TP-CLUSTERING

Battistini Lisa

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
import random
```

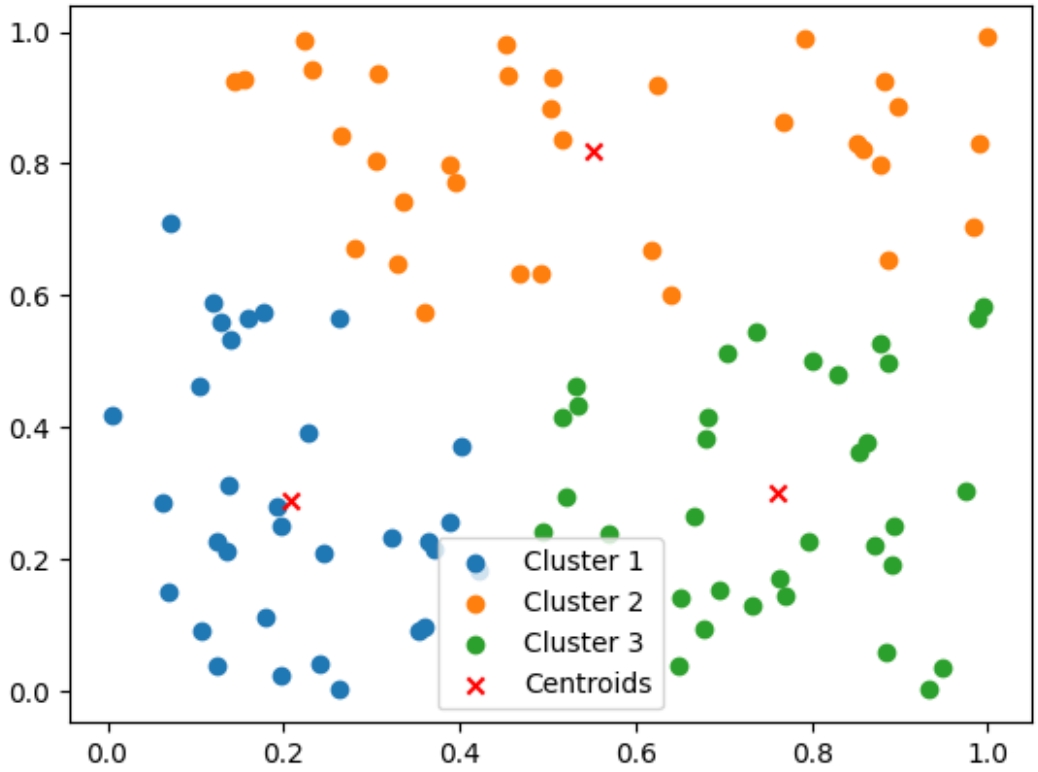
Méthode des K-moyennes

```
def k_means(X, K):
    # Initialisation aléatoire des centres de clusters (centroïdes)
    centroids = X[np.random.choice(X.shape[0], K, replace=False)]

    # Répétez jusqu'à convergence
    while True:
        # Attribution de chaque point au centroïde le plus proche
        clusters = [[] for _ in range(K)]
        for x in X:
            distances = [np.linalg.norm(x - centroid) for centroid in
centroids]
            closest_centroid = np.argmin(distances)
            clusters[closest_centroid].append(x)

        # Calcul des nouveaux centroïdes
        new_centroids = np.array([np.mean(cluster, axis=0) for cluster
in clusters])

        # Vérifiez si les centroïdes ont changé
        if np.all(centroids == new_centroids):
            break

        centroids = new_centroids

    return clusters, centroids
```

On test notre algorithme sur des données simulées, pour vérifier son fonctionnement.

```
# Générez des données simulées
X = np.random.rand(100, 2)  # 100 points dans un espace 2D
```

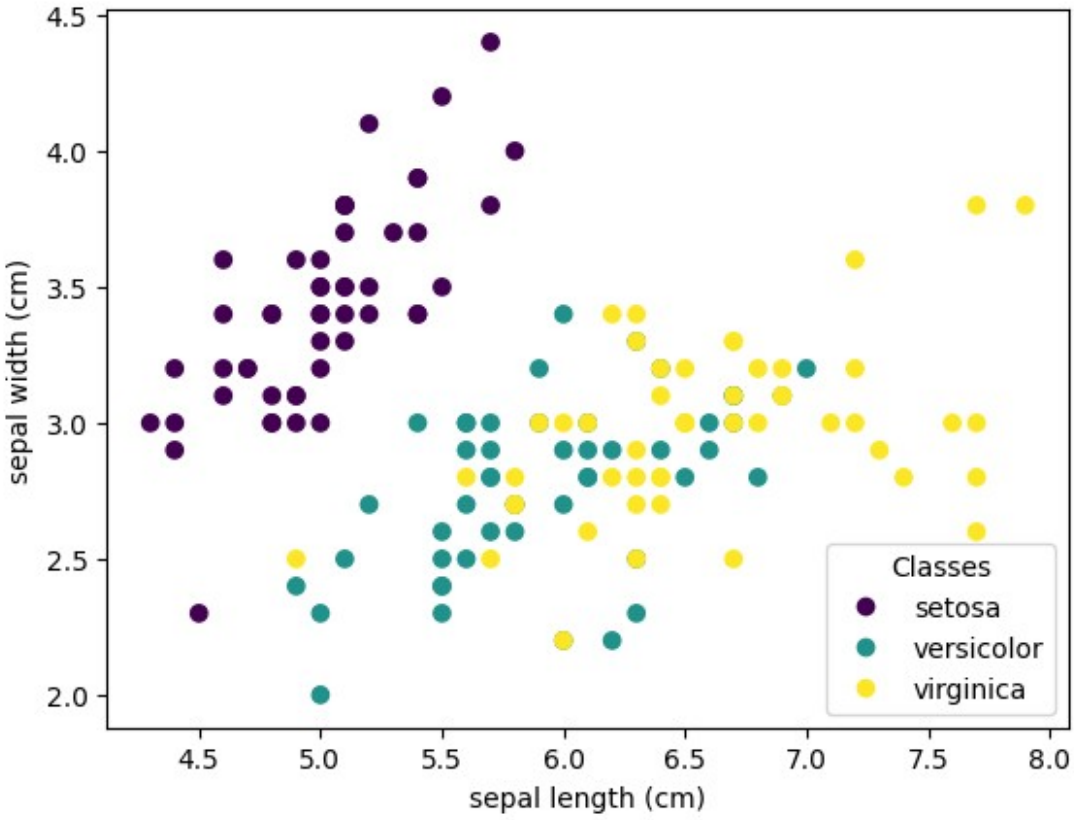
```
# Appliquez la méthode K-means
K = 3  # Nombre de clusters
clusters, centroids = k_means(X, K)

# Affichez les résultats
import matplotlib.pyplot as plt
for i, cluster in enumerate(clusters):
    cluster = np.array(cluster)
    plt.scatter(cluster[:, 0], cluster[:, 1], label=f'Cluster {i+1}')
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='x',
label='Centroids')
plt.legend()
plt.show()
```



##Préparation du jeu de données

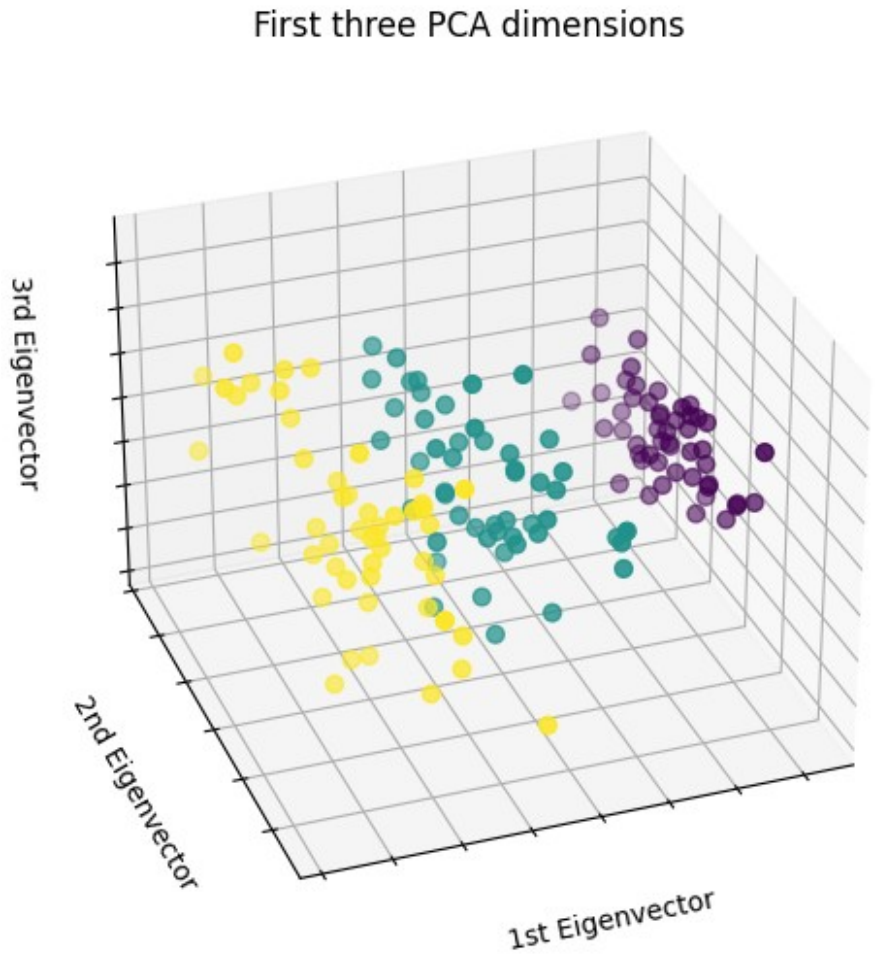
```
#téléchargement des données IRIS
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
y = iris.target

#Visualisation 2D
import matplotlib.pyplot as plt
```

```
_, ax = plt.subplots()
scatter = ax.scatter(iris.data[:, 0], iris.data[:, 1], c=iris.target)
ax.set(xlabel=iris.feature_names[0], ylabel=iris.feature_names[1])
_ = ax.legend(
    scatter.legend_elements()[0], iris.target_names, loc="lower
right", title="Classes"
)
```




```
#Visualisation 3D
import mpl_toolkits.mplot3d  # noqa: F401

from sklearn.decomposition import PCA

fig = plt.figure(1, figsize=(8, 6))
ax = fig.add_subplot(111, projection="3d", elev=-150, azim=110)

X_reduced = PCA(n_components=3).fit_transform(iris.data)
ax.scatter(
    X_reduced[:, 0],
    X_reduced[:, 1],
    X_reduced[:, 2],
    c=iris.target,
    s=40,
```

```
)

ax.set_title("First three PCA dimensions")
ax.set_xlabel("1st Eigenvector")
ax.xaxis.set_ticklabels([])
ax.set_ylabel("2nd Eigenvector")
ax.yaxis.set_ticklabels([])
ax.set_zlabel("3rd Eigenvector")
ax.zaxis.set_ticklabels([])

plt.show()
```




```
# Convertir en DataFrame pour une meilleure analyse
df_iris = pd.DataFrame(X, columns=iris.feature_names)
print(df_iris.head())
print(df_iris.describe())
```

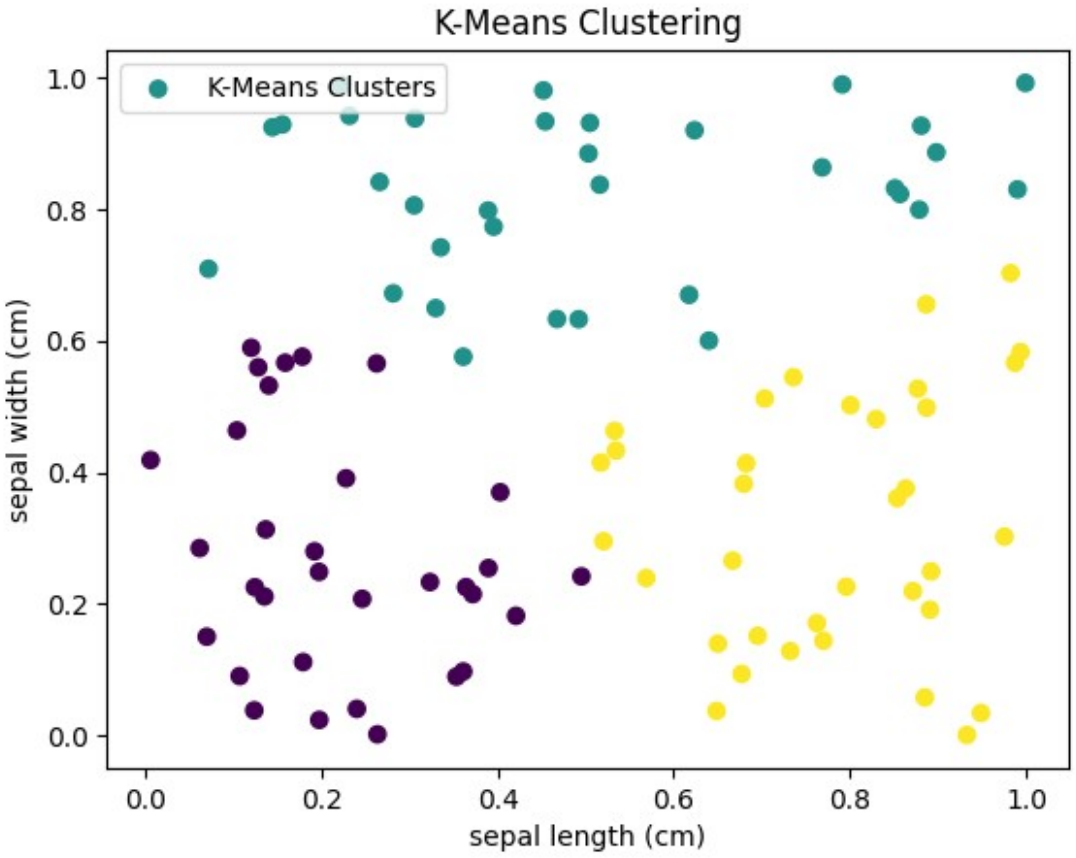
Comme les classes sont déjà formées sur notre jeu de données, on ne considère pas la dernière colonne.

Application du Clustering

K-moyennes

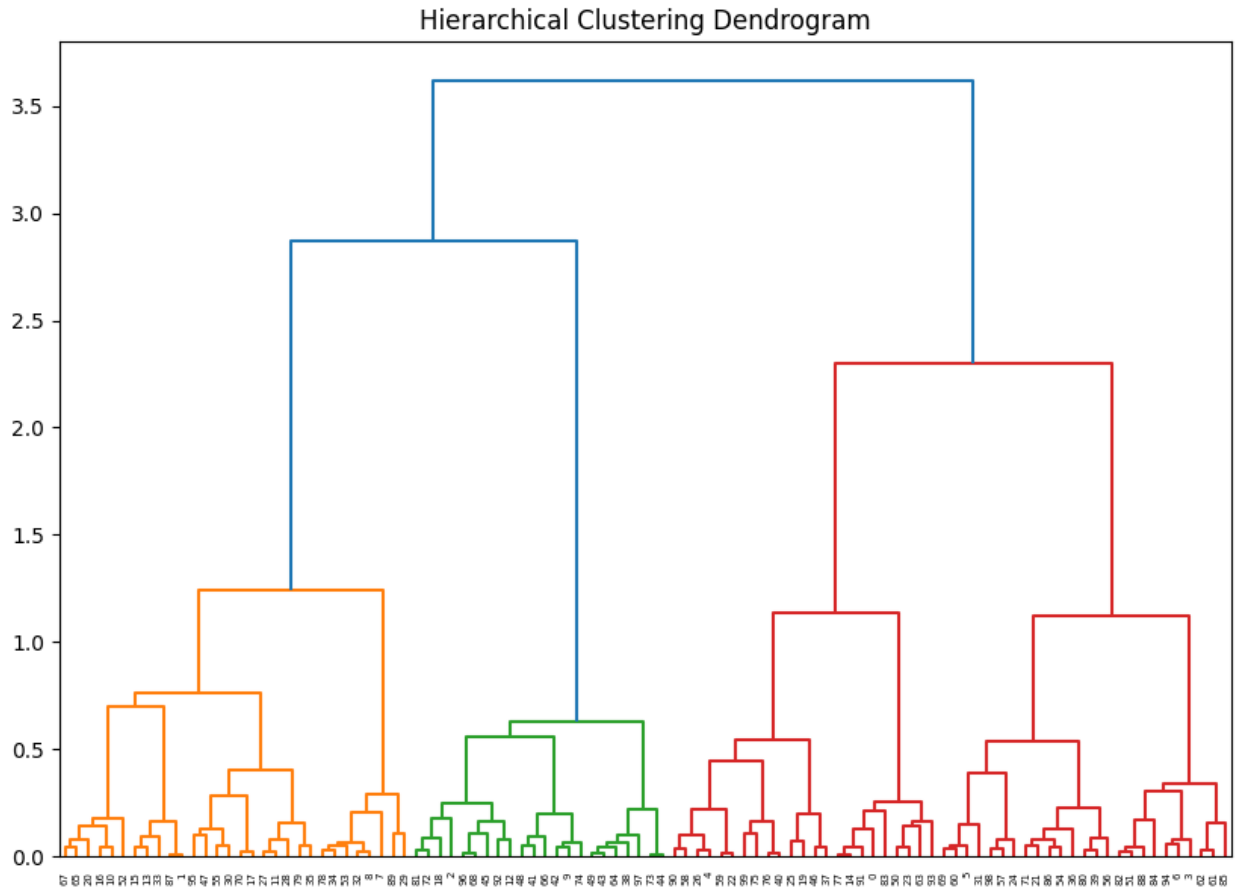
```
from sklearn.cluster import KMeans

# Application du clustering K-Means
kmeans = KMeans(n_clusters=3, random_state=0)
clusters_kmeans = kmeans.fit_predict(X)

# Visualisation des clusters K-Means
plt.scatter(X[:, 0], X[:, 1], c=clusters_kmeans, cmap='viridis',
label='K-Means Clusters')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title("K-Means Clustering")
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
```



Clustering hiérarchique

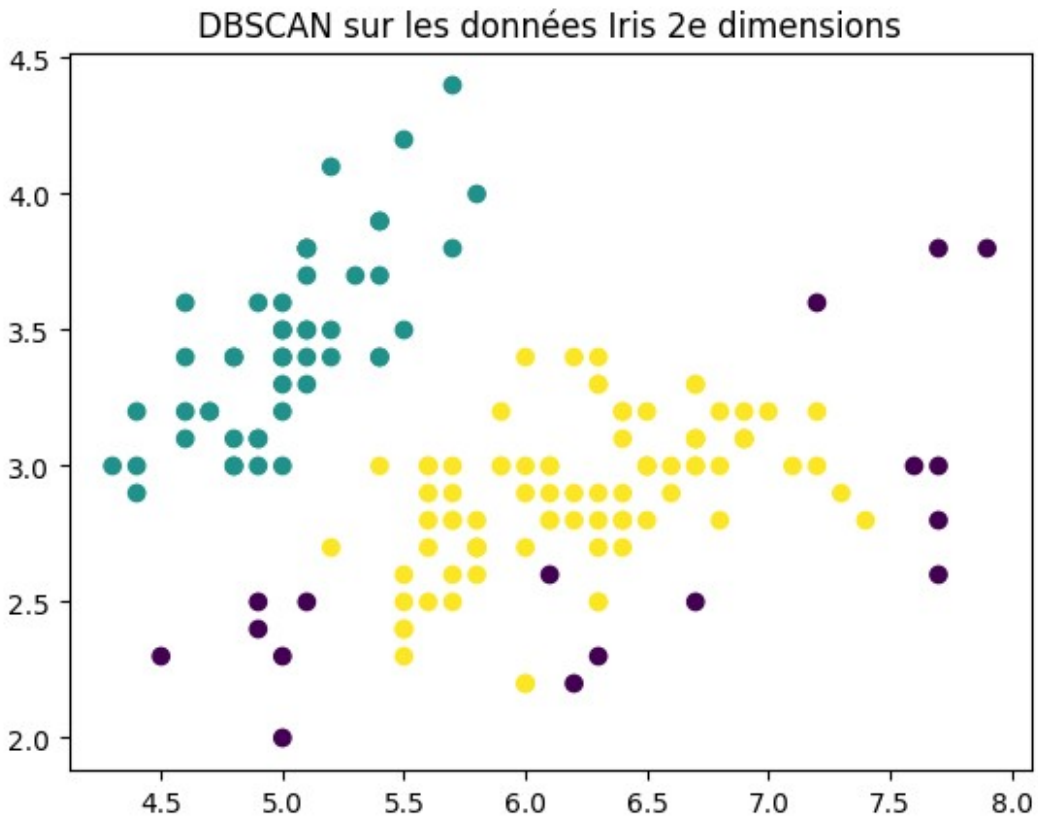
```
from scipy.cluster.hierarchy import dendrogram, linkage

# Application du clustering hiérarchique
linked = linkage(X, 'ward')

# Visualisation du dendrogramme
plt.figure(figsize=(10, 7))
dendrogram(linked, orientation='top', distance_sort='descending',
show_leaf_counts=True)
plt.title("Hierarchical Clustering Dendrogram")
plt.show()
```

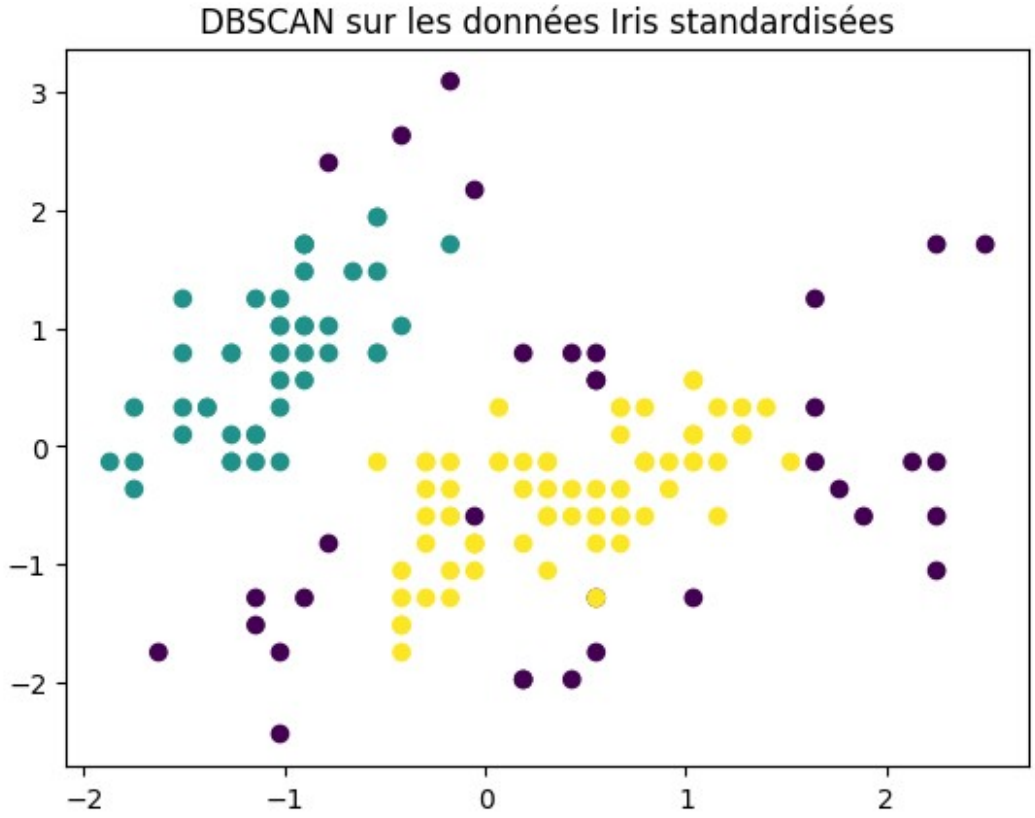


Clustering DBSCAN

```
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn import datasets

# Chargement des données Iris
iris = datasets.load_iris()
```

```
X = iris.data

# Mise à l'échelle des données
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Application de DBSCAN sur les données originales
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan.fit(X)
labels = dbscan.labels_

# Visualisation des clusters DBSCAN sur les données originales
plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.title("DBSCAN sur les données Iris 2e dimensions")
plt.show()

# Application de DBSCAN sur les données standardisées
dbscan_scaled = DBSCAN(eps=0.5, min_samples=5)
dbscan_scaled.fit(X_scaled)
labels_scaled = dbscan_scaled.labels_

# Visualisation des clusters DBSCAN sur les données standardisées
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels_scaled)
plt.title("DBSCAN sur les données Iris standardisées")
plt.show()
```


DBSCAN sur les données Iris 2e dimensions
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


DBSCAN sur les données Iris standardisées
3
2
1
0
−1
−2
−2
−1
0
1
2

Interpretation des résultats

Résumé des Méthodes de Clustering Appliquées au Jeu de Données Iris

Dans notre étude du jeu de données Iris, qui contient des mesures de 150 échantillons de fleurs d'Iris répartis en trois espèces, nous avons appliqué trois méthodes de clustering pour classifier les observations sans utiliser leurs étiquettes prédéfinies.

K-Means Clustering: La première méthode, K-Means, a partitionné les données en trois clusters basés sur la proximité géométrique des points. L'algorithme a été configuré pour diviser les données en trois groupes, correspondant au nombre connu de classes dans le jeu de données Iris. Les résultats ont montré des clusters distincts, ce qui suggère une bonne séparation des données selon les caractéristiques mesurées. Cette méthode est particulièrement efficace pour identifier des groupes bien séparés lorsque la structure des données est globulaire.

Clustering Hiérarchique La deuxième méthode, le Clustering Hiérarchique, a été visualisée à l'aide d'un dendrogramme. Ce dernier démontre la formation de clusters en fonction de la distance ou de la similitude entre les échantillons. En examinant le dendrogramme, nous pourrions inférer une division naturelle des données en trois clusters principaux, ce qui résonne avec la classification des espèces d'Iris. Le Clustering Hiérarchique est avantageux pour son interprétation visuelle qui permet de comprendre la composition et la hiérarchie des groupes.

DBSCAN Enfin, DBSCAN, une méthode de clustering basée sur la densité, a révélé des clusters en fonction de la densité locale des points, tout en identifiant les outliers. Les paramètres de distance maximale (eps) et le nombre minimal de points (min_samples) ont été ajustés pour optimiser la détection des clusters. Les graphiques obtenus montrent que DBSCAN peut capter les variations de densité entre les clusters et exclure efficacement les points aberrants. Cependant, cette méthode est sensible à la mise à l'échelle des caractéristiques, comme l'illustre la différence entre les résultats des données originales et standardisées.

Utilisation de la fonction k-NN

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

# Séparation des données en un jeu d'entraînement et un jeu de test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# Création et entraînement du modèle k-NN
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Prédiction sur le jeu de test
y_pred = knn.predict(X_test)
```

Matrice de Confusion

```
from sklearn.metrics import confusion_matrix
```

```python
# Matrice de confusion
conf_mat = confusion_matrix(y_test, y_pred)
print(conf_mat)
```

```
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

Conclusion

Chacune de ces méthodes offre une perspective unique sur les données. K-Means est robuste et facile à comprendre, le Clustering Hiérarchique fournit une vue détaillée de la structure des données, et DBSCAN est flexible dans la détection des formes de clusters non linéaires et dans la gestion des outliers.

En comparant les résultats avec les étiquettes de classe réelles des données Iris, nous avons pu évaluer la performance de chaque méthode.

Le choix de la méthode dépend fortement de la nature des données et de l'objectif de l'analyse. Pour le jeu de données Iris, bien que toutes les méthodes aient montré une capacité à grouper correctement les échantillons, K-Means et le Clustering Hiérarchique ont été les plus alignés avec la classification originale, tandis que DBSCAN a offert un aperçu intéressant sur la densité et la répartition des outliers.

Concernant la méthode des K-NN

La première classe a 19 instances qui ont été toutes correctement classées.

La deuxième classe a 13 instances qui ont été toutes correctement classées.

La troisième classe a 13 instances qui ont été toutes correctement classées.

Aucune instance n'a été mal classée, car il n'y a pas de valeurs autres que zéro en dehors de la diagonale principale. Cela signifie que le modèle a une précision de 100% sur l'ensemble de test utilisé, ce qui est une performance exceptionnellement bonne.