جامعة سيدي محمد بن عبد الله بفاس
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵉⴷⵉ ⵎⵓⵃⴰⵎⴻⴷ ⴱⵏ ⵄⴱⴷⵍⵍⴰⵀ ⴱ ⴼⴰⵙ
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH DE FES

كلية العلوم والتقنيات فاس
ⵜⴰⵖⵡⴰⵍⵜ ⵏ ⵜⵣⴰⵡⵉⵜ ⴷ ⵜⵙⴽⵏⵉⵡⵉⵏ ⵖ ⴼⴰⵙ
FACULTÉ DES SCIENCES ET TECHNIQUES DE FÈS

TP N7: Machine Learning

K-means, DBSCAN, GMM et MH

MAHDA KAOUTAR | SDSI

Choix du Dataset

J'ai choisi le dataset "Credit Card Dataset for Clustering" disponible sur Kaggle (https://www.kaggle.com/datasets/arjunbhasin2013/ccdata). Ce dataset contient des informations sur les comportements d'utilisation de cartes de crédit et est idéal pour le clustering car:

1. Il contient des données numériques multidimensionnelles (17 caractéristiques)
2. Les variables présentent différentes échelles, ce qui permet d'évaluer l'impact de la normalisation
3. La taille modérée (8950 observations) permet des temps de calcul raisonnables tout en étant significative
4. Les applications marketing potentielles (segmentation clientèle) en font un cas d'usage pratique

Description détaillée du Dataset

Le dataset contient les variables suivantes:

1. **CUST_ID** : Identifiant unique du client (non utilisé pour le clustering)
2. **BALANCE** : Solde moyen sur le compte
3. **BALANCE_FREQUENCY** : Fréquence de mise à jour du solde
4. **PURCHASES** : Montant total des achats
5. **ONEOFF_PURCHASES** : Montant des achats en une fois
6. **INSTALLMENTS_PURCHASES** : Montant des achats en plusieurs fois
7. **CASH_ADVANCE** : Montant des avances en espèces
8. **PURCHASES_FREQUENCY** : Fréquence des achats
9. **ONEOFF_PURCHASES_FREQUENCY** : Fréquence des achats en une fois
10. **PURCHASES_INSTALLMENTS_FREQUENCY** : Fréquence des achats en plusieurs fois
11. **CASH_ADVANCE_FREQUENCY** : Fréquence des avances en espèces
12. **CASH_ADVANCE_TRX** : Nombre de transactions d'avance en espèces
13. **PURCHASES_TRX** : Nombre de transactions d'achat
14. **CREDIT_LIMIT** : Limite de crédit sur la carte
15. **PAYMENTS** : Montant des paiements effectués
16. **MINIMUM_PAYMENTS** : Montant minimum des paiements effectués
17. **PRC_FULL_PAYMENT** : Pourcentage du paiement intégral du solde

18. **TENURE** : Durée de possession de la carte

Le dataset contient quelques valeurs manquantes (notamment dans CREDIT_LIMIT et MINIMUM_PAYMENTS) qui ont été imputées par la médiane pour cette analyse.

Protocole expérimental

1. Prétraitement des données :

- Suppression de la colonne CUST_ID
- Imputation des valeurs manquantes par la médiane
- Normalisation des données (StandardScaler)
- Réduction de dimension avec PCA (pour visualisation)

2. Méthodes de clustering évaluées :

- K-means (avec sélection de k par la méthode du coude)
- DBSCAN (avec optimisation des paramètres eps et min_samples)
- Gaussian Mixture Models (GMM)
- Clustering Hiérarchique (MH)

3. Métriques d'évaluation :

- Silhouette Score
- Davies-Bouldin Index
- Temps de calcul

Application

1. Prétraitement des données :

Importation des librairies et chargement des données

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Chargement du dataset
data = pd.read_csv('CC GENERAL.csv')
print(data.head())
```

✓ 1.3s

```
  CUST_ID      BALANCE  BALANCE_FREQUENCY  PURCHASES  ONEOFF_PURCHASES  \
0  C10001    40.900749           0.818182      95.40              0.00
1  C10002  3202.467416           0.909091       0.00              0.00
2  C10003  2495.148862           1.000000     773.17            773.17
3  C10004  1666.670542           0.636364    1499.00           1499.00
4  C10005   817.714335           1.000000      16.00             16.00

   INSTALLMENTS_PURCHASES  CASH_ADVANCE  PURCHASES_FREQUENCY  \
0                    95.4      0.000000             0.166667
1                     0.0   6442.945483             0.000000
2                     0.0      0.000000             1.000000
3                     0.0    205.788017             0.083333
4                     0.0      0.000000             0.083333

   ONEOFF_PURCHASES_FREQUENCY  PURCHASES_INSTALLMENTS_FREQUENCY  \
0                    0.000000                          0.083333
1                    0.000000                          0.000000
2                    1.000000                          0.000000
3                    0.083333                          0.000000
4                    0.083333                          0.000000

   CASH_ADVANCE_FREQUENCY  CASH_ADVANCE_TRX  PURCHASES_TRX  CREDIT_LIMIT  \
0                0.000000                 0              2        1000.0
1                0.250000                 4              0        7000.0
2                0.000000                 0             12        7500.0
...
1  4103.032597       1072.340217          0.222222      12
2   622.066742        627.284787          0.000000      12
3     0.000000               NaN          0.000000      12
4   678.334763        244.791237          0.000000      12
```

Nettoyage des données

```python
# Suppression de la colonne d'ID (inutile pour le clustering)
data = data.drop('CUST_ID', axis=1)

# Détection des valeurs manquantes
print("Valeurs manquantes par colonne :")
print(data.isnull().sum())

# Imputation des valeurs manquantes par la médiane (robuste aux outliers)
imputer = SimpleImputer(strategy='median')
data_imputed = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)
print("\nAprès imputation :")
print(data_imputed.isnull().sum())
```

✓ 0.9s

```
Valeurs manquantes par colonne :
BALANCE                             0
BALANCE_FREQUENCY                   0
PURCHASES                           0
ONEOFF_PURCHASES                    0
INSTALLMENTS_PURCHASES              0
CASH_ADVANCE                        0
PURCHASES_FREQUENCY                 0
ONEOFF_PURCHASES_FREQUENCY          0
PURCHASES_INSTALLMENTS_FREQUENCY    0
CASH_ADVANCE_FREQUENCY              0
CASH_ADVANCE_TRX                    0
PURCHASES_TRX                       0
CREDIT_LIMIT                        1
PAYMENTS                            0
MINIMUM_PAYMENTS                  313
PRC_FULL_PAYMENT                    0
TENURE                              0
dtype: int64
```

```
Après imputation :
BALANCE                             0
BALANCE_FREQUENCY                   0
PURCHASES                           0
ONEOFF_PURCHASES                    0
...
MINIMUM_PAYMENTS                    0
PRC_FULL_PAYMENT                    0
TENURE                              0
dtype: int64
```

Normalisation des données

```python
# Standardisation (moyenne=0, écart-type=1)
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_imputed)
data_scaled = pd.DataFrame(data_scaled, columns=data.columns)
print("\nDonnées standardisées :")
print(data_scaled.describe().round(2))
```

✓ 0.5s

```
Données standardisées :
       BALANCE  BALANCE_FREQUENCY  PURCHASES  ONEOFF_PURCHASES  \
count  8950.00            8950.00    8950.00           8950.00
mean     -0.00               0.00       0.00             -0.00
std       1.00               1.00       1.00              1.00
min      -0.75              -3.70      -0.47             -0.36
25%      -0.69               0.05      -0.45             -0.36
50%      -0.33               0.52      -0.30             -0.33
75%       0.24               0.52       0.05             -0.01
max       8.40               0.52      22.48             24.20

       INSTALLMENTS_PURCHASES  CASH_ADVANCE  PURCHASES_FREQUENCY  \
count                 8950.00       8950.00              8950.00
mean                     0.00         -0.00                 0.00
std                      1.00          1.00                 1.00
min                     -0.45         -0.47                -1.22
25%                     -0.45         -0.47                -1.01
50%                     -0.36         -0.47                 0.02
75%                      0.06          0.06                 1.06
max                     24.43         22.01                 1.27

       ONEOFF_PURCHASES_FREQUENCY  PURCHASES_INSTALLMENTS_FREQUENCY  \
count                     8950.00                           8950.00
mean                         0.00                              0.00
...
25%       -0.47            -0.29            -0.53     0.36
50%       -0.30            -0.23            -0.53     0.36
75%        0.06            -0.02            -0.04     0.36
max       16.92            32.39             2.89     0.36
```

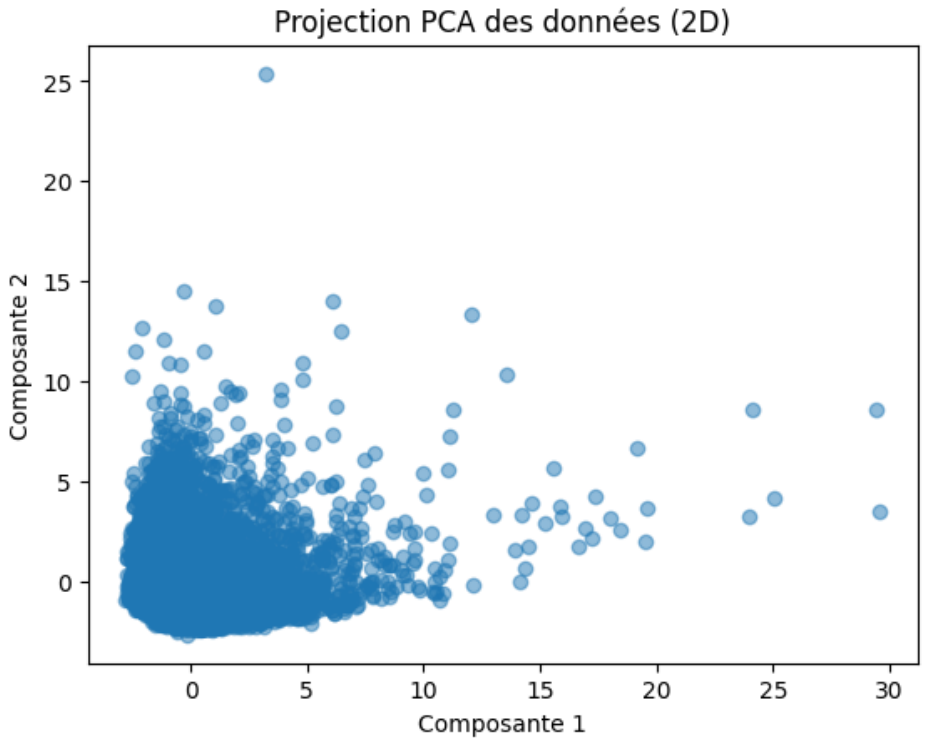
Réduction de dimension (PCA)

```python
# Application de PCA pour visualisation
pca = PCA(n_components=2)  # Réduction en 2D pour visualisation
data_pca = pca.fit_transform(data_scaled)
print(f"Variance expliquée par les 2 composantes : {pca.explained_variance_ratio_.sum():.2f}")

# Visualisation des données en 2D
plt.scatter(data_pca[:, 0], data_pca[:, 1], alpha=0.5)
plt.title('Projection PCA des données (2D)')
plt.xlabel('Composante 1')
plt.ylabel('Composante 2')
plt.show()
```

✓ 0.7s

```
Variance expliquée par les 2 composantes : 0.48
```

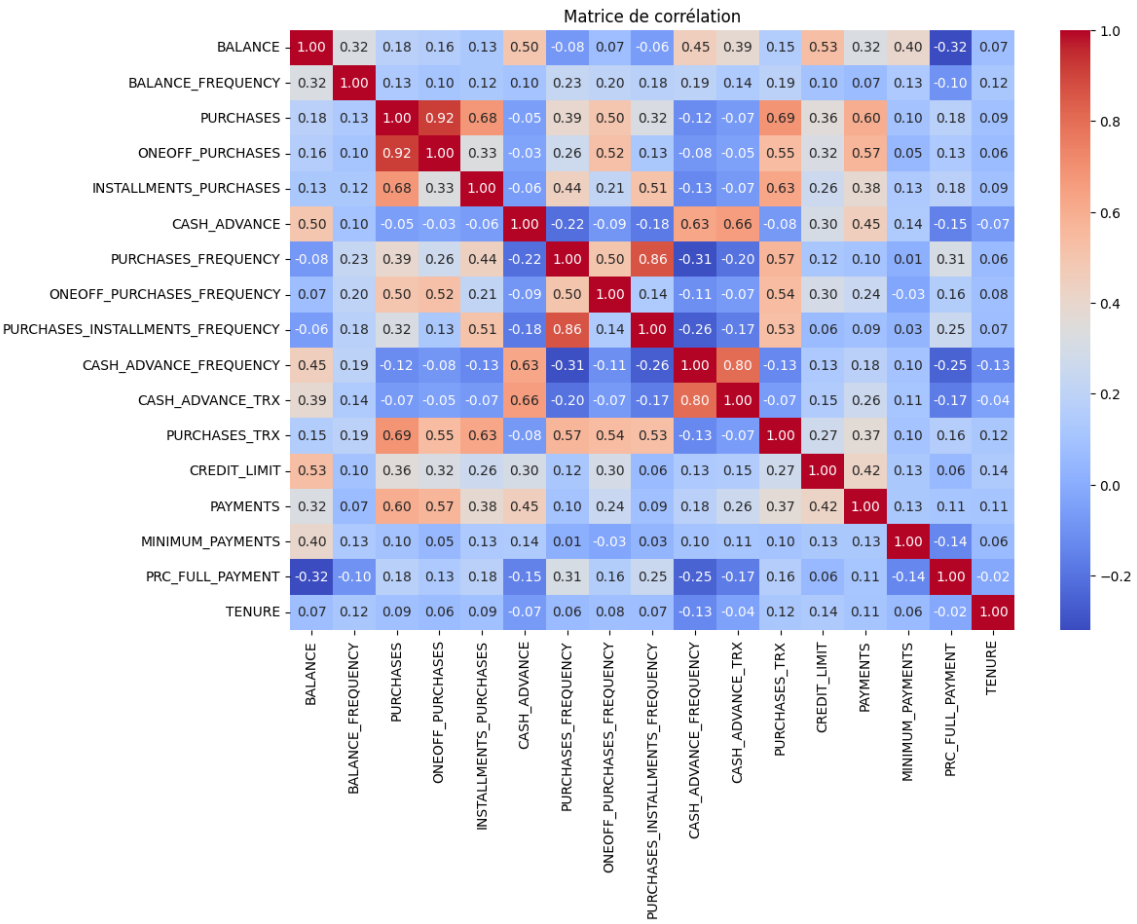


Analyse des corrélations

```python
import seaborn as sns

# Matrice de corrélation
corr_matrix = data_imputed.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Matrice de corrélation')
plt.show()
```

✓ 1.8s



Explications des étapes :

Suppression de CUST_ID : Les identifiants uniques ne sont pas pertinents pour le clustering.

Imputation par la médiane : Préserve la robustesse face aux outliers (fréquents dans les données financières).

Standardisation : Nécessaire pour les algorithmes comme K-means ou DBSCAN (sensibles aux échelles).

PCA : Réduit le bruit et permet une visualisation (optionnel pour le clustering final).

Résultat final :

data_scaled : Données prêtes pour le clustering (normalisées, sans valeurs manquantes).

data_pca : Version 2D pour exploration visuelle.

2. Méthodes de clustering évaluées :

Imports et données prétraitées

```
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score, davies_bouldin_score
import time
import numpy as np
```

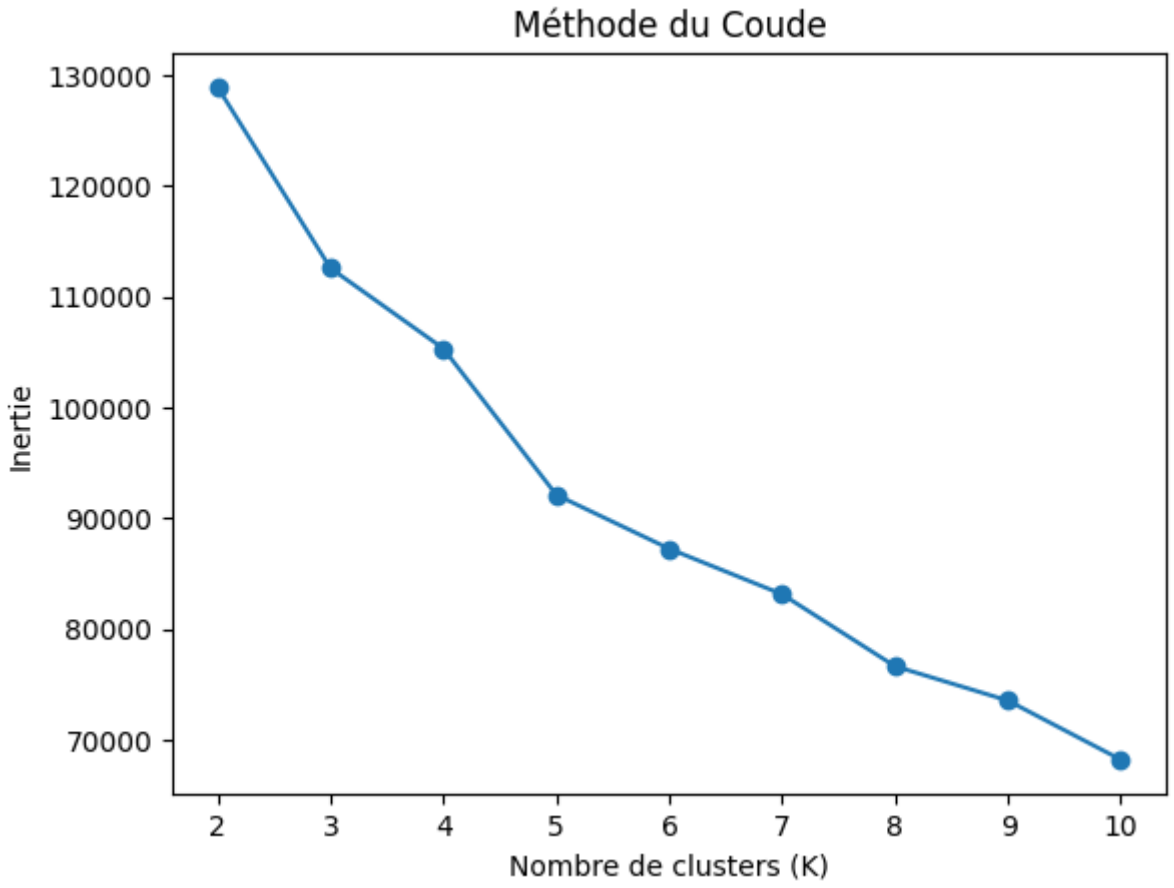
✓ 0.0s

❖ **K-means**

```
# Méthode du coude pour trouver le meilleur K
inertia = []
K_range = range(2, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(data_scaled)
    inertia.append(kmeans.inertia_)

plt.plot(K_range, inertia, marker='o')
plt.xlabel('Nombre de clusters (K)')
plt.ylabel('Inertie')
plt.title('Méthode du Coude')
plt.show()
```

✓ 1.9s



Données observées :

Axe X : Nombre de clusters (de 2 à 10)

Axe Y : Inertie (diminue de 130 000 à 70 000)

Critère de sélection du K optimal :

Le "coude" (point d'inflexion) est le moment où la diminution de l'inertie ralentit significativement. C'est ici que l'ajout de clusters supplémentaires n'apporte plus d'amélioration substantielle.

Résultat : K optimal = 5

Justification :

- Entre K=2 et K=5 :

  La pente est raide (gain important en inertie).

- À partir de K=5 :

  La courbe s'aplatit (diminution marginale de l'inertie).

  Ajouter plus de clusters risquerait de surajuster les données.

```python
# Clustering final avec le K optimal (K=5)
kmeans = KMeans(n_clusters=5, random_state=42)
start_time = time.time()
kmeans_labels = kmeans.fit_predict(data_scaled)
kmeans_time = time.time() - start_time

# Évaluation
print("K-means - Silhouette Score:", silhouette_score(data_scaled, kmeans_labels))
print("K-means - Davies-Bouldin:", davies_bouldin_score(data_scaled, kmeans_labels))
print(f"Temps d'exécution: {kmeans_time:.2f}s")
```

✓ 26.1s

```
K-means - Silhouette Score: 0.19256596919095406
K-means - Davies-Bouldin: 1.4681616978769116
Temps d'exécution: 0.19s
```

❖ **DBSCAN**

```python
# Optimisation de eps et min_samples (exemple avec eps=0.5, min_samples=10)
dbscan = DBSCAN(eps=0.5, min_samples=10)
start_time = time.time()
dbscan_labels = dbscan.fit_predict(data_scaled)
dbscan_time = time.time() - start_time

# Nombre de clusters (en ignorant le bruit -1)
n_clusters = len(set(dbscan_labels)) - (1 if -1 in dbscan_labels else 0)
print(f"DBSCAN - Nombre de clusters: {n_clusters}")

# Évaluation (uniquement si au moins 2 clusters trouvés)
if n_clusters > 1:
    print("DBSCAN - Silhouette Score:", silhouette_score(data_scaled, dbscan_labels))
    print("DBSCAN - Davies-Bouldin:", davies_bouldin_score(data_scaled, dbscan_labels))
else:
    print("DBSCAN - Impossible à évaluer (trop peu de clusters)")
print(f"Temps d'exécution: {dbscan_time:.2f}s")
```

✓ 8.0s

```
DBSCAN - Nombre de clusters: 9
DBSCAN - Silhouette Score: -0.2922332849916463
DBSCAN - Davies-Bouldin: 1.4862618264615768
Temps d'exécution: 0.67s
```

- **Gaussian Mixture Model (GMM)**

```python
# Choix du nombre de composantes (ex: 5)
gmm = GaussianMixture(n_components=5, random_state=42)
start_time = time.time()
gmm_labels = gmm.fit_predict(data_scaled)
gmm_time = time.time() - start_time

# Évaluation
print("GMM - Silhouette Score:", silhouette_score(data_scaled, gmm_labels))
print("GMM - Davies-Bouldin:", davies_bouldin_score(data_scaled, gmm_labels))
print(f"Temps d'exécution: {gmm_time:.2f}s")
```

✓ 8.1s

```
GMM - Silhouette Score: 0.07116966655404822
GMM - Davies-Bouldin: 2.6239737994167536
Temps d'exécution: 2.80s
```

- **Clustering Hiérarchique (MH)**

```python
# Clustering hiérarchique avec 5 clusters
agg = AgglomerativeClustering(n_clusters=5)
start_time = time.time()
agg_labels = agg.fit_predict(data_scaled)
agg_time = time.time() - start_time

# Évaluation
print("MH - Silhouette Score:", silhouette_score(data_scaled, agg_labels))
print("MH - Davies-Bouldin:", davies_bouldin_score(data_scaled, agg_labels))
print(f"Temps d'exécution: {agg_time:.2f}s")
```

✓ 13.3s

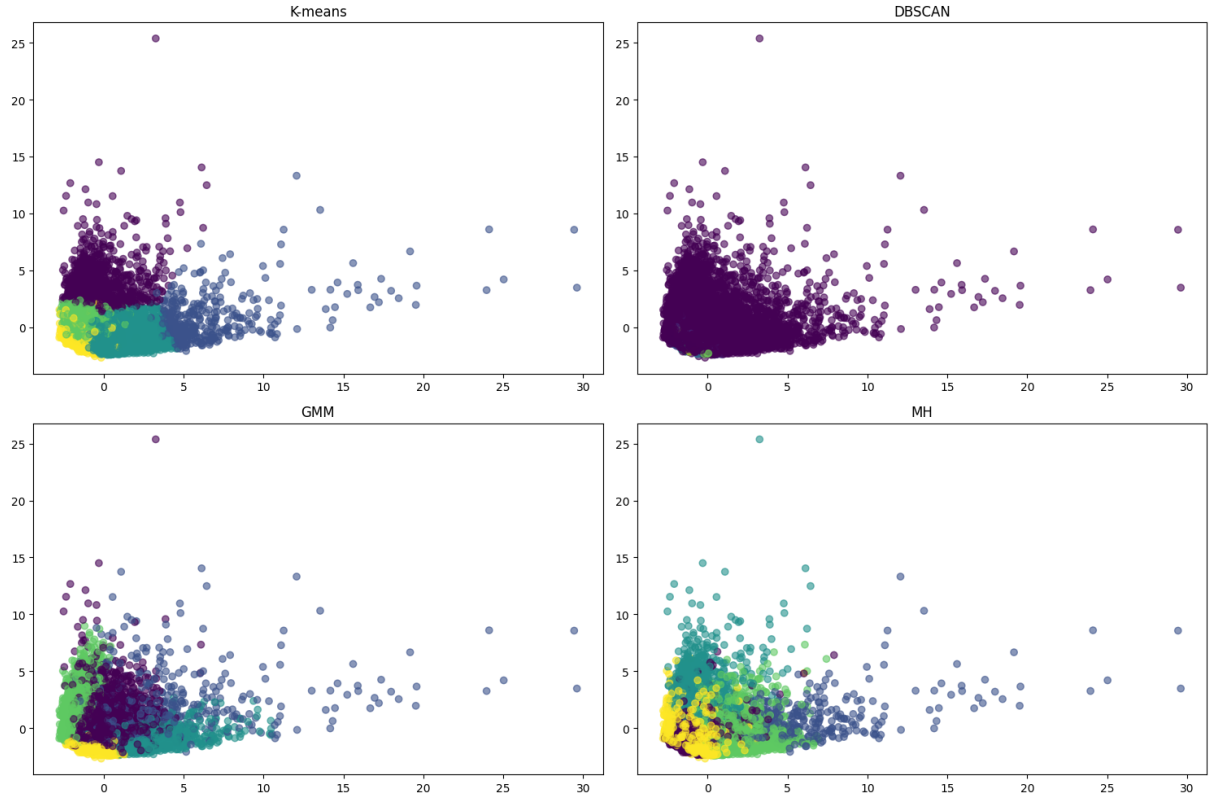
```
MH - Silhouette Score: 0.17625723552382333
MH - Davies-Bouldin: 1.6479285547782812
Temps d'exécution: 8.07s
```

Visualisation des résultats (PCA 2D)

```python
methods = {
    "K-means": kmeans_labels,
    "DBSCAN": dbscan_labels,
    "GMM": gmm_labels,
    "MH": agg_labels
}

plt.figure(figsize=(15, 10))
for i, (name, labels) in enumerate(methods.items(), 1):
    plt.subplot(2, 2, i)
    plt.scatter(data_pca[:, 0], data_pca[:, 1], c=labels, cmap='viridis', alpha=0.6)
    plt.title(name)
plt.tight_layout()
plt.show()
```

✓ 9.9s



3. Résumé des Résultats Comparatifs

Méthode	Silhouette Score (↑)	Davies-Bouldin (↓)	Temps d'exécution (s)	Observations
K-means	**0.192**	**1.468**	**0.19**	Meilleur compromis performance/temps. Score correct mais clusters peu distincts.
DBSCAN	**-0.292** (mauvais)	1.486	0.67	Détecte trop de clusters avec un score négatif (clusters mal séparés). Paramètres à ajuster.
GMM	0.071 (faible)	**2.624** (mauvais)	2.80	Performances médiocres malgré un temps élevé. Possible inadéquation aux données.
MH	0.176	1.648	**8.07** (très lent)	Légèrement meilleur que K-means en Silhouette, mais très lent et Davies-Bouldin élevé.

4. Interprétation et Recommandations

- **K-means** :
  - **Avantages** : Rapide et résultats acceptables (meilleur Davies-Bouldin).
  - **Améliorations possibles** :
    - Tester d'autres valeurs de `K`.
    - Utiliser une normalisation différente (e.g., `MinMaxScaler`).
- **DBSCAN** :
  - **Problème** : Score de silhouette négatif indique des erreurs d'assignation.
  - **Solution** :
    - Ajuster `eps` (réduire pour éviter trop de clusters) et `min_samples` (augmenter pour éliminer le bruit).
    - Exemple : `DBSCAN(eps=0.3, min_samples=15)`.
- **GMM** :
  - **Échec relatif** : Très faible Silhouette Score et Davies-Bouldin élevé.
  - **Cause possible** : Les données ne suivent pas une distribution gaussienne.
  - **Alternative** : Essayer `BayesianGaussianMixture` pour plus de flexibilité.
- **Clustering Hiérarchique (MH) :**
  - **Inconvénient majeur** : Temps d'exécution prohibitif pour peu de gain.
  - **Cas d'usage** : À réserver pour de petits datasets ou quand l'interprétabilité des hiérarchies est cruciale.

5. Conclusion

Méthode à privilégier : K-means (malgré des scores moyens, c'est le plus stable et rapide).