TP : Apprentissage non supervisé

Dans ce TP, nous allons passez en revue trois modèles de clustering populaires : Kmeans, clustering hiérarchique, DBSCAN.

Étape 1 : Importation des bibliothèques et des données

- Importer numpy, pandas, matplotlib.pyplot, seaborn
- Importer le fichier `Mall_customers.csv` et le stocker dans df
- Afficher le nombre de lignes et de colonnes
- Afficher les statistiques des variables
- Afficher le typage des variables
- Afficher les 10 première lignes de df

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Étape 2 : Clustrering avec Kmeans

Importer Kmeans de sklearn.

```
from sklearn.cluster import KMeans
```

Voici un exemple d'utilisation de Kmeans pour former 3 clusters

```
model = KMeans(n_clusters=3, n_init=10)
model.fit(df)
print(model.inertia_)
df['Labels'] = model.labels_
plt.figure(figsize=(12, 8))
sns.scatterplot(x=df['Income'], y=df['Score'],
hue=df['Labels'], palette=sns.color_palette('hls', 3))
plt.title('KMeans avec 3 Clusters')
plt.show()
```

- Expliquez les paramètres de KMeans : `n_clusters, n_init`
- Quels sont les autres paramètres qu'on pourrait sépécifier pour Kmeans : `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html`

— Que retourne Kmeans dans l'attribut `labels_` et l'attribut `inertia_`

Choix de K

Une stratégie simple pour identifier le nombre de clusters consiste à faire varier K et surveiller l'évolution de l'inertie intra-classes. L'idée est de visualiser le "coude". La méthode du coude nous dit de sélectionner le nombre de clusters lorsqu'il y a un changement significatif de l'inertie.

Afin de définir le nombre de cluster, Testez plusieurs Kmeans avec `n_clusters` qui varie de 1 à 10. Ensuite tracer l'évolution de la courbe d'inertie en fonction du nombre de cluster et repérer le coude. Quelles sont les meilleurs valeurs de `n_clusters` ?

Analyse des clusters

Nous avons identifier des groupes cependant il faut analyser les différences entre ces groupes afin de pouvoir donner un nom plus explicite que. Pour cela nous allons réaliser des graphiques. Il est possible de comparer nos populations avec des tests statistiques afin de juger de la significativité des différences entre clusters.

— Afficher le swarmplot entre "Labels" et "Income"
— Afficher le swarmplot entre "Labels" et "Score"
— Analyser les graphiques afin de trouver des noms à nos clusters

```
fig = plt.figure(figsize=(20,8))
ax = fig.add_subplot(121)
sns.swarmplot(x='Labels', y='Income', data=df, ax=ax)
ax = fig.add_subplot(122)
sns.swarmplot(x='Labels', y='Score', data=df, ax=ax)
plt.show()
```

Étape 3 : Hierarchical Clustering

Le clustering hiérarchique est une approche ascendante. Nous utiliserons le lien complet comme critère de liaison. Importer AgglomerativeClustering de sklearn puis Instancier la méthode avec les paramètres `n_clusters = 5, linkage= 'complete'`.

```
from sklearn.cluster import AgglomerativeClustering

modelAC5= AgglomerativeClustering(n_clusters=5, linkage= 'complete')
modelAC5.fit(df)
```

La classe de regroupement agglomératif nécessitera deux entrées :

— `n_clusters` : Le nombre de grappes à former ainsi que le nombre de centroïdes à générer.

— `linkage` : Quel critère de liaison à utiliser. Le critère de lien détermine la distance à utiliser entre les clusters. L'algorithme fusionnera les clusters qui minimisent ce critère.

Dendogramme

Pour visualiser le dendogramme, nous utilisons `hierarchy` de `scipy`.

```
from scipy.cluster.hierarchy import dendrogram, linkage

matrice_dist = linkage(df, 'complete')
plt.figure(figsize=(18, 50))
dendrogram(matrice_dist, leaf_rotation=0,
 leaf_font_size=12, orientation='right')
plt.show()
```

Étape 4 : Clustering avec DBSCAN

DBSCAN est l'acronyme de *Density-Based Spatial Clustering of Applications with Noise*. Cette technique fonctionne sur la base de la densité de l'objet. L'idée générale est que si un point particulier appartient à une cluster, il doit être proche de nombreux autres points de ce cluster.

Voici un exemple d'utilisation de DBSCAN avec les paramètres $eps = 11$ et $min\_samples = 6$

```
from sklearn.cluster import DBSCAN
model = DBSCAN(eps=11, min_samples=6)
model.fit(df)
```

Afficher et Analyser les résultats du DBSCAN.