## II. Programming and critical analysis

The code for the questions is in the Appendix of this document.

1) Seed: 0

   Silhouette: 0.11362027575179438

   Purity: 0.7671957671957672

   Seed: 1

   Silhouette: 0.1140355420137708

   Purity: 0.7632275132275133

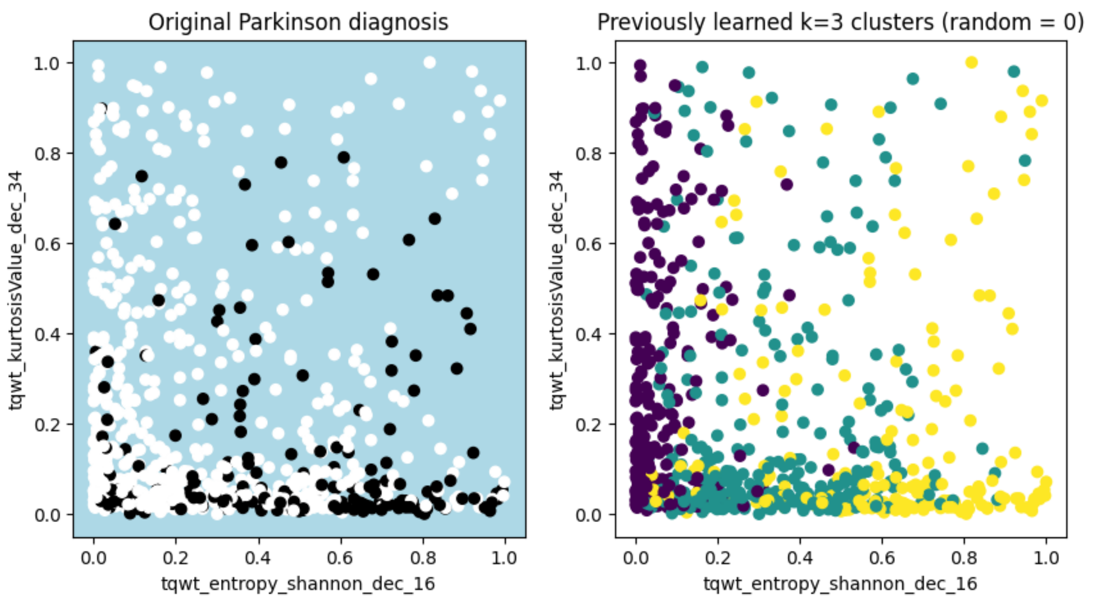
   Seed: 2

   Silhouette: 0.11362027575179438

   Purity: 0.7671957671957672

2) In the previous question, we observed that the results of the silhouette and purity were non-deterministic.
   The **`cluster.KMeans()`** function has a parameter called **`random_state`**. This parameter determines random number generation for centroid initialization and we use an integer in this parameter to make the randomness deterministic.
   Different **`random_state`** values will not prevent the algorithm from converging to the same final point since we can see that seed 0 and seed 2 results are the same.
   However, the results for seed 1 are different from the others due to the randomness in the initialization of the centroids and that is what is causing the non-determinism.

3)



4) Number of components: 31

## III. APPENDIX

Code used in question 1) of **II. Programming and critical analysis**:

Loads the data:

```
import numpy as np
import pandas as pd
from scipy.io.arff import loadarff
from sklearn.metrics import silhouette_score

# Load the data
data = loadarff('pd_speech.arff')
df = pd.DataFrame(data[0])
df['class'] = df['class'].str.decode('utf-8')

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

Calculates the silhouettes and purities:

```
from sklearn import datasets, metrics, cluster, mixture
from sklearn.preprocessing import MinMaxScaler

#normalize the data with MinMaxScaler
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
X = pd.DataFrame(X, columns=df.columns[:-1])

#compute purity
def purity_score(y_true, y_pred):
   # compute contingency/confusion matrix
   confusion_matrix = metrics.cluster.contingency_matrix(y_true, y_pred)
   return np.sum(np.amax(confusion_matrix, axis=0)) / np.sum(confusion_matrix)

seeds = [0, 1, 2]
#apply k-means clustering fully unsupervised (without targets) on the normalized data with
k = 3 and three different seeds (using random ε {0,1,2}).
for seed in seeds:
   #parameterize clustering
   kmeans = cluster.KMeans(n_clusters=3, random_state=seed)

   #learn model
   kmeans_model = kmeans.fit(X)

    # check the produced clusters
    y_pred = kmeans_model.labels_
   print("Seed: ", seed)
   print("Silhouette:", metrics.silhouette_score(X, y_pred, metric='euclidean'))
   print("Purity:", purity_score(Y, y_pred))
   print("\n")
```

Code used in question 3) of **II. Programming and critical analysis** (this code is the continuation of the code used in question 1) ):

```
import matplotlib.pyplot as plt

#select the 2 features with the highest variance
Xnew = X.var().sort_values(ascending=False).head(2)

colors = [Y, y_pred]

#plot side by side
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
for i in range(2):
   ax[i].scatter(X[Xnew.index[0]], X[Xnew.index[1]], c=colors[i])
   ax[i].set_xlabel(Xnew.index[0])
   ax[i].set_ylabel(Xnew.index[1])
   if i == 0:
       #background color
       ax[i].set_facecolor('lightblue')
       ax[i].set_title('Original Parkinson diagnosis')
   else:
       ax[i].set_title('Previously learned k=3 clusters (random = 0)')
plt.show()
```

Code used in question 4) of **II. Programming and critical analysis** (this code is the continuation of the code used in question 1) and 3) ):

```
from sklearn.decomposition import PCA

#apply PCA
pca = PCA(n_components=0.8)
pca.fit(X)

#number of components
print("Number of components:", pca.n_components_)
```