, two\_highest\_variance[0]], X\_normalized[:, two\_highest\_variance[1]], c=y\_pred0)

plt.legend(handles=[plt.scatter([], [], label='Cluster 0', c='#7F00FF'),

                    plt.scatter([], [], label='Cluster 1', c='#006666'),

                    plt.scatter([], [], label='Cluster 2', c='#FFF333')])

plt.title("K-Means clustering (random\_state = 0)")

plt.xlabel(X.columns[two\_highest\_variance[0]])

plt.ylabel(X.columns[two\_highest\_variance[1]])

plt.show()

#######                     Ex 4                      #######

pca = PCA()

pca.fit(X\_normalized)

i = 1

for var in np.cumsum(pca.explained\_variance\_ratio\_):

    if var > 0.8:

        break

    i += 1

print("Number of principal components:", i)

**END**