TP 6 - CLUSTERING

NOM : GHODBANI

PRÉNOM : Fares

Groupe : Miage

Réalisé avec Canva, pandas, scikit-learn et Visual studio code

1. Exercice 1 : Clustering sur données simulées

Étape 1 : Génération des données

Objectif : Créer un jeu de données simulé avec 4 clusters.

Code :

```
from sklearn.datasets import make_blobs

import matplotlib.pyplot as plt

X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

plt.scatter(X[:, 0], X[:, 1], s=50)

plt.title("Nuage de points des données simulées")

plt.show()
```

Résultat :

```
>>> import matplotlib.pyplot as plt
>>> plt.scatter(X[:, 0], X[:, 1], s=50)
<matplotlib.collections.PathCollection object at 0x000001E32CEF5940>
>>> print("X :\n", X)
X :
 [[ 8.36856841e-01  2.13635938e+00]
 [-1.41365810e+00  7.40962324e+00]
 [ 1.15521298e+00  5.09961887e+00]
 [-1.01861632e+00  7.81491465e+00]
 [ 1.27135141e+00  1.89254207e+00]
 [ 3.43761754e+00  2.61654166e-01]
 [-1.80822253e+00  1.59701749e+00]
 [ 1.41372442e+00  4.38117707e+00]
 [-2.04932168e-01  8.43209665e+00]
 [-7.11099611e-01  8.66043846e+00]
 [-1.71237268e+00  2.77780226e+00]
 [-2.67000792e+00  8.35389140e+00]
 [ 1.24258802e+00  4.50399192e+00]
 [-2.22783649e+00  6.89479938e+00]
 [ 1.45513831e+00 -2.91989981e-02]
 [ 4.53791789e-01  3.95647753e+00]
 [ 1.06923853e+00  4.53068484e+00]
 [ 2.56936589e+00  5.07048304e-01]
```

```
>>> print("\ny_true :\n", y_true)

y_true :
 [1 3 0 3 1 1 2 0 3 3 2 3 0 3 1 0 0 1 2 2 1 1 0 2 2 0 1 0 2 0 3 3 0 3 3 3 3
 3 2 1 0 2 0 0 2 2 3 2 3 1 2 1 3 1 1 2 3 2 3 1 3 0 3 2 2 2 3 1 3 2 0 2 3 2
 2 3 2 0 1 3 1 0 1 1 3 0 1 0 3 3 0 1 3 2 2 0 1 1 0 2 3 1 3 1 0 1 1 0 3 0 2
 2 1 3 1 0 3 1 1 0 2 1 2 1 1 1 1 2 1 2 3 2 2 1 3 2 2 3 0 3 3 2 0 2 0 2 3 0
 3 3 3 0 3 0 1 2 3 2 1 0 3 0 0 1 0 2 2 0 1 0 0 3 1 0 2 3 1 1 0 2 1 0 2 2 0
 0 0 0 1 3 0 2 0 0 2 2 2 0 2 3 0 2 1 2 0 3 2 3 0 3 0 2 0 0 3 2 2 1 1 0 3 1
 1 2 1 2 0 3 3 0 0 3 0 1 2 0 1 2 3 2 1 0 1 3 3 3 3 2 2 3 0 2 1 0 2 2 2 1 1
 3 0 0 2 1 3 2 0 3 0 1 1 2 2 0 1 1 1 0 3 3 1 1 0 1 1 1 3 2 3 0 1 1 3 3 3 1
 1 0 3 2]
>>> plt.scatter(X[:, 0], X[:, 1], s=50)
<matplotlib.collections.PathCollection object at 0x000001E32CF260D0>
>>> plt.title("Données générées (4 clusters)")
Text(0.5, 1.0, 'Données générées (4 clusters)')
>>> plt.xlabel("Feature 1")
Text(0.5, 0, 'Feature 1')
>>> plt.ylabel("Feature 2")
Text(0, 0.5, 'Feature 2')
>>> plt.show()
>>> plt.show()
```

figure_1

Remarque :

- ***y_true contient les vraies classes pour chaque point, utiles pour comparer ensuite avec le clustering.***
- ***Les points sont répartis en 4 groupes distincts mais aléatoires.***

Étape 2 : Application de K-Means

Objectif : Identifier automatiquement les clusters avec K-Means.

Code :

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=4, random_state=0)

y_kmeans = kmeans.fit_predict(X)

print("Classe prédite du 2ème point :", y_kmeans[1], "Vraie classe :", y_true[1])

print("Classe prédite du 5ème point :", y_kmeans[4], "Vraie classe :", y_true[4])
```

Résultat :

```
>>> kmeans = KMeans(n_clusters=4, random_state=0)
>>> kmeans.fit(X)
KMeans(n_clusters=4, random_state=0)
>>> y_kmeans = kmeans.predict(X)
>>> y_kmeans = kmeans.predict(X)
>>> print("Classes prédites par k-Means pour les éléments de X :")
Classes prédites par k-Means pour les éléments de X :
>>> print(y_kmeans)
[1 2 0 2 1 1 3 0 2 2 3 2 0 2 1 0 0 1 3 3 1 1 0 3 3 0 1 0 3 0 2 2 0 2 2 2 2
 2 3 1 0 3 0 0 3 3 2 3 2 1 3 1 2 1 1 3 2 3 2 1 2 0 2 3 3 3 2 1 2 3 0 3 2 3
 3 2 3 0 1 2 1 0 1 1 2 0 1 0 2 2 0 1 2 3 3 0 1 1 0 3 2 1 2 1 0 1 1 0 2 0 3
 3 1 2 1 0 2 1 1 0 3 1 3 1 1 1 1 3 1 3 2 3 3 1 2 3 3 2 0 2 2 3 0 3 0 3 2 0
 2 2 2 0 2 0 1 3 2 3 1 0 2 0 0 1 0 3 3 0 1 0 0 2 1 0 3 2 1 1 0 3 1 0 3 3 0
 0 0 0 1 2 0 3 0 0 3 3 3 0 3 2 0 3 1 3 0 2 3 2 0 2 0 3 0 0 2 3 3 1 1 0 2 1
 1 3 1 3 0 2 2 0 0 2 0 1 3 0 1 3 2 3 1 0 1 2 2 2 2 3 3 2 0 3 1 0 3 3 3 1 1
 2 0 0 3 1 2 3 0 2 0 1 1 3 3 0 1 1 1 0 2 2 1 1 0 1 1 1 2 3 2 0 1 1 2 2 2 1
 1 0 2 3]
>>> print("Deuxième élément de X :", X[1])
Deuxième élément de X : [-1.4136581   7.40962324]
>>> print("Classe prédite :", y_kmeans[1])
Classe prédite : 2
>>> print("Vraie classe :", y_true[1])
Vraie classe : 3
>>> # Cinquième élément de X
>>> print("Cinquième élément de X :", X[4])
Cinquième élément de X : [1.27135141 1.89254207]
>>>
>>> # Classe prédite par k-Means
>>> print("Classe prédite :", y_kmeans[4])
Classe prédite : 1
>>>
>>> # Vraie classe
>>> print("Vraie classe :", y_true[4])
Vraie classe : 1
```

Remarque :

- **Les numéros de clusters attribués par K-Means ne correspondent pas nécessairement aux vraies classes (ordre arbitraire).**
- **K-Means regroupe les points de manière cohérente selon leur proximité.**

Étape 3 : Visualisation des clusters

Code :

```
for i in range(4):
```

```
plt.scatter(X[y_kmeans == i, 0], X[y_kmeans == i, 1], s=50)

plt.title("Clusters identifiés par K-Means")

plt.show()
```

Résultat :

```
>>> # Vraie classe
>>> print("Vraie classe :", y_true[4])
Vraie classe : 1
>>> print("Vraies classes (y_true) :")
Vraies classes (y_true) :
>>> print(y_true)
[1 3 0 3 1 1 2 0 3 3 2 3 0 3 1 0 0 1 2 2 1 1 0 2 2 0 1 0 2 0 3 3 0 3 3 3 3
 3 2 1 0 2 0 0 2 2 3 2 3 1 2 1 3 1 1 2 3 2 3 1 3 0 3 2 2 2 3 1 3 2 0 2 3 2
 2 3 2 0 1 3 1 0 1 1 3 0 1 0 3 3 0 1 3 2 2 0 1 1 0 2 3 1 3 1 0 1 1 0 3 0 2
 2 1 3 1 0 3 1 1 0 2 1 2 1 1 1 1 2 1 2 3 2 2 1 3 2 2 3 0 3 3 2 0 2 0 2 3 0
 3 3 3 0 3 0 1 2 3 2 1 0 3 0 0 1 0 2 2 0 1 0 0 3 1 0 2 3 1 1 0 2 1 0 2 2 0
 0 0 0 1 3 0 2 0 0 2 2 2 0 2 3 0 2 1 2 0 3 2 3 0 3 0 2 0 0 3 2 2 1 1 0 3 1
 1 2 1 2 0 3 3 0 0 3 0 1 2 0 1 2 3 2 1 0 1 3 3 3 3 2 2 3 0 2 1 0 2 2 2 1 1
 3 0 0 2 1 3 2 0 3 0 1 1 2 2 0 1 1 1 0 3 3 1 1 0 1 1 1 3 2 3 0 1 1 3 3 3 1
 1 0 3 2]
>>> print("Classes prédites par k-Means (y_kmeans) :")
Classes prédites par k-Means (y_kmeans) :
>>> print(y_kmeans)
[1 2 0 2 1 1 3 0 2 2 3 2 0 2 1 0 0 1 3 3 1 1 0 3 3 0 1 0 3 0 2 2 0 2 2 2 2
 2 3 1 0 3 0 0 3 3 2 3 2 1 3 1 2 1 1 3 2 3 2 1 2 0 2 3 3 3 2 1 2 3 0 3 2 3
 3 2 3 0 1 2 1 0 1 1 2 0 1 0 2 2 0 1 2 3 3 0 1 1 0 3 2 1 2 1 0 1 1 0 2 0 3
 3 1 2 1 0 2 1 1 0 3 1 3 1 1 1 1 3 1 3 2 3 3 1 2 3 3 2 0 2 2 3 0 3 0 3 2 0
 2 2 2 0 2 0 1 3 2 3 1 0 2 0 0 1 0 3 3 0 1 0 0 2 1 0 3 2 1 1 0 3 1 0 3 3 0
 0 0 0 1 2 0 3 0 0 3 3 3 0 3 2 0 3 1 3 0 2 3 2 0 2 0 3 0 0 2 3 3 1 1 0 2 1
 1 3 1 3 0 2 2 0 0 2 0 1 3 0 1 3 2 3 1 0 1 2 2 2 2 3 3 2 0 3 1 0 3 3 3 1 1
 2 0 0 3 1 2 3 0 2 0 1 1 3 3 0 1 1 1 0 2 2 1 1 0 1 1 1 2 3 2 0 1 1 2 2 2 1
 1 0 2 3]
```

figure_1.2

Remarque :

- La couleur des clusters est automatique.
- On peut comparer visuellement avec le nuage de points initial pour juger de la qualité du clustering.

2. Exercice 2 : Clustering sur le dataset Wine

Étape 1 : Chargement et exploration des données

Code :

```
import pandas as pd

df = pd.read_csv("wine.csv")

print(df.head(10))

print(df.info())

print(df.isnull().sum())
```

Résultat :

```
>>> print(df.head(10))
   Class  Alcohol  MalicAcid   Ash  AlcalinityAsh  ...  Proanthocyanins  Color   Hue  OD280/OD315  Proline
0      1    14.23       1.71  2.43           15.6  ...             2.29   5.64  1.04         3.92     1065
1      1    13.20       1.78  2.14           11.2  ...             1.28   4.38  1.05         3.40     1050
2      1    13.16       2.36  2.67           18.6  ...             2.81   5.68  1.03         3.17     1185
3      1    14.37       1.95  2.50           16.8  ...             2.18   7.80  0.86         3.45     1480
4      1    13.24       2.59  2.87           21.0  ...             1.82   4.32  1.04         2.93      735
5      1    14.20       1.76  2.45           15.2  ...             1.97   6.75  1.05         2.85     1450
6      1    14.39       1.87  2.45           14.6  ...             1.98   5.25  1.02         3.58     1290
7      1    14.06       2.15  2.61           17.6  ...             1.25   5.05  1.06         3.58     1295
8      1    14.83       1.64  2.17           14.0  ...             1.98   5.20  1.08         2.85     1045
9      1    13.86       1.35  2.27           16.0  ...             1.85   7.22  1.01         3.55     1045

[10 rows x 14 columns]
>>> print(df.info())
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Class                178 non-null    int64
 1   Alcohol              178 non-null    float64
 2   MalicAcid            178 non-null    float64
 3   Ash                  178 non-null    float64
 4   AlcalinityAsh        178 non-null    float64
 5   Magnesium            178 non-null    int64
 6   PhenolsTotal         178 non-null    float64
 7   Flavanoids           178 non-null    float64
 8   NonflavanoidPhenols  178 non-null    float64
 9   Proanthocyanins      178 non-null    float64
 10  Color                178 non-null    float64
 11  Hue                  178 non-null    float64
 12  OD280/OD315          178 non-null    float64
 13  Proline              178 non-null    int64
dtypes: float64(11), int64(3)
```

```
memory usage: 19.6 KB
None
>>> print(df.isnull().sum())
Class                  0
Alcohol                0
MalicAcid              0
Ash                    0
AlcalinityAsh          0
Magnesium              0
PhenolsTotal           0
Flavanoids             0
NonflavanoidPhenols    0
Proanthocyanins        0
Color                  0
Hue                    0
OD280/OD315            0
Proline                0
dtype: int64
```

Remarques :

- ***Le dataset contient 178 vins, 13 attributs chimiques et une colonne Class indiquant le producteur.***
- ***Pas de valeurs nulles.***
- ***La colonne Class sera utilisée seulement pour comparaison après clustering.***

Étape 2 : Préparation des données pour PCA

Objectif : Réduire les données à 2 dimensions pour visualisation.

Code :

```
from sklearn.decomposition import PCA

import numpy as np

donnees = df.drop("Class", axis=1).to_numpy()

pca = PCA(n_components=2)

reduced_data = pca.fit_transform(donnees)
```

Remarques :

- ***PCA permet de conserver la plus grande partie de la variance tout en projetant les données en 2D.***
- ***reduced_data contiendra les coordonnées à deux dimensions pour chaque vin.***

Étape 3 : Application de K-Means

Code :

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=0)

classe_kmeans = kmeans.fit_predict(reduced_data)

labels_uniques = np.unique(classe_kmeans)

print("Valeurs des classes prédites :", labels_uniques)
```

Résultat :

```
>>> print("Valeurs des classes prédites :", labels_uniques)
Valeurs des classes prédites : [0 1 2]
```

Remarques :

- ***K-Means tente de regrouper les vins en 3 clusters.***
- ***Les numéros des clusters ne correspondent pas nécessairement aux vrais labels (Class).***

Étape 4 : Visualisation des clusters prédits

Code :

```
import matplotlib.pyplot as plt

for label in labels_uniques:
```

```
    to_plot = reduced_data[np.where(classe_kmeans == label)[0]]
    plt.scatter(to_plot[:, 0], to_plot[:, 1], s=20)
plt.title("Clusters prédits par K-Means (Wine)")
plt.show()
```

*Résultat : *figure 3.**

Remarques :

- **La visualisation montre la séparation automatique des points.**
- **Certains points peuvent être regroupés différemment par rapport aux vraies classes.**

Étape 5 : Visualisation par vraies classes

Code :

```
classe = df["Class"]
labels_uniques = np.unique(classe)
for label in labels_uniques:
    to_plot = reduced_data[np.where(classe == label)[0]]
    plt.scatter(to_plot[:, 0], to_plot[:, 1], s=20)
plt.title("Répartition des vins selon les vraies classes")
plt.show()
```

*Résultat : *figure 2.**

Remarques :

- **Permet de comparer visuellement la performance de K-Means.**
- **On remarque que certains clusters K-Means regroupent des points de différentes vraies classes.**
- **La réduction PCA peut aussi affecter la position relative des points.**

3. Observations générales

- **K-Means fonctionne mieux lorsque les clusters sont bien séparés et sphériques.**
- **Les couleurs des clusters sont automatiques et peuvent changer à chaque exécution.**
- **PCA est utile pour visualiser des données à haute dimension mais peut légèrement déformer les distances.**

4. Conclusion

- ***Le clustering K-Means permet d'identifier des groupes similaires dans les données, même sans labels.***
- ***Les clusters trouvés ne correspondent pas toujours exactement aux vraies classes, mais la tendance générale est respectée.***
- ***La combinaison PCA + K-Means est très pratique pour explorer visuellement la structure des données.***