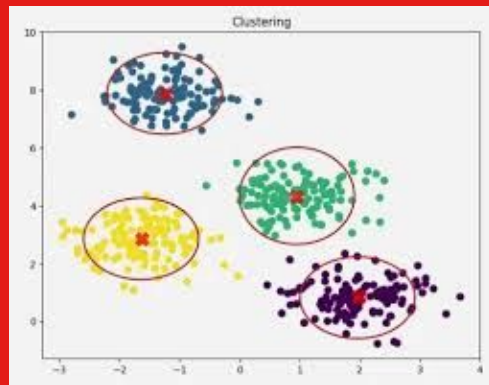


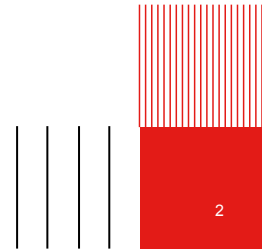
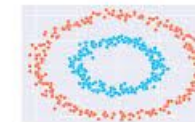
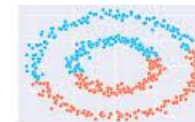
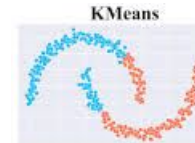
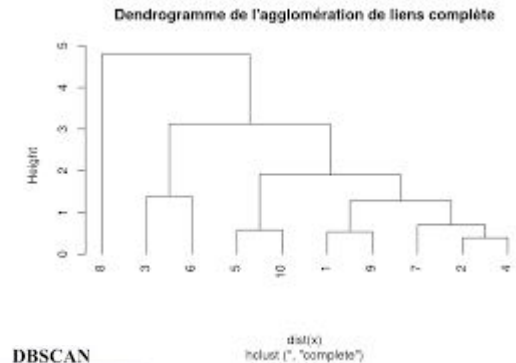
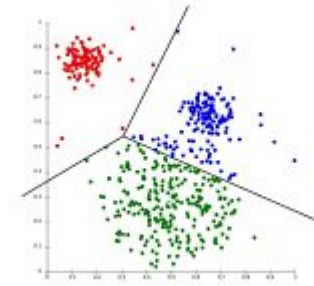
Clustering



Hassouna Mohamed Amine
Laforge Mateo

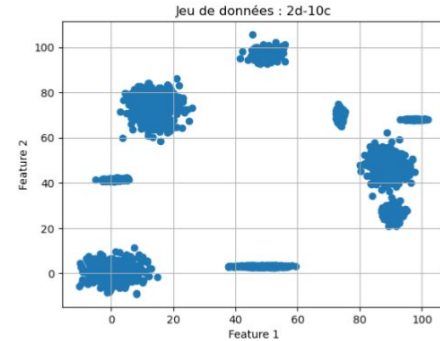
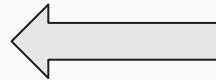
PLAN :

- Présentation du jeux de données
- Méthode des K-Means
- Méthode Aggléomératif
- Méthode DBSCAN
- Méthode HDBSCAN
 - Principe de fonctionnement et caractéristiques principales
 - Résultat Visuels
 - Automatisation de choix des paramètres
 - Avantages et limites de la méthode



Présentation du jeux de données

```
1 import os
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from scipy.io import arff
5 from sklearn.cluster import KMeans
6 from sklearn.preprocessing import StandardScaler
7 from sklearn.metrics import silhouette_score
8
9 dataset_name = 'smile3.arff'
10
11 def load_arff_data(file_path):
12     """Load numeric data from a .arff file and return as NumPy array."""
13     data, meta = arff.loadarff(file_path)
14     X = np.array([list(row[:-1]) for row in data], dtype=float)
15     return X
16
17 file_path = os.path.join('../dataset', 'artificial', dataset_name)
18 X = load_arff_data(file_path)
19 print("Shape du dataset :", X.shape)
20
21 scaler = StandardScaler()
22 X_scaled = scaler.fit_transform(X)
23
24 plt.scatter(X_scaled[:, 0], X_scaled[:, 1], s=10, c='blue')
25 plt.title(f"Visualisation du dataset {dataset_name}")
26 plt.xlabel("Dimension 1")
27 plt.ylabel("Dimension 2")
28 plt.show()
```



- ✗ mauvais choix de centres
- ✗ clusters mal séparés
- ✗ silhouette score plus faible
- ✗ interprétation incorrecte des résultats



Méthode des K-Means :

Principe : une méthode itérative de clustering partionnel

Objectif : minimiser la distance intra-cluster : distance moyennes des exemples au centre de leur cluster

Entrée : nombre de clusters fixé en Avance

Forme de clusters uniquement convexes et sphériques

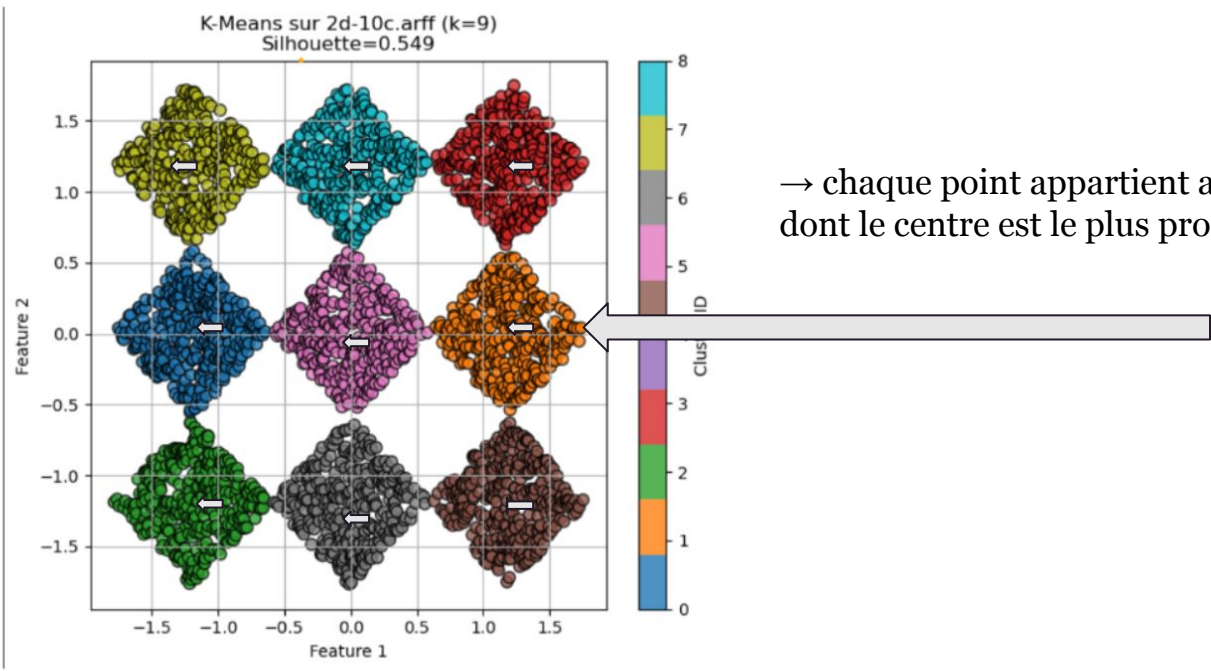
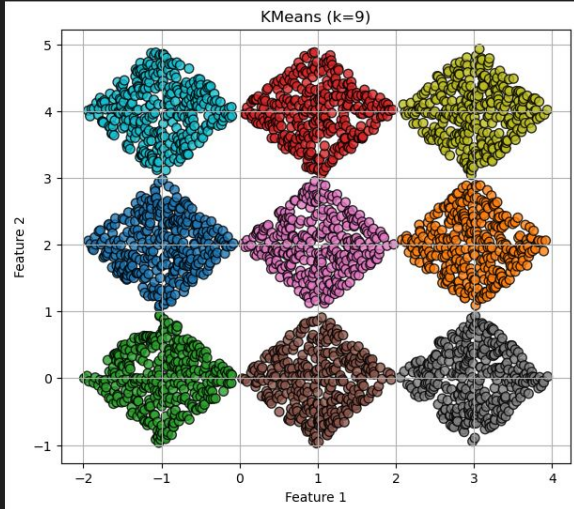


FIGURE 2.1 – Résultat du K-Means sur le dataset `diamond9.arff` avec $K = 9$



Automatisation de choix des paramètres

Shape du dataset : (3000, 2)



```
import numpy as np
|
|
from sklearn.cluster import KMeans
from Method import Method

def elbow_kmeans(data, k_max=10):
    """
    Trouve le meilleur k pour k-means en utilisant la méthode du coude.

    Paramètres:
    - data: numpy array ou pandas DataFrame, les données à clusteriser.
    - k_max: int, le nombre maximum de clusters à tester.

    Retourne:
    - kmeans: KMeans, instance d'object KMeans entraînée
    - meilleur_k: int, le k suggéré par la méthode du coude.
    - inerties: list, les inerties pour chaque k (pour tracer le graphique).
    """
    inerties = []
    kmeans_liste: list[KMeans] = []
    for k in range(1, k_max + 1):
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(data)
        kmeans_liste.append(kmeans)
        inerties.append(kmeans.inertia_)

    # Calcul des différences pour trouver le "coude"
    diffs = np.diff(inerties)
    diff_ratios = diffs[:-1] / diffs[1:]
    meilleur_k = np.argmax(diff_ratios) + 2 # +2 car diff commence à k=2
    kmeans: KMeans = kmeans_liste[meilleur_k-1]

    return kmeans, meilleur_k

class KMeansMethod(Method):

    def predict(self, X):
        kmean_inst, _ = elbow_kmeans(X, 10)
        labels = kmean_inst.predict(X)
        return labels

    def __repr__(self):
        return "k-means"
```

Comparaison des métriques de clustering sur diamond9.arff

+

+

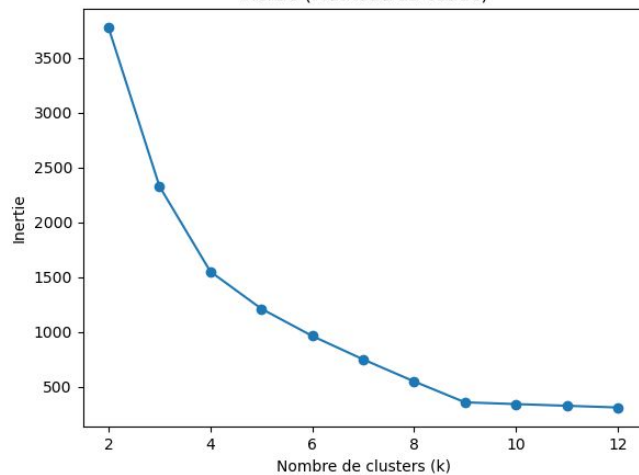
+

+

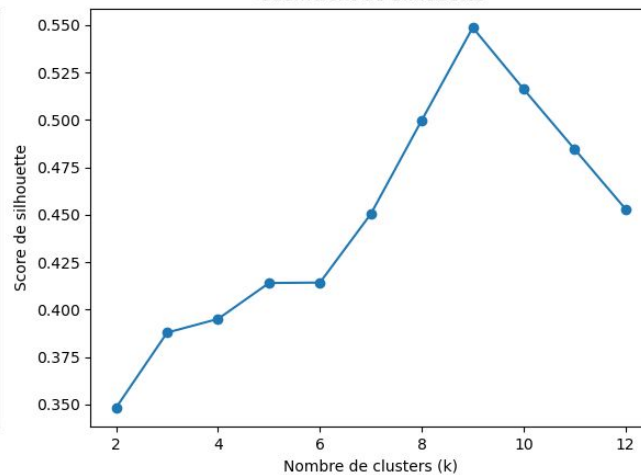
+

+

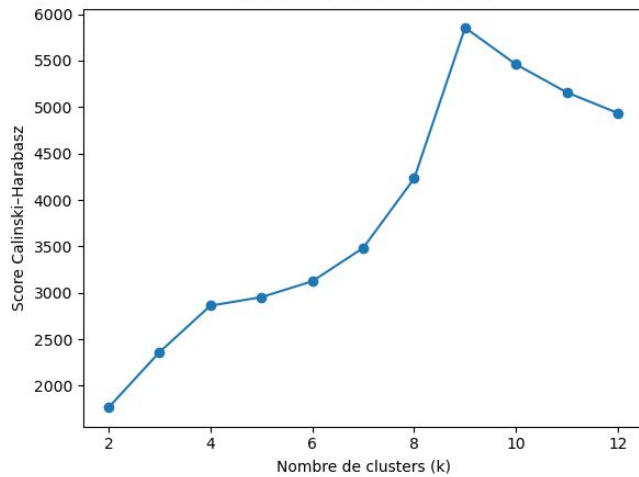
Inertie (Méthode du coude)



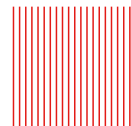
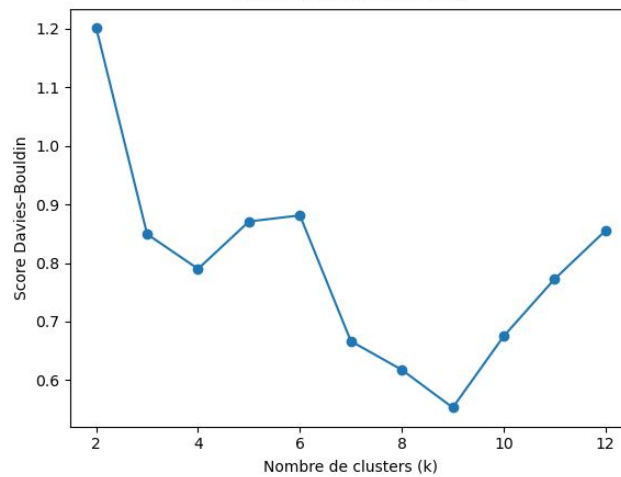
Coefficient de Silhouette



Indice de Calinski-Harabasz

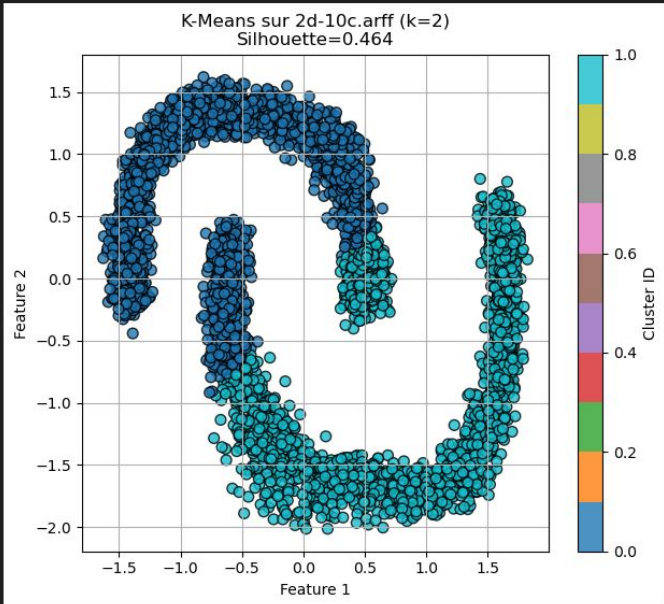


Indice de Davies-Bouldin

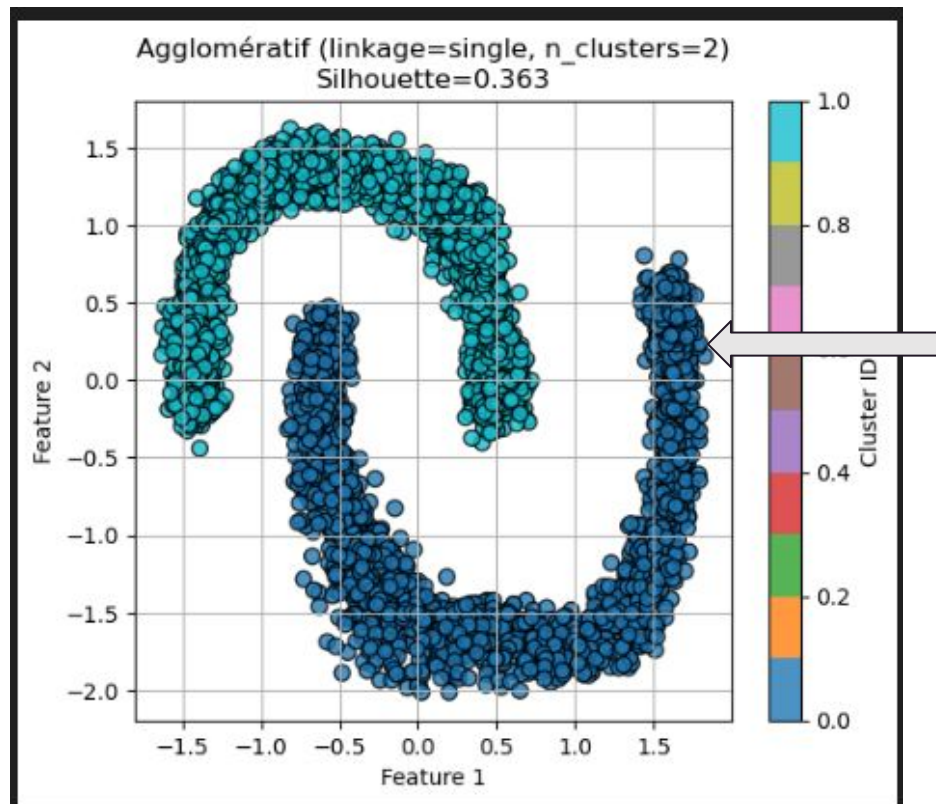


Limites : Forme de clusters uniquement convexes et sphériques

Exemple non Réussi de K-mean banana.arff k=2



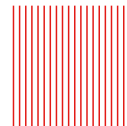
➡ Les barycentres des solutions ne font pas parties des clusters eux mêmes



Contrairement à K-Means :

- pas besoin de centres de clusters
- pas besoin de clusters convexes ou sphériques
- pas de notion d'inertie à minimiser

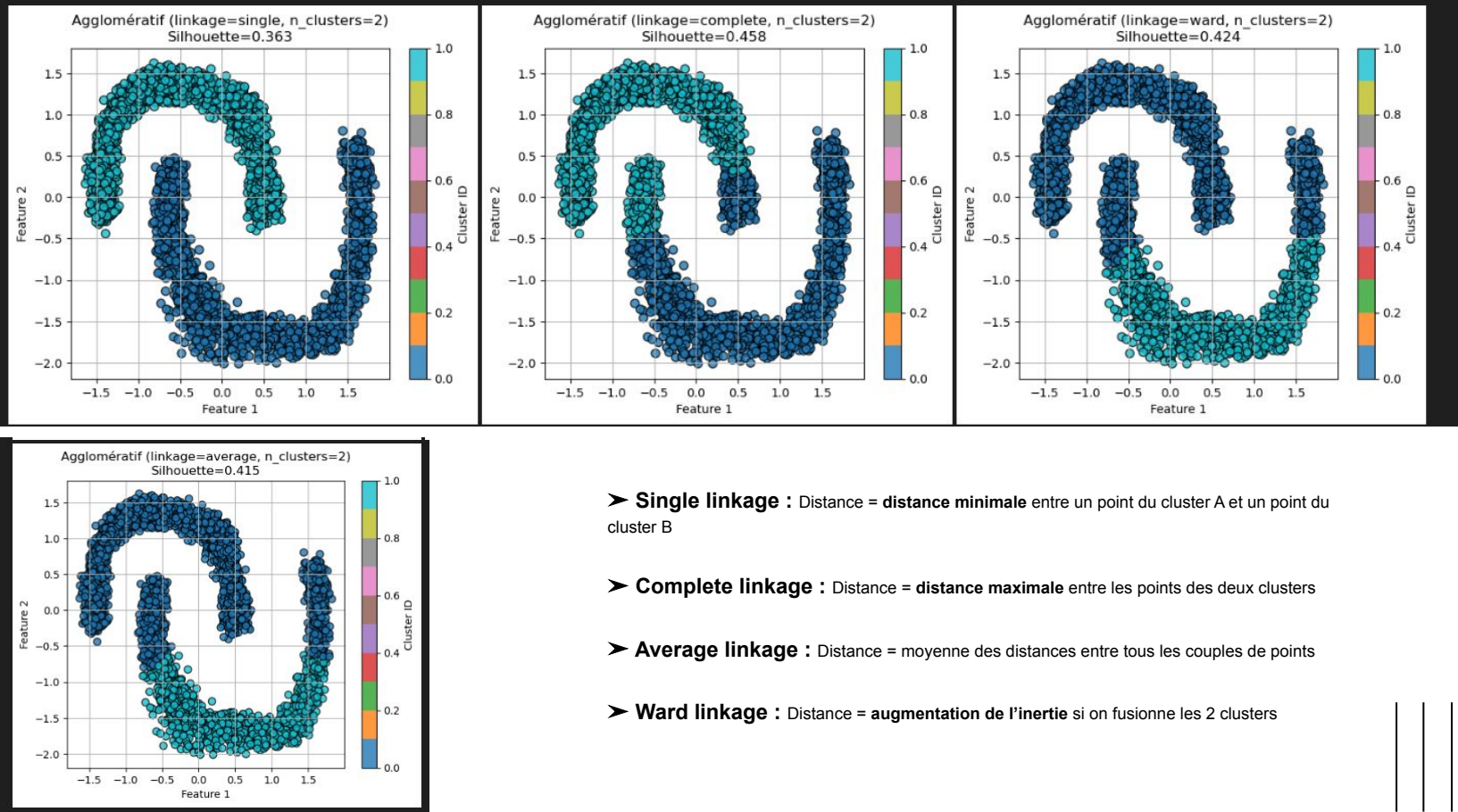
→ On part de **chaque point comme un cluster individuel**,
→ Puis on **fusionne progressivement** les clusters les plus proches,
Jusqu'à n'obtenir que le nombre de clusters souhaité.





Les Linkages (règles de fusions) :

comment mesurer la distance entre deux clusters ?



- **Single linkage** : Distance = **distance minimale** entre un point du cluster A et un point du cluster B
- **Complete linkage** : Distance = **distance maximale** entre les points des deux clusters
- **Average linkage** : Distance = moyenne des distances entre tous les couples de points
- **Ward linkage** : Distance = **augmentation de l'inertie** si on fusionne les 2 clusters

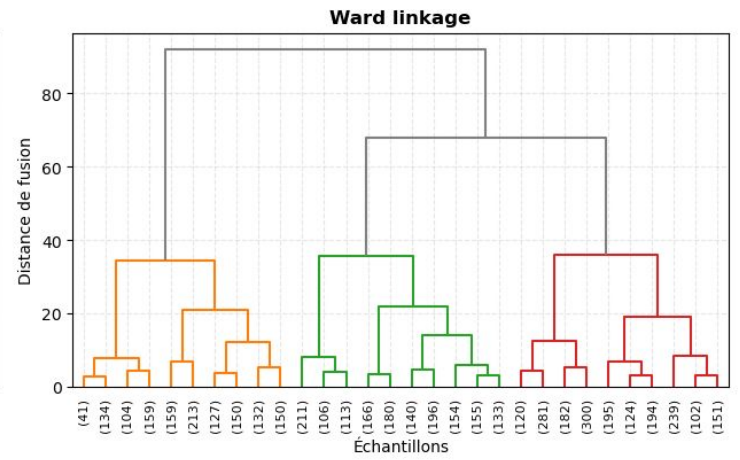
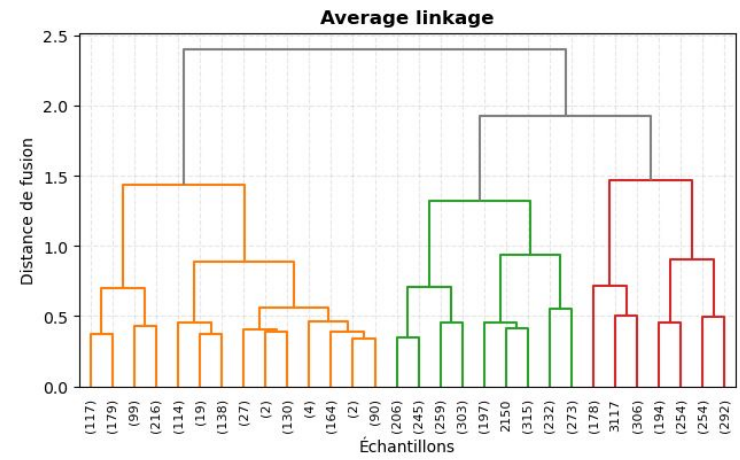
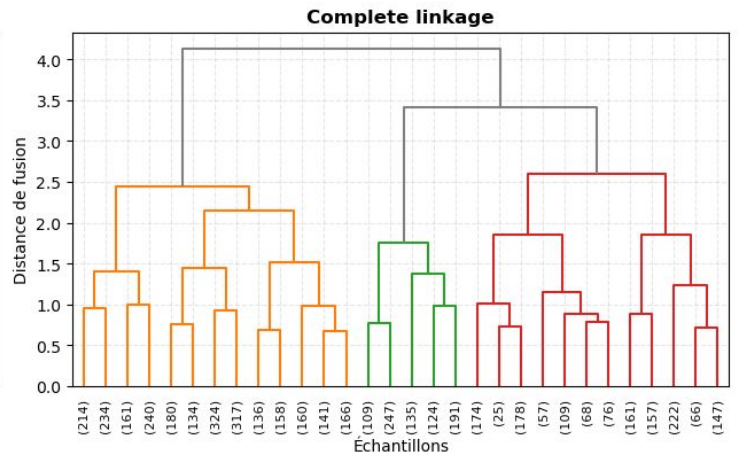
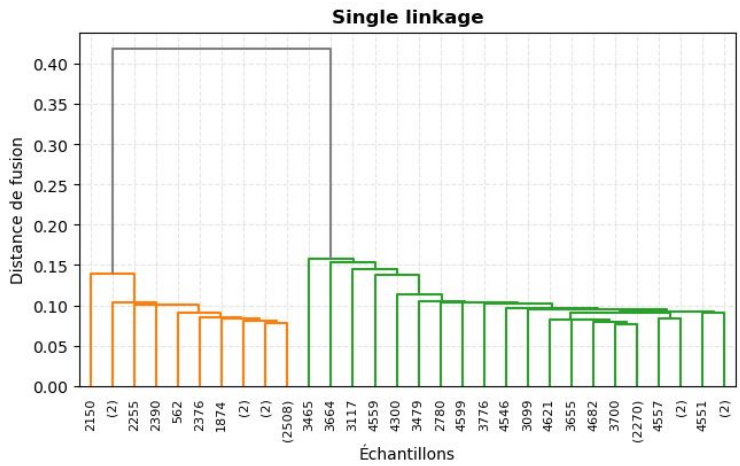




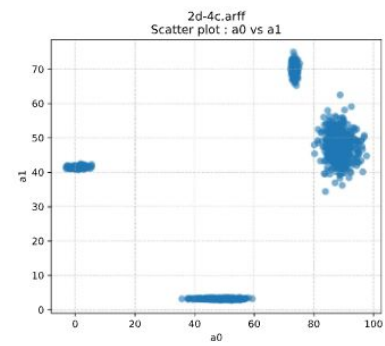
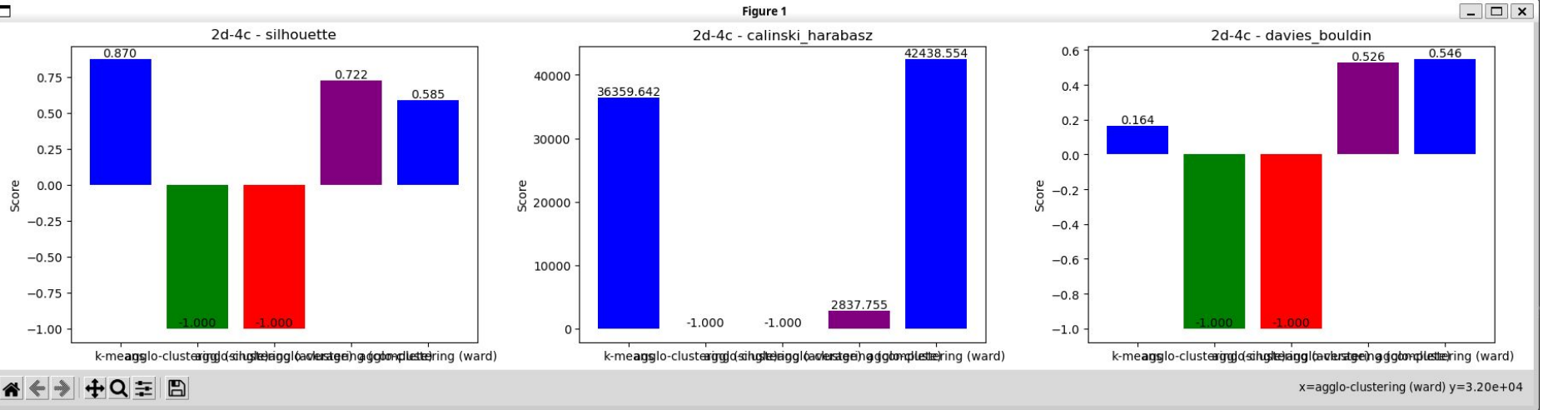
Le Dendrogramme:

L'arbre qui visualise les fusions successives des clusters.

Comparaison des méthodes agglomératives sur banana.arff

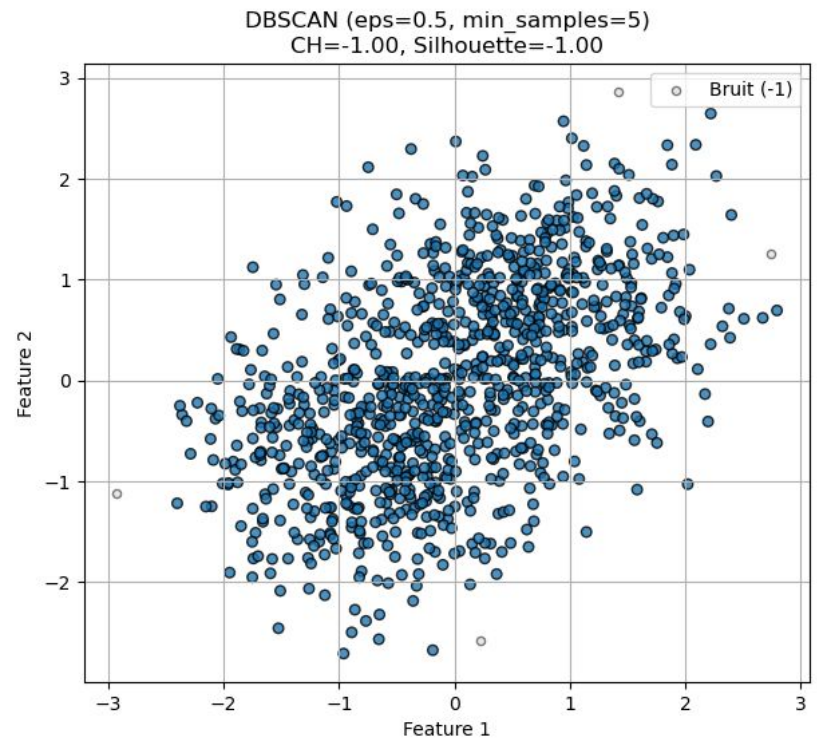
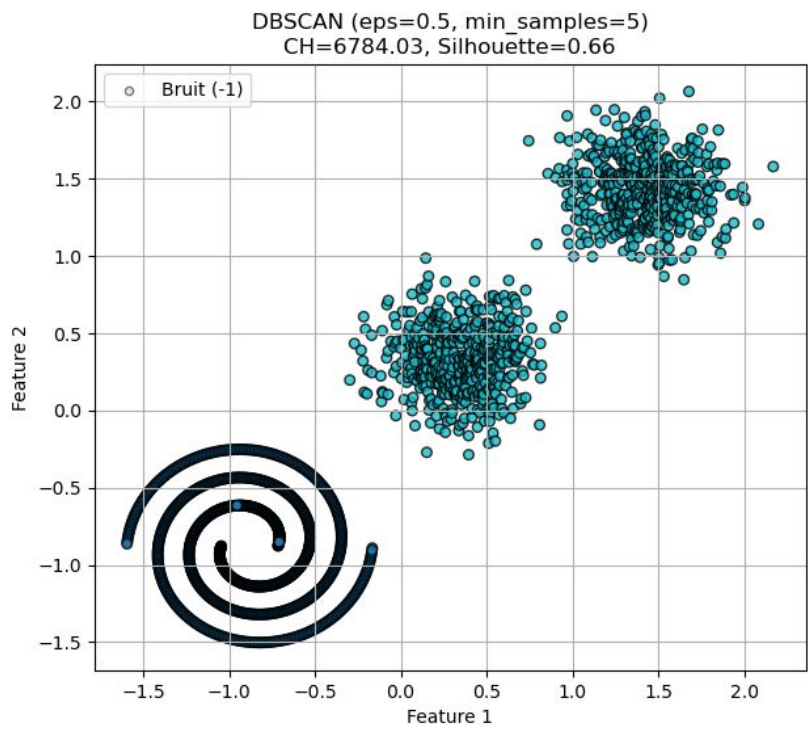


Etude Comparative:



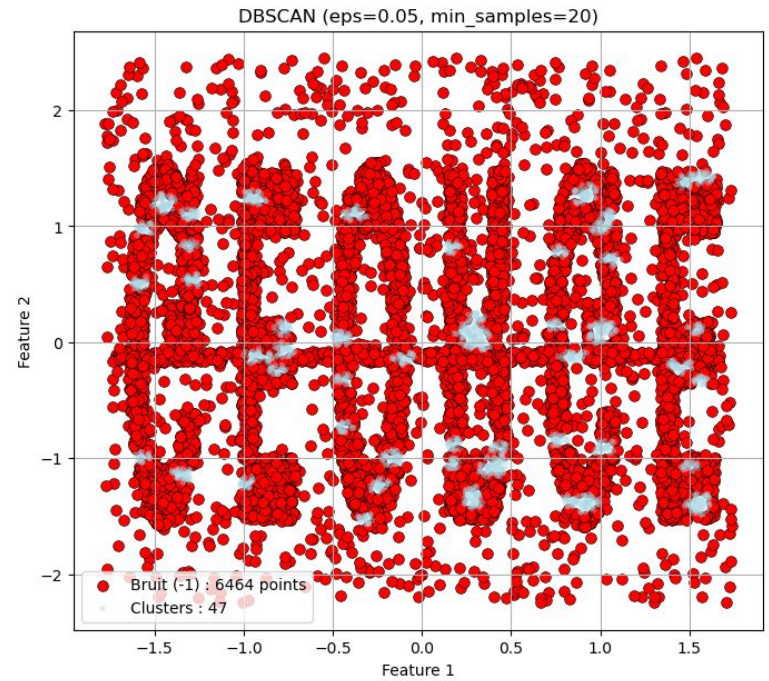
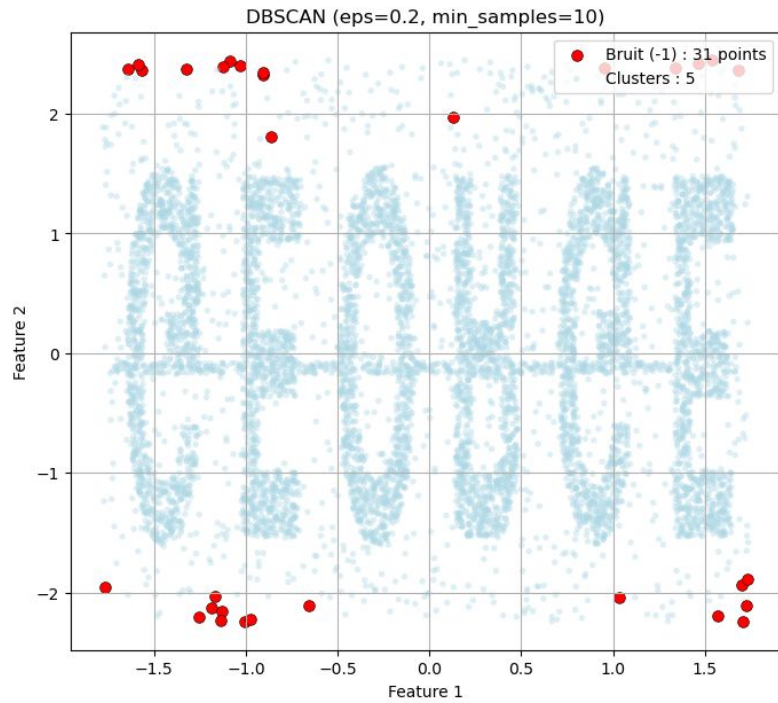


Méthode DBSCAN:





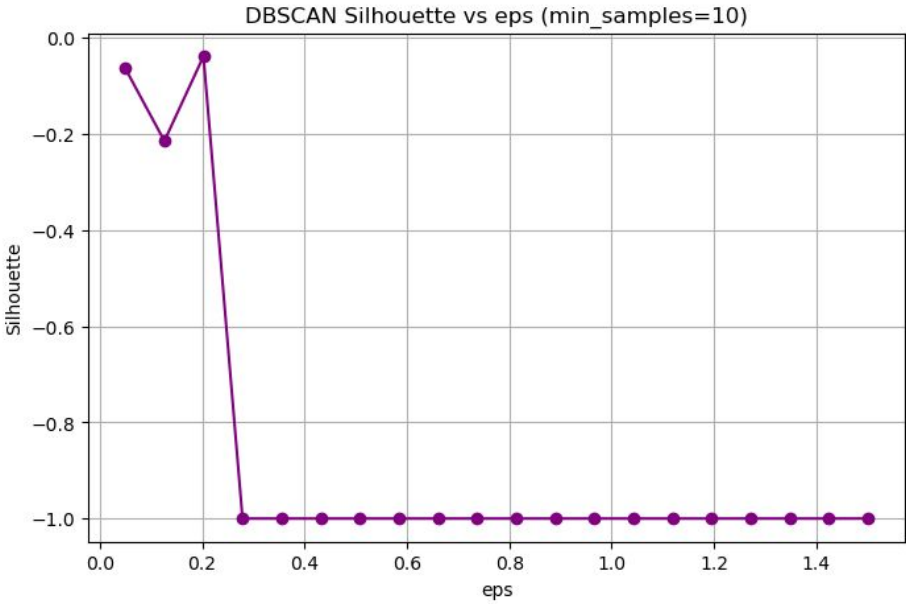
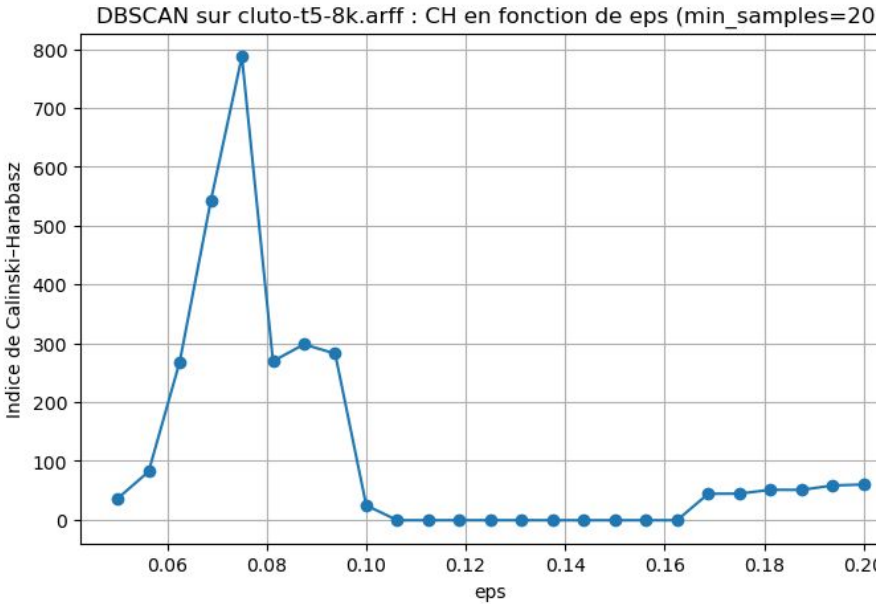
Choix de paramètres:





La distance Epsilon:

le rayon qui définit la distance maximale pour considérer qu'un point est voisin d'un autre. Plus ϵ est grand, plus DBSCAN fusionne de régions ensemble et forme des clusters plus larges.

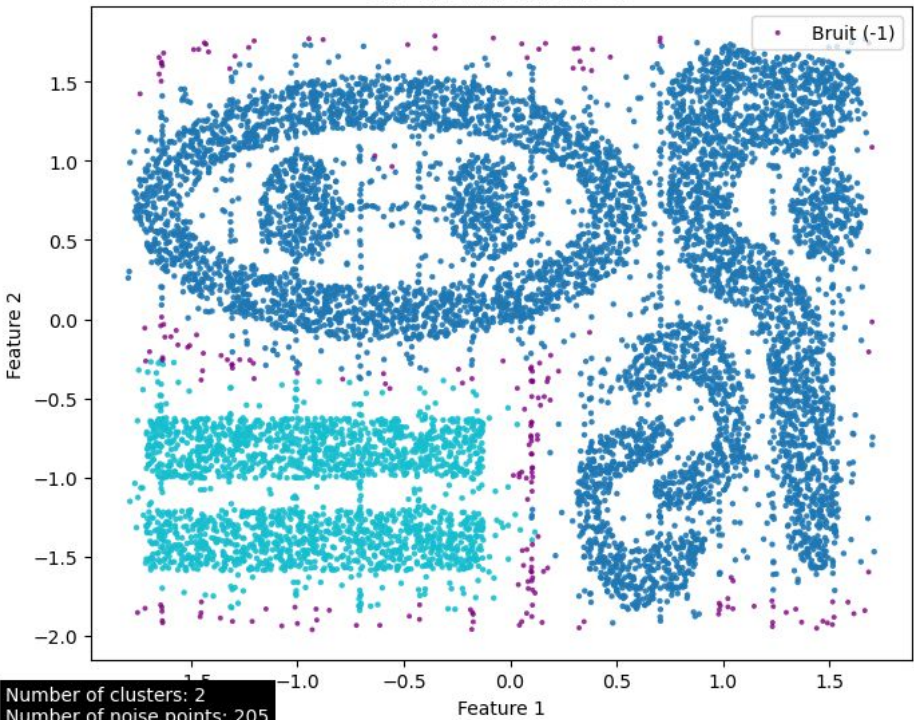




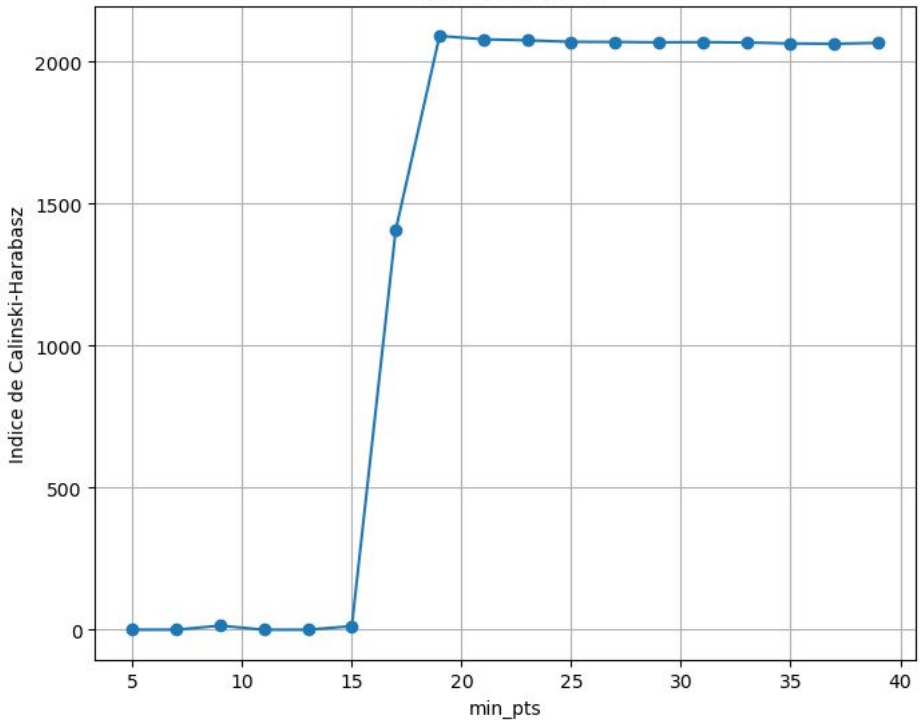
min_samples :

le nombre minimum de points nécessaires dans le voisinage ϵ pour qu'un point soit considéré comme un "core point". Plus min_samples est grand, plus DBSCAN exige une densité élevée pour former un cluster.

DBSCAN - eps=0.15, min_pts=19
CH=2091.4, clusters=2

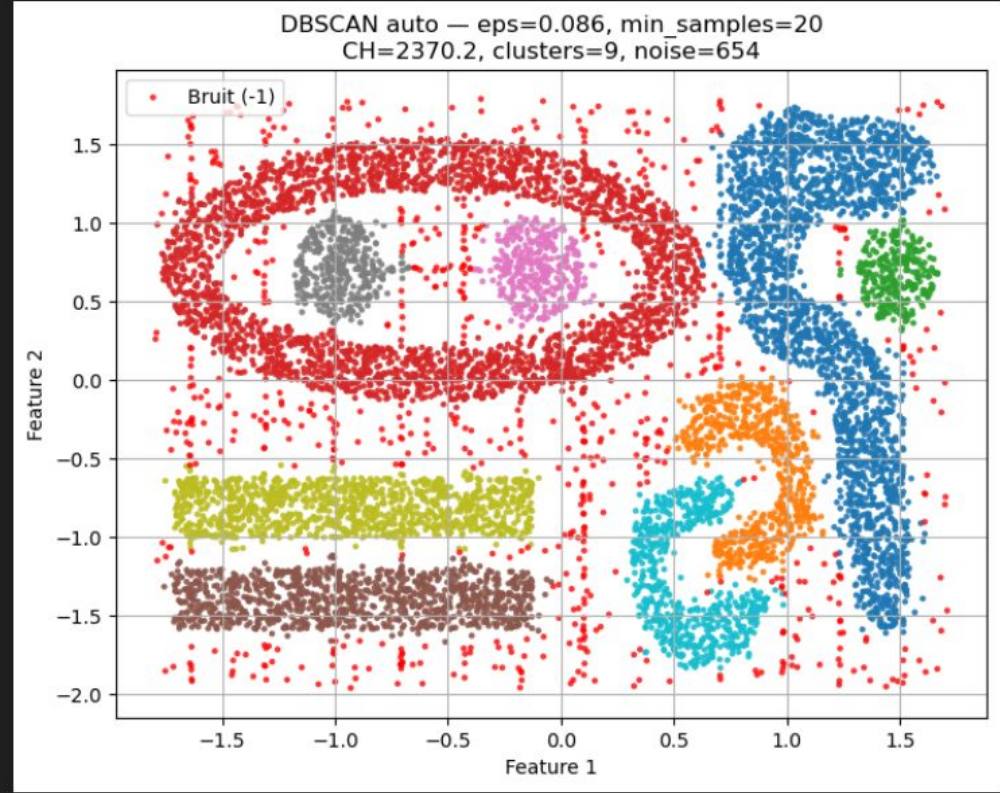


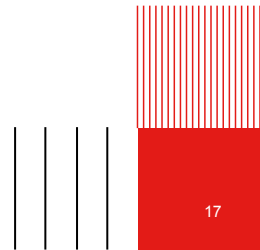
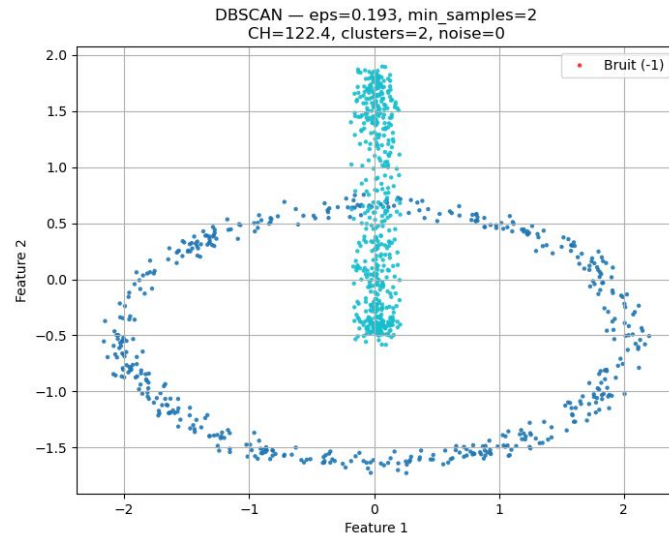
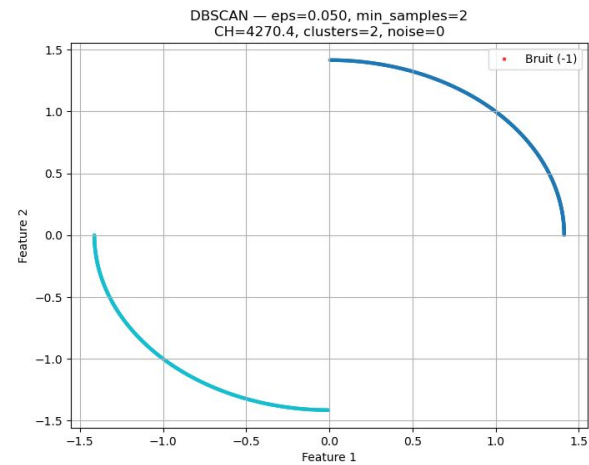
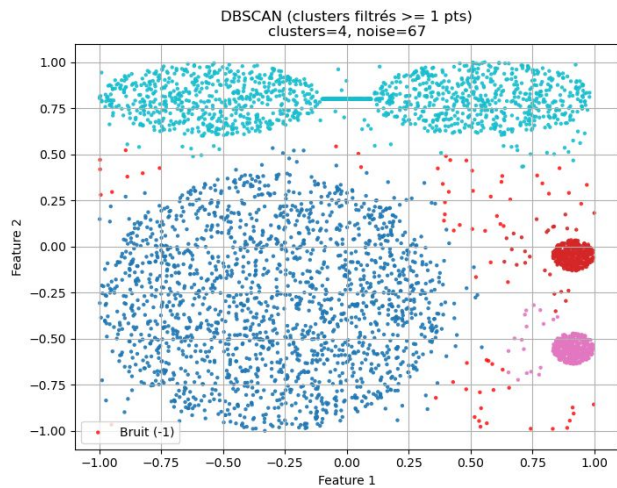
Calinski-Harabasz



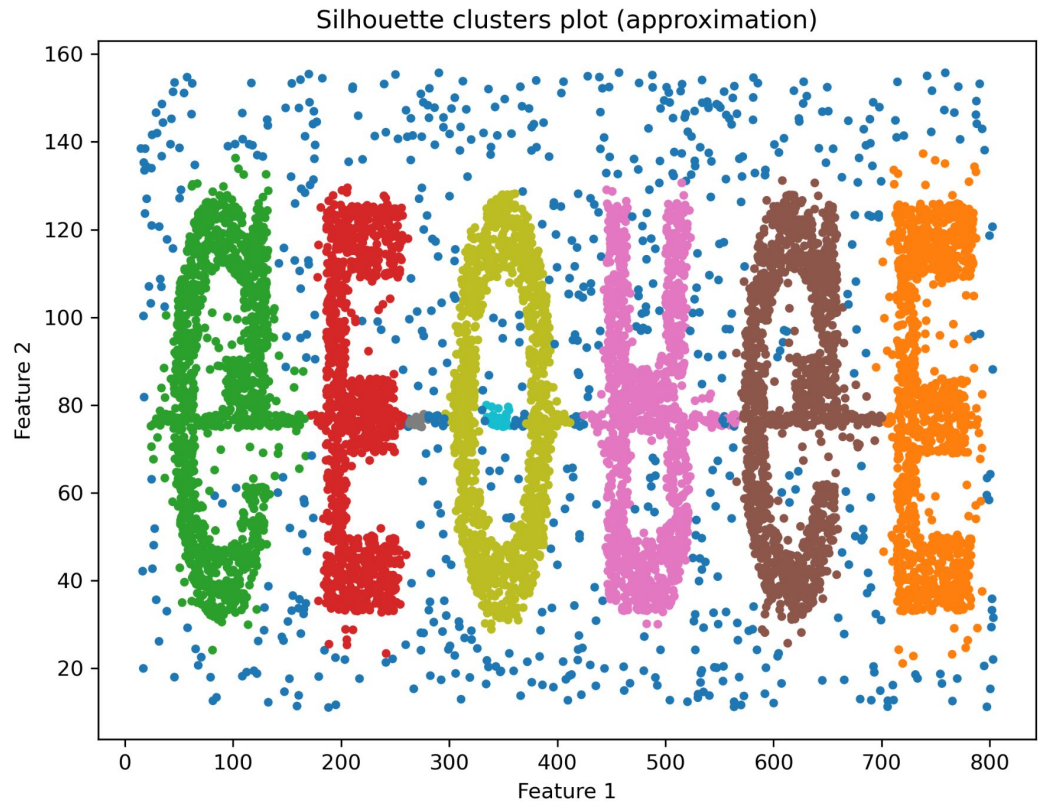
Automatisation des choix de paramètres:

Meilleurs paramètres (CH max) : $\text{eps}=0.086$, $\text{min_samples}=20$
Score CH = 2370.16
Nombre de clusters = 9, nombre de points de bruit = 654



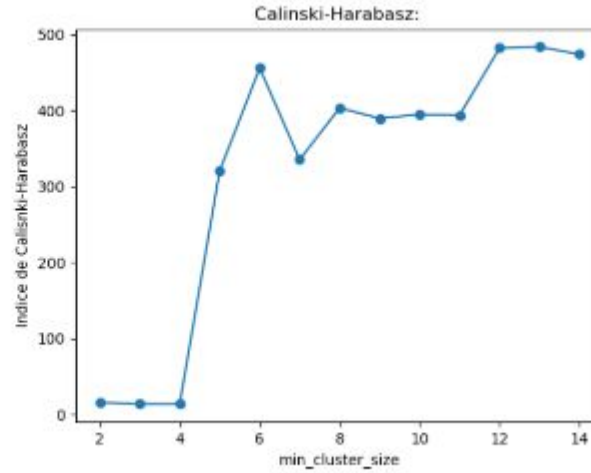
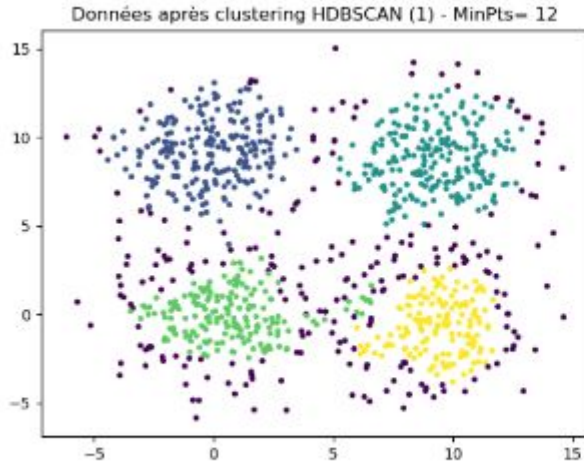


Méthode HDBSCAN:





Paramètres:

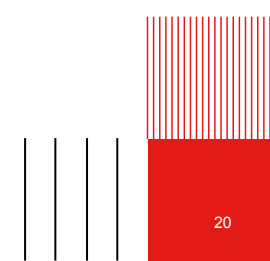
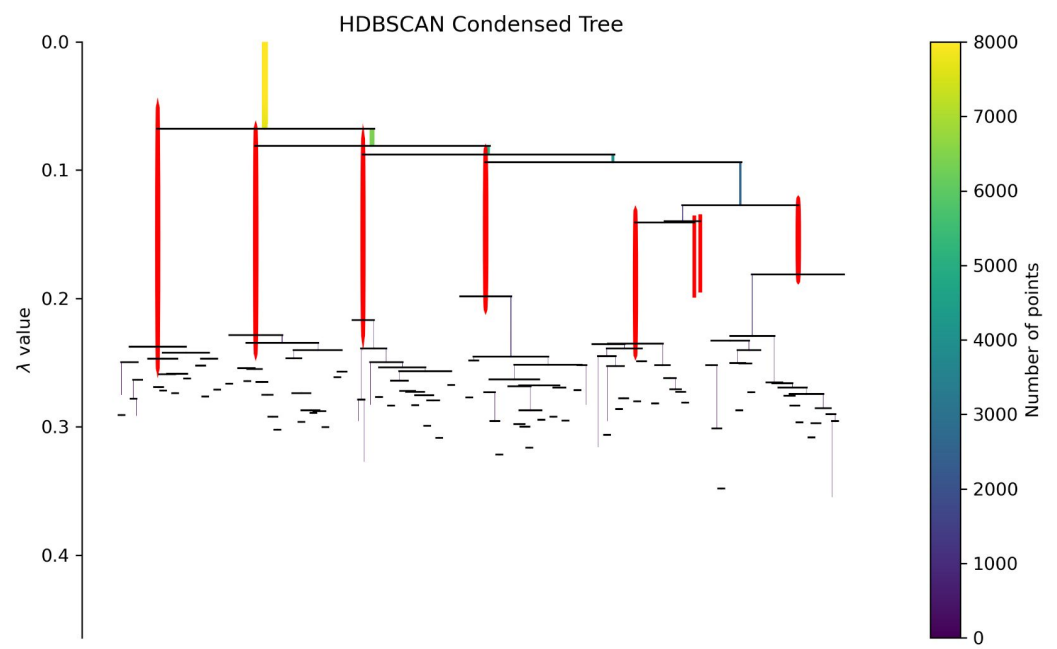


HDBSCAN utilise principalement le paramètre ***min_cluster_size***, qui contrôle la taille minimale des clusters et influence directement leur granularité.

Il utilise aussi ***min_samples***, qui ajuste la sensibilité à la densité : plus il est grand, plus l'algorithme devient conservateur et détecte davantage de bruit.



Condensed Tree de HDBSCAN cluto-t5-8k





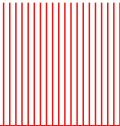
Comparaison des Méthodes HDBSCAN et DBSCAN:

Méthode	Davies-Bouldin	Calinski-Harabasz	Silhouette	Temps (ms)	Jeu de Données
Dbscan 6 eps, 8 min pts	1.76	1750	0.44	1750 ms	xclara
Hdbscan 6	1.57	6412	0.65	3702.18 ms	xclara
Dbscan 1 eps, 20 min	1.25	514.57	0.42	599.71 ms	sizes3
Hdbscan 10	1.56	461.86	0.48	447.11 ms	sizes3
Dbscan	0.31	125	0.51	2033.09 ms	long2
Hdbscan 7	1.99	119	0.29	2562.36 ms	long2
Dbscan 1.2 eps pour 8	1.49	381.46	0.47	1006.2 ms	sizes4
Hdbscan 8	1.43	433.62	0.49	2602.85 ms	sizes4

Par rapport à DBSCAN,
Plus robuste à la variation de paramètres
Gère bien les densités variables

• **DBSCAN** est simple à implémenter et fonctionne bien sur des clusters de densité uniforme, mais il est limité lorsque les données ont des structures de densité variable.

• **HDBSCAN** est plus flexible et performant pour gérer des clusters de densités variables, mais il est plus complexe à paramétrer et peut être plus lent.





Code source :

NB : le code implémenté fait voir plusieurs testes fait surtout ceux qui sont dans Rapport.ipynb dans le dossier TP1

projet Globale : <https://github.com/Clustering-Laforge-Hassouna/TP-Clustering>

partie Clustering:

<https://github.com/Clustering-Laforge-Hassouna/TP-Clustering/tree/main/TP1>

