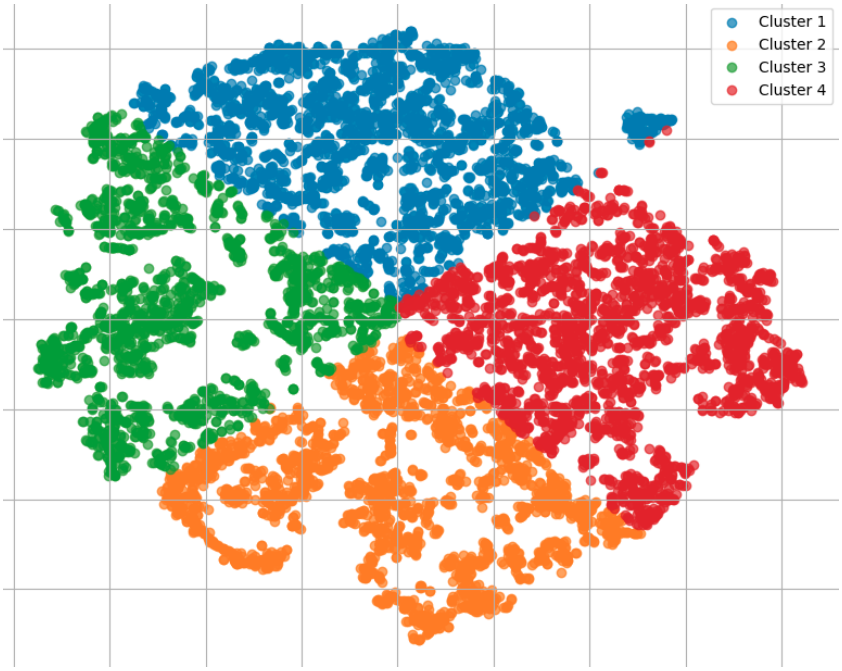
Université Abdelmalek Essaadi
Faculté des Sciences et Techniques de Tanger
Département Génie Informatique

Atelier 3 : «Clustering»



Encadré par :
Prof. ELAACHAK LOTFI

Réalisé par :
BARBYCH Mohamed

Cycle d'ingenieur Logiciels et Systèmes Intelligents
S3 - 2024/2025

Session d'octobre 2024

Table des matières

1 Introduction

Le présent atelier a pour objectif principal de pratiquer et approfondir les concepts fondamentaux du clustering en utilisant un dataset dédié aux transactions de cartes de crédit. Plus précisément, cet atelier est structuré en deux parties principales :

— **Partie 1 : Visualisation des données** - Cette étape consiste à explorer le dataset, analyser ses propriétés statistiques, visualiser les relations entre les variables, et appliquer des techniques de réduction de dimensions telles que PCA (Principal Component Analysis) et t-SNE (t-Distributed Stochastic Neighbor Embedding).

— **Partie 2 : Clustering** - L'objectif de cette section est de construire des modèles de clustering en utilisant plusieurs algorithmes (KMeans, Fuzzy CMeans, DBSCAN, SOM, et clustering hiérarchique), de visualiser les clusters obtenus, et de comparer les performances de ces méthodes.

Pour atteindre ces objectifs, nous avons utilisé les outils suivants :

— **Python** : Langage de programmation principal utilisé pour l'analyse et la modélisation.

— **Pandas** : Bibliothèque essentielle pour la manipulation et l'analyse des données tabulaires.

— **Scikit-learn** : Outil central pour la mise en œuvre des algorithmes de clustering et des techniques de réduction de dimensions.

— **Matplotlib et Seaborn** : Bibliothèques pour la visualisation des données sous forme de graphiques.

— **Skfuzzy et Minisom** : Bibliothèques dédiées respectivement aux algorithmes Fuzzy CMeans et SOM (Self-Organizing Map).

1.1 Dataset

Le dataset utilisé dans cet atelier est le **Credit Card Data Set**, disponible sur Kaggle (lien vers le dataset). Ce dataset contient diverses informations sur les transactions par carte de crédit, avec des caractéristiques telles que le solde moyen, les avances de trésorerie, la limite de crédit, et la fréquence des achats. Ces données sont idéales pour démontrer les techniques de clustering.

1.2 Structure du rapport

Ce rapport est divisé en sections suivant les étapes méthodologiques de l'atelier :

1. Exploration et visualisation des données (Partie 1).
2. Construction et analyse des modèles de clustering (Partie 2).
3. Interprétation des résultats et comparaison des algorithmes.

En conclusion, cet atelier nous permettra non seulement de comprendre les principes théoriques du clustering, mais également de les appliquer à un problème pratique, tout en comparant différentes approches pour en tirer des enseignements utiles.

2 Partie 1 : Visualisation des données

Cette section est dédiée à l'exploration et à la visualisation du dataset **CC_GENERAL.csv**. Nous y effectuons plusieurs étapes clés, notamment l'exploration des données, l'analyse statistique, la visualisation des relations entre les variables, et l'application de techniques de réduction de dimensions comme PCA et t-SNE.

2.1 Étape 1 : Exploration des données

L'objectif de cette étape est de charger le dataset, d'en examiner les premières lignes, et de fournir des informations générales sur la structure des données.

Code Python :

```
# Charger le dataset
import pandas as pd

# Charger les donn es
data = pd.read_csv("CC_GENERAL.csv")

# Aper u des premi res lignes
data.head()

# Informations g n rales sur le dataset
data.info()

# V rification des valeurs manquantes
missing_values = data.isnull().sum()
print(missing_values)
```

Explications :

— La fonction `head()` permet d'avoir un aperçu des cinq premières lignes du dataset.

— La méthode `info()` fournit des informations comme le type de données de chaque colonne et le nombre de valeurs non nulles.

— La méthode `isnull().sum()` compte les valeurs manquantes pour chaque colonne.

Visualisation :

[127]:
```
# Load the dataset
data = pd.read_csv("CC_GENERAL.csv")
```

[128]:
```
# PART 1: DATA VISUALIZATION
# 1. Exploring the dataset
print("Aperçu des 5 premières lignes du dataset :")
print(data.head())
```

```
Aperçu des 5 premières lignes du dataset :
  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
3                0.083333                 1              1        7500.0
4                0.000000                 0              1        1200.0

      PAYMENTS  MINIMUM_PAYMENTS  PRC_FULL_PAYMENT  TENURE
0   201.802084        139.509787          0.000000      12
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
```

FIGURE 1 – Exploration du dataset : aperçu et informations générales

[129]:
```
print("\nInformations générales sur le dataset :")
print(data.info())
```

```
Informations générales sur le dataset :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   CUST_ID                           8950 non-null   object
 1   BALANCE                           8950 non-null   float64
 2   BALANCE_FREQUENCY                 8950 non-null   float64
 3   PURCHASES                         8950 non-null   float64
 4   ONEOFF_PURCHASES                  8950 non-null   float64
 5   INSTALLMENTS_PURCHASES            8950 non-null   float64
 6   CASH_ADVANCE                      8950 non-null   float64
 7   PURCHASES_FREQUENCY               8950 non-null   float64
 8   ONEOFF_PURCHASES_FREQUENCY        8950 non-null   float64
 9   PURCHASES_INSTALLMENTS_FREQUENCY  8950 non-null   float64
 10  CASH_ADVANCE_FREQUENCY            8950 non-null   float64
 11  CASH_ADVANCE_TRX                  8950 non-null   int64
 12  PURCHASES_TRX                     8950 non-null   int64
 13  CREDIT_LIMIT                      8949 non-null   float64
 14  PAYMENTS                          8950 non-null   float64
 15  MINIMUM_PAYMENTS                  8637 non-null   float64
 16  PRC_FULL_PAYMENT                  8950 non-null   float64
 17  TENURE                            8950 non-null   int64
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
None
```

FIGURE 2 – Informations Générales

```
[130]: print("\nStatistiques des valeurs manquantes dans chaque colonne :")
       missing_values = data.isnull().sum()
       print(missing_values)
```

```
Statistiques des valeurs manquantes dans chaque colonne :
CUST_ID                             0
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
```

FIGURE 3 – Valeurs Manquantes

2.2 Étape 2 : Résumé statistique avec interprétation

Après avoir identifié les valeurs manquantes, celles-ci sont remplacées par la moyenne pour chaque colonne numérique. Ensuite, un résumé statistique est généré pour analyser la distribution des données.

Code Python :

```python
# Remplacement des valeurs manquantes par la moyenne
numeric_columns = data.select_dtypes(include=["float64", "int64"]).columns
data[numeric_columns] = data[numeric_columns].apply(lambda col: col.fillna(
    col.mean()), axis=0)

# R sum  statistique
data.describe()
```

Explications :

— `select_dtypes()` permet de sélectionner uniquement les colonnes numériques.

— `fillna(col.mean())` remplace les valeurs manquantes par la moyenne des colonnes correspondantes.

— `describe()` génère des statistiques descriptives comme la moyenne, l'écart-type, les minimums et maximums.

Visualisations :

```
Résumé statistique :
            BALANCE  BALANCE_FREQUENCY     PURCHASES  ONEOFF_PURCHASES  \
count   8950.000000        8950.000000   8950.000000       8950.000000
mean    1564.474828           0.877271   1003.204834        592.437371
std     2081.531879           0.236904   2136.634782       1659.887917
min        0.000000           0.000000      0.000000          0.000000
25%      128.281915           0.888889     39.635000          0.000000
50%      873.385231           1.000000    361.280000         38.000000
75%     2054.140036           1.000000   1110.130000        577.405000
max    19043.138560           1.000000  49039.570000      40761.250000

       INSTALLMENTS_PURCHASES  CASH_ADVANCE  PURCHASES_FREQUENCY  \
count             8950.000000   8950.000000          8950.000000
mean               411.067645    978.871112             0.490351
std                904.338115   2097.163877             0.401371
min                  0.000000      0.000000             0.000000
25%                  0.000000      0.000000             0.083333
50%                 89.000000      0.000000             0.500000
75%                468.637500   1113.821139             0.916667
max              22500.000000  47137.211760             1.000000

       ONEOFF_PURCHASES_FREQUENCY  PURCHASES_INSTALLMENTS_FREQUENCY  \
count                 8950.000000                       8950.000000
mean                     0.202458                          0.364437
std                      0.298336                          0.397448
min                      0.000000                          0.000000
25%                      0.000000                          0.000000
50%                      0.083333                          0.166667
75%                      0.300000                          0.750000
max                      1.000000                          1.000000

       CASH_ADVANCE_FREQUENCY  CASH_ADVANCE_TRX  PURCHASES_TRX  CREDIT_LIMIT  \
count             8950.000000       8950.000000    8950.000000   8950.000000
mean                 0.135144          3.248827      14.709832   4494.449450
std                  0.200121          6.824647      24.857649   3638.612411
min                  0.000000          0.000000       0.000000     50.000000
25%                  0.000000          0.000000       1.000000   1600.000000
50%                  0.000000          0.000000       7.000000   3000.000000
75%                  0.222222          4.000000      17.000000   6500.000000
max                  1.500000        123.000000     358.000000  30000.000000

           PAYMENTS  MINIMUM_PAYMENTS  PRC_FULL_PAYMENT       TENURE
count   8950.000000       8950.000000       8950.000000  8950.000000
mean    1733.143852        864.206542          0.153715    11.517318
std     2895.063757       2330.588021          0.292499     1.338331
min        0.000000          0.019163          0.000000     6.000000
25%      383.276166        170.857654          0.000000    12.000000
50%      856.901546        335.628312          0.000000    12.000000
75%     1901.134317        864.206542          0.142857    12.000000
max    50721.483360      76406.207520          1.000000    12.000000
```

FIGURE 4 – Résumé statistique des colonnes numériques

Code Python :

```python
# Visualisation des statistiques : Moyenne et m diane
import matplotlib.pyplot as plt

mean_values = summary.loc["mean"]
median_values = summary.loc["50%"]  # M diane (50e percentile)

plt.figure(figsize=(12, 6))
plt.bar(mean_values.index, mean_values, alpha=0.7, label="Moyenne", color="
    skyblue")
plt.bar(median_values.index, median_values, alpha=0.5, label="M diane",
    color="orange")
plt.xticks(rotation=90)
plt.title("Comparaison des moyennes et des m dianes des colonnes
    num riques")
plt.legend()
plt.tight_layout()
plt.show()
```

Visualisation des statistiques : Moyenne et médiane

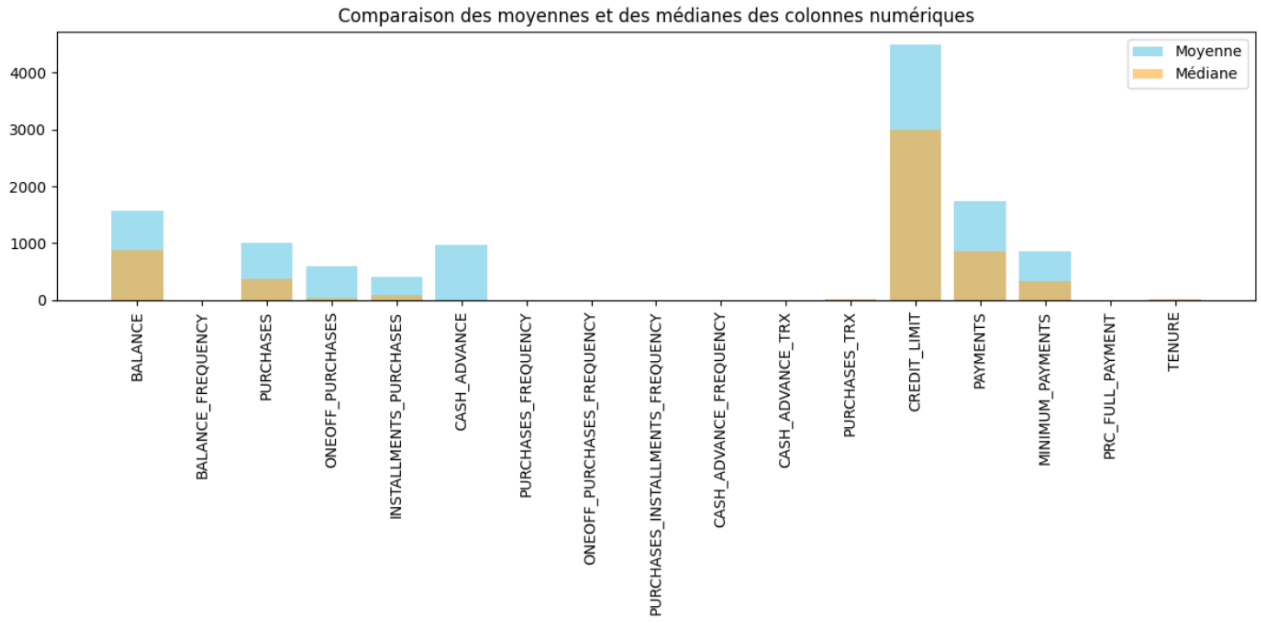


FIGURE 5 – Visualisation des statistiques : Moyenne et médiane

2.3 Étape 3 : Nuages de points (Scatter Matrix)

Pour visualiser les relations entre certaines colonnes importantes, une matrice de dispersion (*scatter matrix*) est utilisée.

Code Python :

```
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt

# S lection de certaines colonnes importantes
selected_features = ["BALANCE", "PURCHASES", "CASH_ADVANCE", "CREDIT_LIMIT"]

# Visualisation des relations
scatter_matrix(data[selected_features], figsize=(12, 12), alpha=0.7,
    diagonal="kde", color="blue")
plt.suptitle("Nuages des points pour les Features s lectionn es", fontsize
    =16)
plt.show()
```

Explications :

— Les nuages de points montrent les corrélations potentielles entre les variables sélectionnées.

— La diagonale utilise une estimation par noyau (*kde*) pour montrer la distribution des variables individuelles.

Visualisation (Page suivante) :

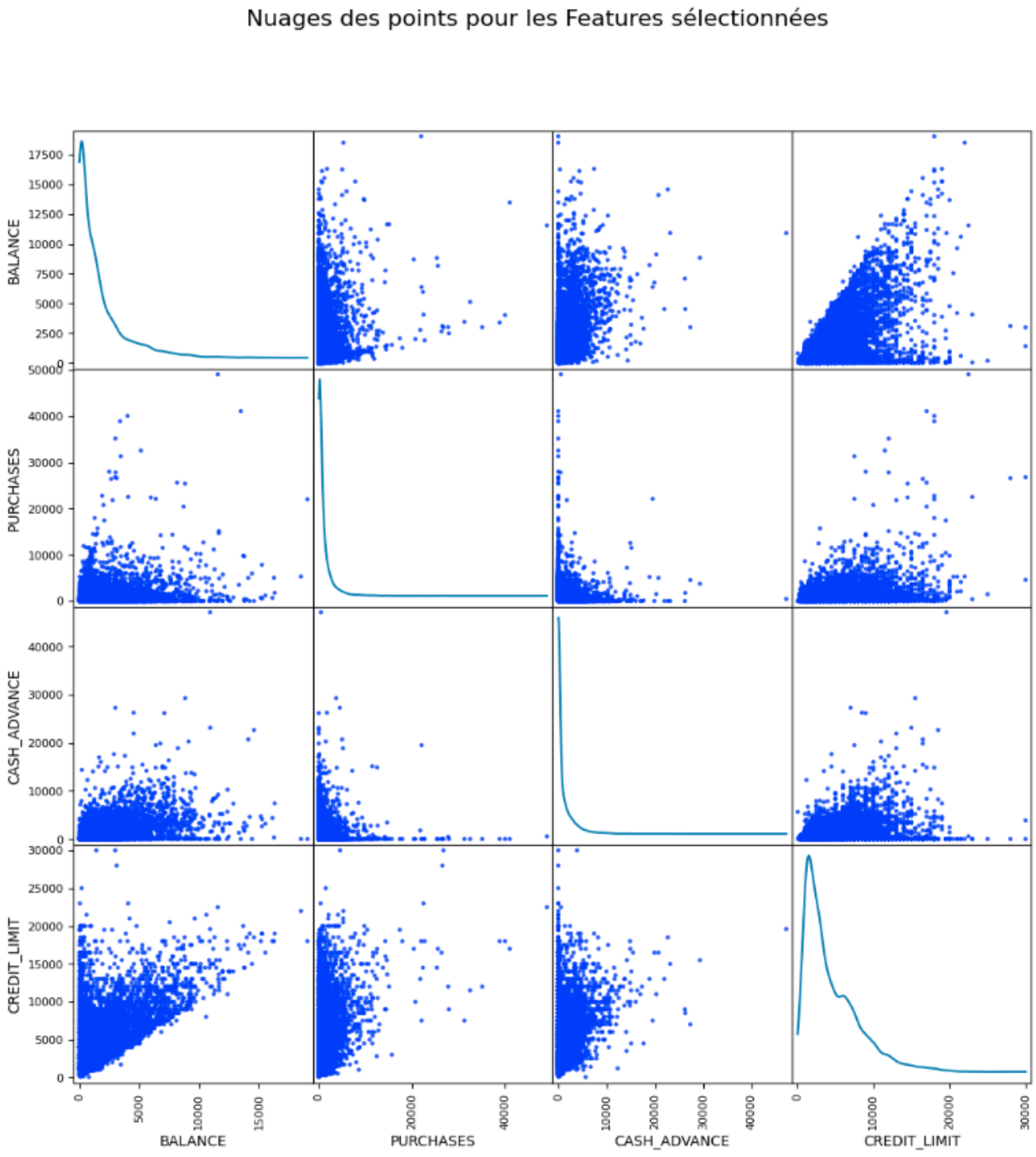


FIGURE 6 – Nuages des points pour les colonnes sélectionnées

2.4 Étape 4 : Application de PCA et t-SNE

Les techniques de réduction de dimensions, **PCA** et **t-SNE**, sont appliquées pour visualiser les données en deux dimensions.

2.4.1 PCA (Principal Component Analysis)

Code Python :

```
from sklearn.decomposition import PCA

# R duction des dimensions avec PCA
pca = PCA(n_components=2)
pca_features = pca.fit_transform(data_scaled)

# Visualisation des r sultats PCA
plt.figure(figsize=(10, 8))
plt.scatter(pca_features[:, 0], pca_features[:, 1], s=10, c="blue", alpha
    =0.6)
plt.title("Projection des donn es avec PCA")
plt.xlabel("Composante principale 1")
plt.ylabel("Composante principale 2")
plt.grid(True)
plt.show()
```

Visualisation (Page suivante) :

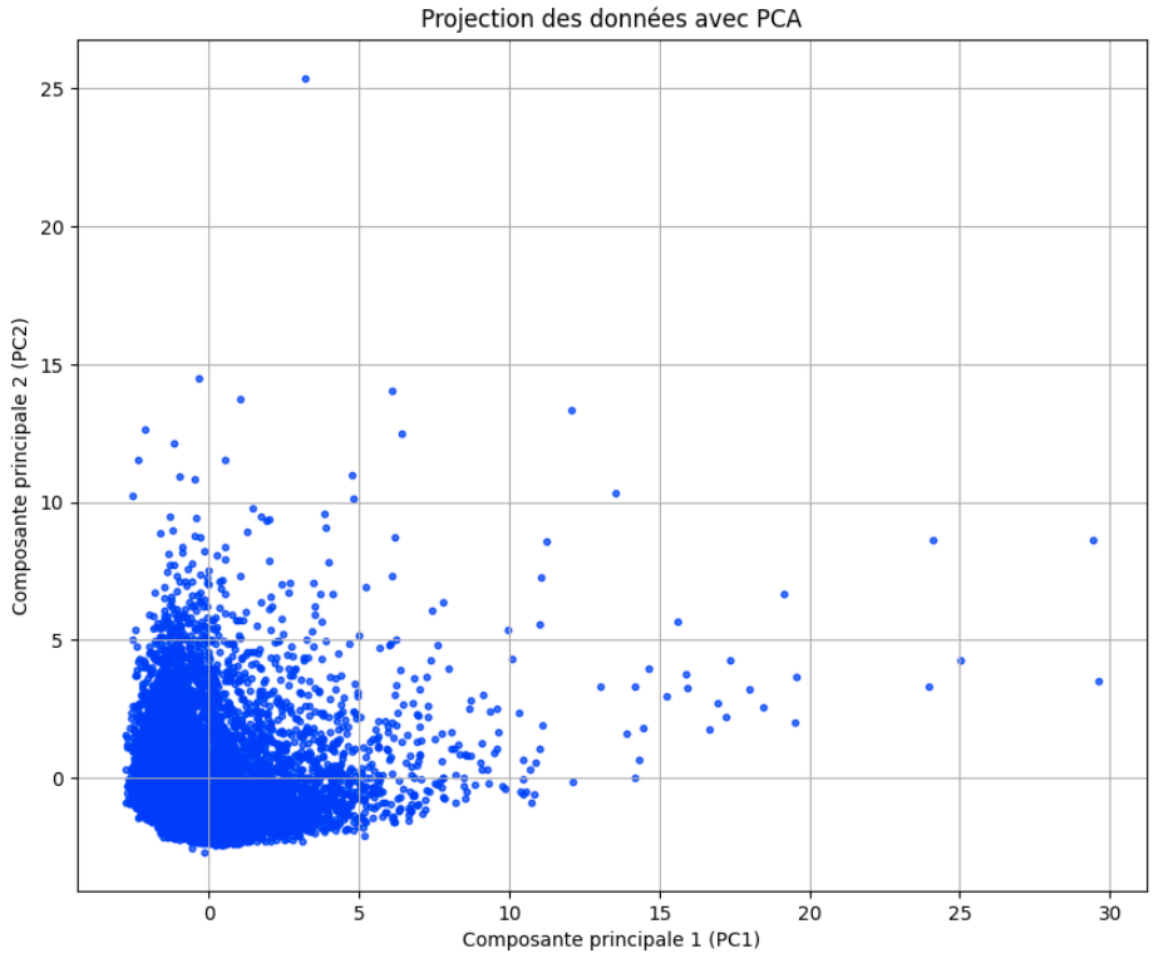


FIGURE 7 – Projection des données avec PCA

2.4.2 t-SNE (t-Distributed Stochastic Neighbor Embedding)

Code Python :

```
from sklearn.manifold import TSNE

# R duction des dimensions avec t-SNE
tsne = TSNE(n_components=2, random_state=42, perplexity=30, verbose=1)
tsne_features = tsne.fit_transform(data_scaled)

# Visualisation des r sultats t-SNE
plt.figure(figsize=(10, 8))
plt.scatter(tsne_features[:, 0], tsne_features[:, 1], s=10, c="green", alpha
    =0.6)
plt.title("Projection des donn es avec t-SNE")
plt.xlabel("Composante t-SNE 1")
plt.ylabel("Composante t-SNE 2")
plt.grid(True)
```

```
14 plt.show()
```

Visualisation :

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 8950 samples in 0.002s...
[t-SNE] Computed neighbors for 8950 samples in 0.497s...
[t-SNE] Computed conditional probabilities for sample 1000 / 8950
[t-SNE] Computed conditional probabilities for sample 2000 / 8950
[t-SNE] Computed conditional probabilities for sample 3000 / 8950
[t-SNE] Computed conditional probabilities for sample 4000 / 8950
[t-SNE] Computed conditional probabilities for sample 5000 / 8950
[t-SNE] Computed conditional probabilities for sample 6000 / 8950
[t-SNE] Computed conditional probabilities for sample 7000 / 8950
[t-SNE] Computed conditional probabilities for sample 8000 / 8950
[t-SNE] Computed conditional probabilities for sample 8950 / 8950
[t-SNE] Mean sigma: 0.427264
[t-SNE] KL divergence after 250 iterations with early exaggeration: 83.965378
[t-SNE] KL divergence after 1000 iterations: 1.501112
```

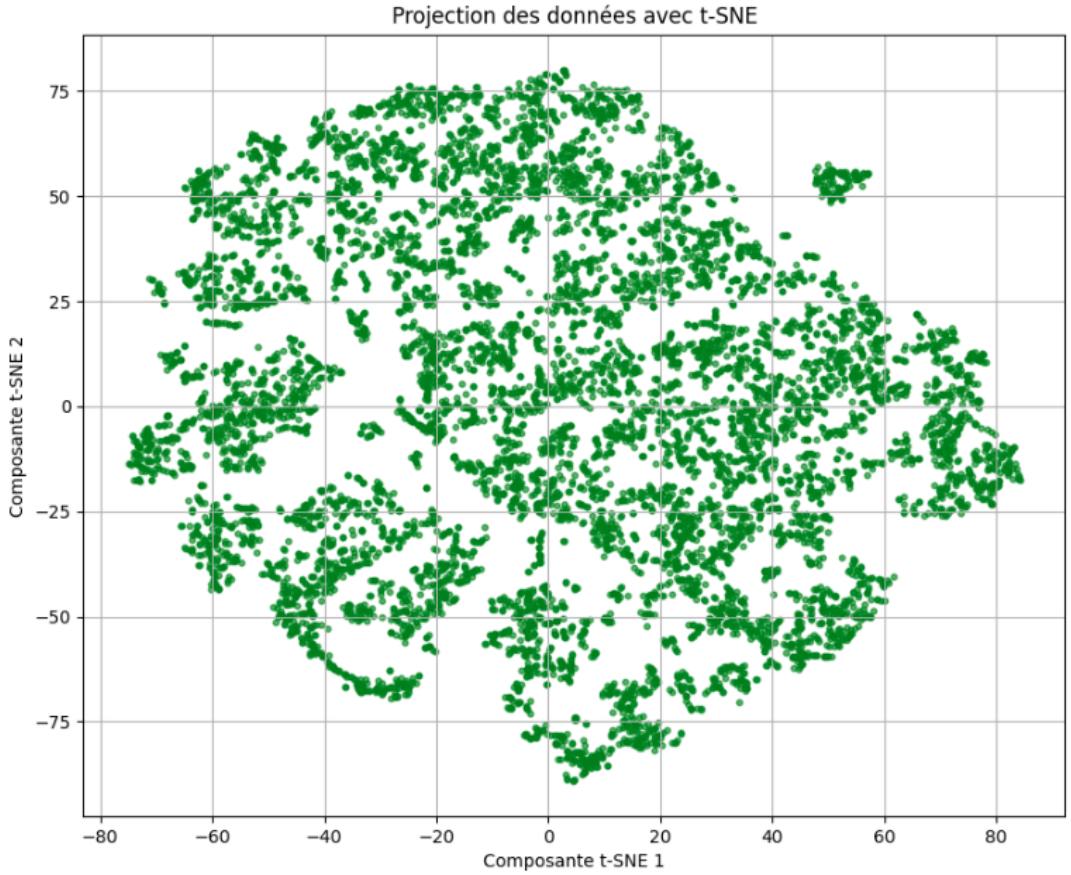


FIGURE 8 – Projection des données avec t-SNE

Interprétations :

— PCA montre une séparation des données basée sur des composantes principales, mais sans clusters bien définis.

— t-SNE, en revanche, montre une organisation locale plus claire des données, adaptée à l'identification de clusters potentiels.

3 Partie 2 : Clustering

3.1 Introduction

Dans cette partie, nous appliquons différentes techniques de clustering sur les nouvelles caractéristiques obtenues via PCA et t-SNE. Ces techniques permettent de regrouper les données en clusters distincts basés sur des similitudes. Nous abordons KMeans, Fuzzy CMeans, DBSCAN, clustering hiérarchique, et SOM (Self-Organizing Map). Chaque méthode est interprétée en détail avec des visualisations et des analyses.

3.2 KMeans Clustering

3.2.1 Méthode d'Elbow pour déterminer le nombre optimal de clusters

La méthode d'Elbow permet d'identifier le nombre optimal de clusters en examinant la variation intra-clusters (SSE). Un graphique est tracé pour PCA et t-SNE.

Code Python :

```
# M thode d'Elbow pour PCA et t-SNE

from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# M thode de l'Elbow pour PCA
sse_pca = []
k_values = range(1, 11)
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(df_pca)
    sse_pca.append(kmeans.inertia_)

plt.figure(figsize=(8, 6))
plt.plot(k_values, sse_pca, marker='o')
plt.title("M thode de l'Elbow pour PCA")
plt.xlabel("Nombre de clusters (k)")
plt.ylabel("SSE (Inertie intra-clusters)")
plt.grid(True)
plt.show()

# M thode de l'Elbow pour t-SNE
sse_tsne = []
```

```
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(df_tsne)
    sse_tsne.append(kmeans.inertia_)

plt.figure(figsize=(8, 6))
plt.plot(k_values, sse_tsne, marker='o')
plt.title("M thode de l'Elbow pour t-SNE")
plt.xlabel("Nombre de clusters (k)")
plt.ylabel("SSE (Inertie intra-clusters)")
plt.grid(True)
plt.show()
```

Résultat :

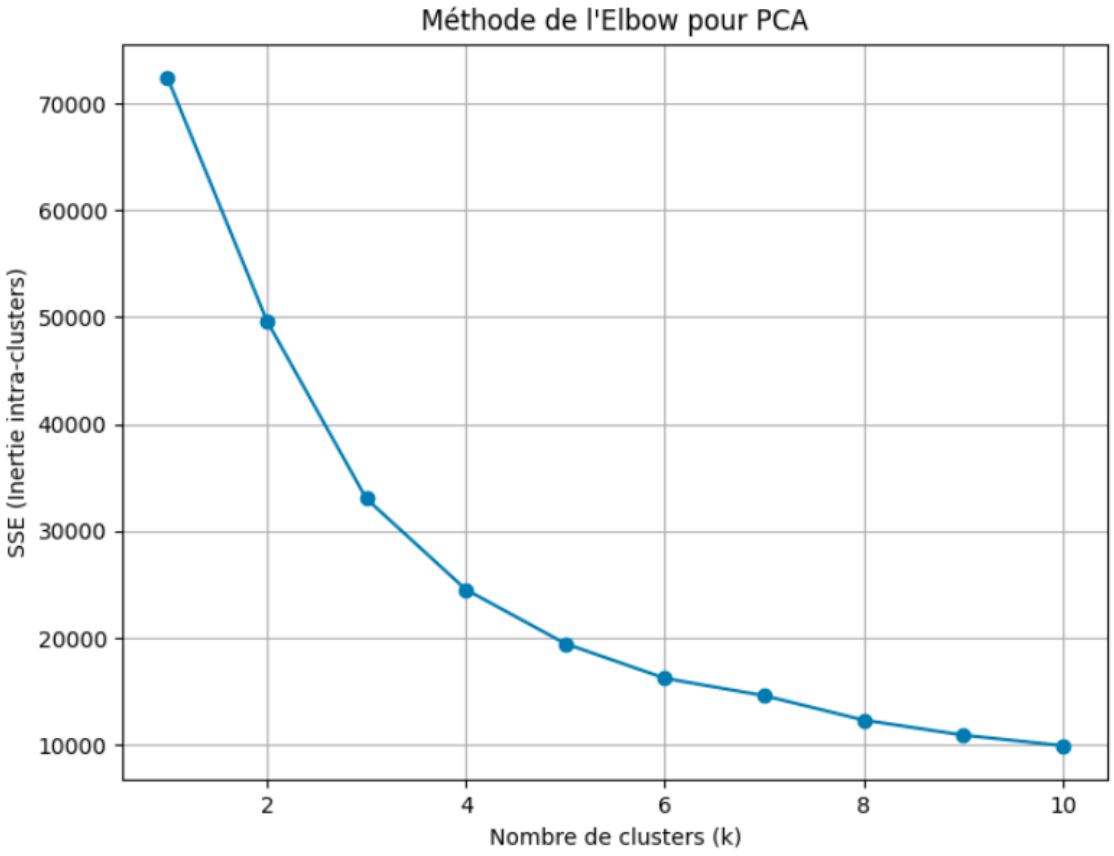


FIGURE 9 – Méthode d'Elbow pour PCA

Résultat :

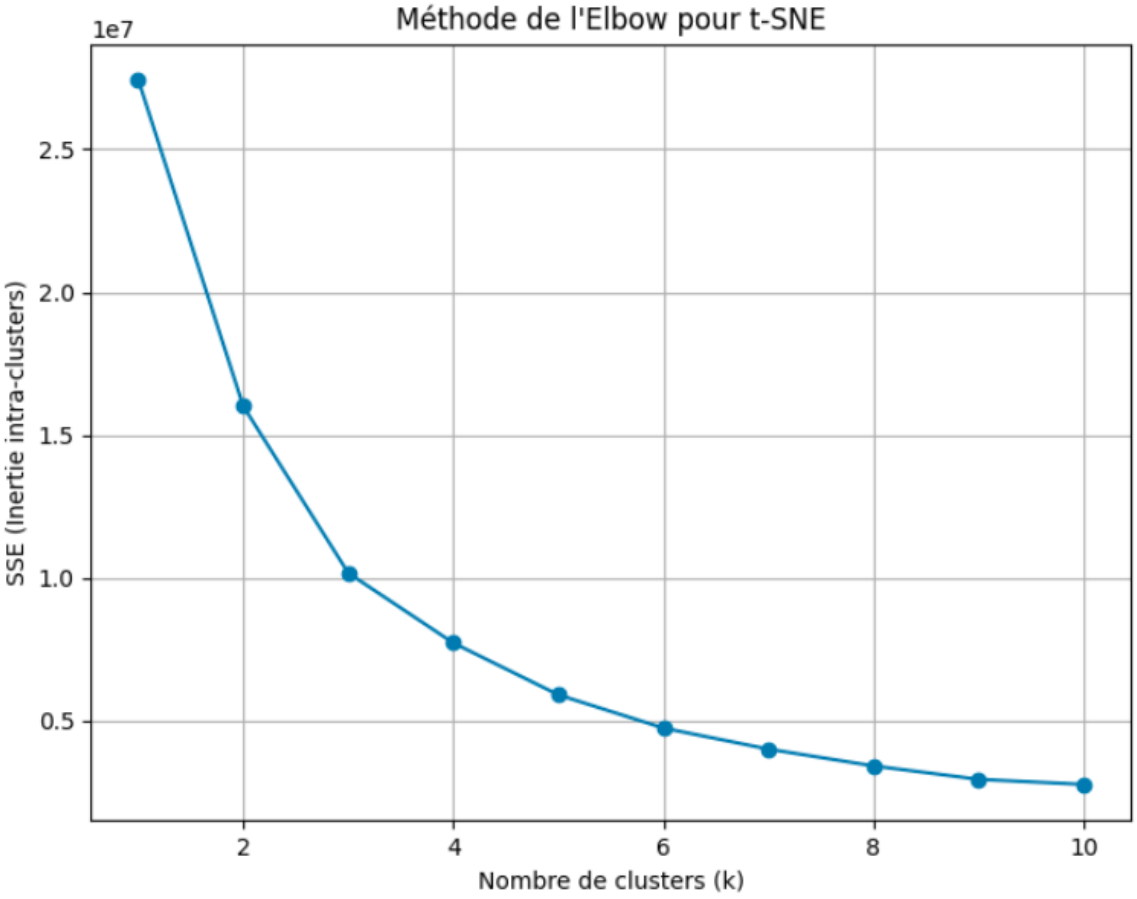


FIGURE 10 – Méthode d'Elbow pour TSNE

Interprétation : L'endroit où la courbe "coude" est observé indique le nombre optimal de clusters. Ici, le coude apparaît autour de k=4.

3.2.2 Application de KMeans sur PCA et t-SNE

Nous appliquons KMeans sur les données projetées en 2D par PCA et t-SNE avec k=4.

Code Python :

```
# Clustering et Visualisation des r sultats KMeans

# KMeans sur PCA
kmeans_pca = KMeans(n_clusters=4, random_state=42)  # Adaptez le nombre
    optimal de clusters
labels_pca = kmeans_pca.fit_predict(df_pca)

plt.figure(figsize=(10, 8))
for cluster in range(4):
```

```
    cluster_data = df_pca[labels_pca == cluster]
    plt.scatter(
        cluster_data["PC1"], cluster_data["PC2"],
        label=f"Cluster {cluster + 1}", alpha=0.7
    )
plt.title("Clustering KMeans sur PCA")
plt.xlabel("Composante principale 1 (PC1)")
plt.ylabel("Composante principale 2 (PC2)")
plt.legend()
plt.grid(True)
plt.show()
```

Résultat :

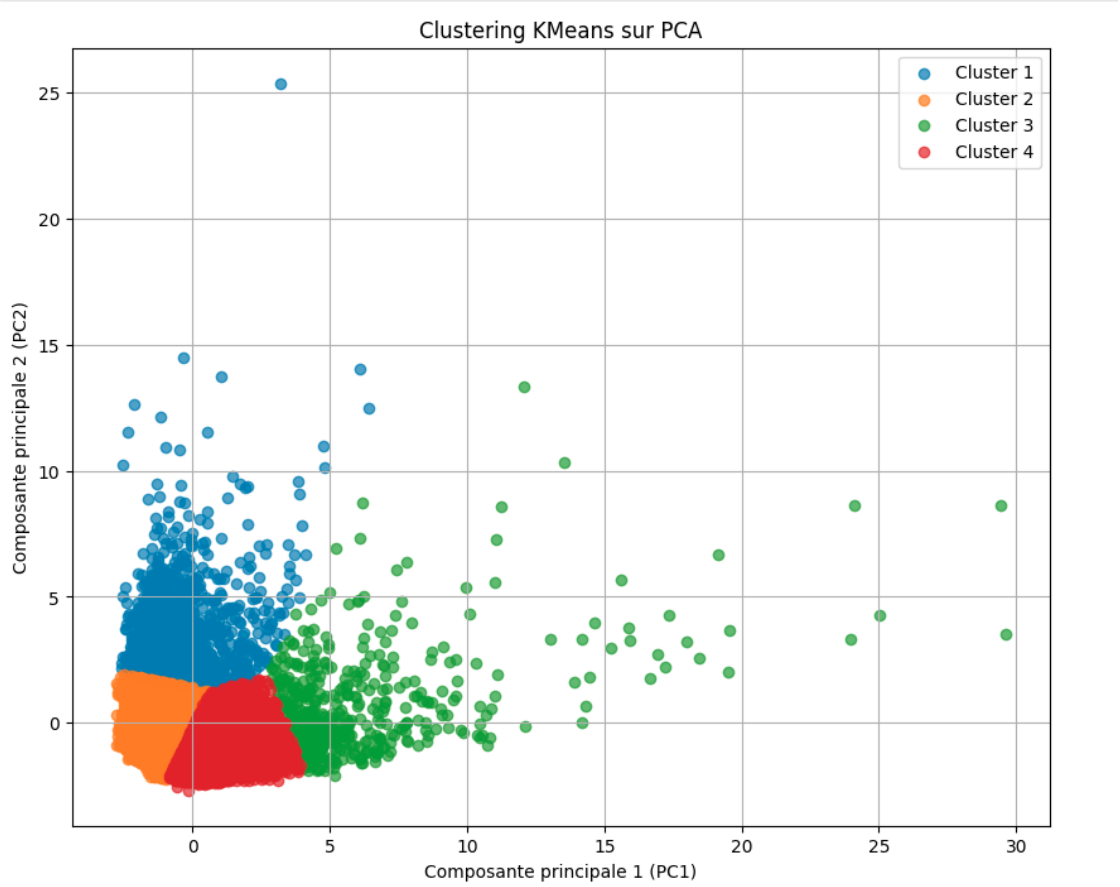


FIGURE 11 – Clustering KMeans sur PCA

Interprétation : Les clusters sont bien séparés en utilisant PCA, reflétant des groupes distincts dans les données.

Code Python :

```
# KMeans sur t-SNE
kmeans_tsne = KMeans(n_clusters=4, random_state=42)
```

```
labels_tsne = kmeans_tsne.fit_predict(df_tsne)

plt.figure(figsize=(10, 8))
for cluster in range(4):
    cluster_data = df_tsne[labels_tsne == cluster]
    plt.scatter(
        cluster_data["t-SNE1"], cluster_data["t-SNE2"],
        label=f"Cluster {cluster + 1}", alpha=0.7
    )
plt.title("Clustering KMeans sur t-SNE")
plt.xlabel("Composante t-SNE 1")
plt.ylabel("Composante t-SNE 2")
plt.legend()
plt.grid(True)
plt.show()
```

Résultat (Page suivante) :

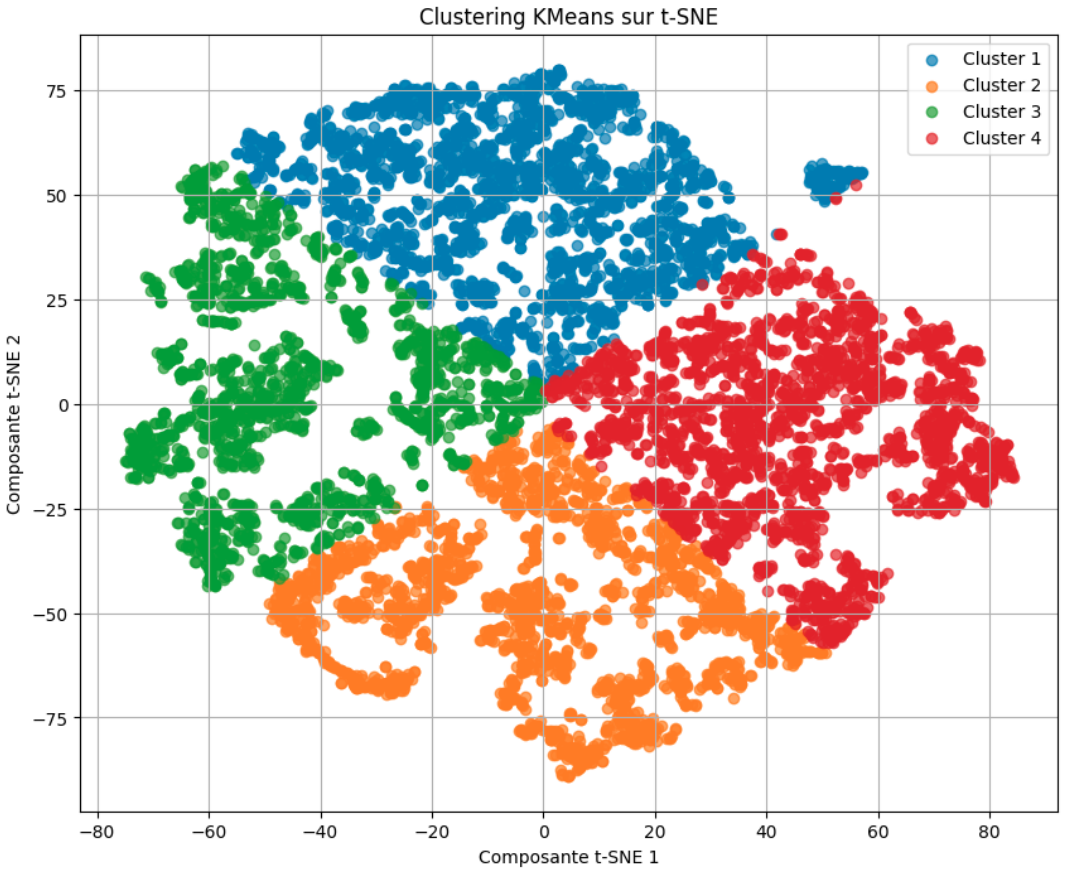


FIGURE 12 – Clustering KMeans sur TSNE

3.3 Fuzzy CMeans Clustering

Nous utilisons l'algorithme Fuzzy CMeans pour une approche de clustering flou où chaque point peut appartenir à plusieurs clusters avec des degrés d'appartenance.

Code Python :

```
# Fuzzy CMeans sur PCA
import skfuzzy as fuzz
import numpy as np

cntr, u, _, _, _, _, _ = fuzz.cluster.cmeans(df_pca.T, c=4, m=2, error
    =0.005, maxiter=1000)
fuzzy_labels = np.argmax(u, axis=0)

plt.figure(figsize=(10, 8))
for cluster in range(4):
    cluster_data = df_pca[fuzzy_labels == cluster]
    plt.scatter(
        cluster_data["PC1"], cluster_data["PC2"],
        label=f"Cluster {cluster + 1}", alpha=0.7
    )
plt.title("Clustering Fuzzy CMeans sur PCA")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend()
plt.grid(True)
plt.show()
```

Résultat :

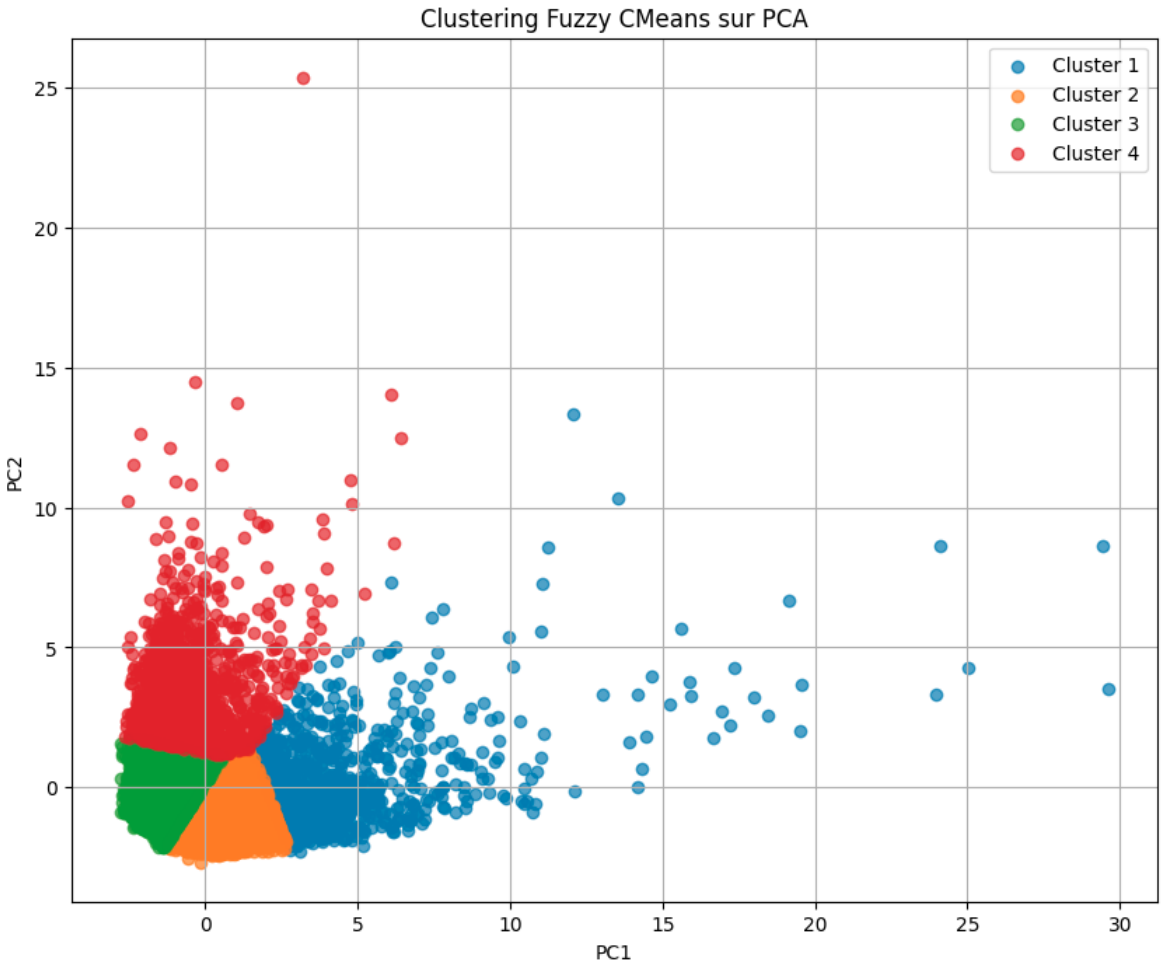


FIGURE 13 – Clustering Fuzzy CMeans sur PCA

Interprétation : Les clusters sont identifiés avec des frontières plus douces, ce qui permet une meilleure gestion des chevauchements dans les données.

3.4 DBSCAN (Density-Based Spatial Clustering)

DBSCAN identifie les clusters basés sur la densité, permettant de détecter des groupes et des anomalies.

Code Python :

```
# DBSCAN sur PCA
dbscan_pca = DBSCAN(eps=2, min_samples=5).fit(df_pca)
dbscan_labels_pca = dbscan_pca.labels_

plt.figure(figsize=(10, 8))
plt.scatter(df_pca["PC1"], df_pca["PC2"], c=dbscan_labels_pca, cmap="viridis
    ", s=50, alpha=0.6)
plt.title("Clustering DBSCAN sur PCA")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.grid(True)
plt.show()
```

Résultat :

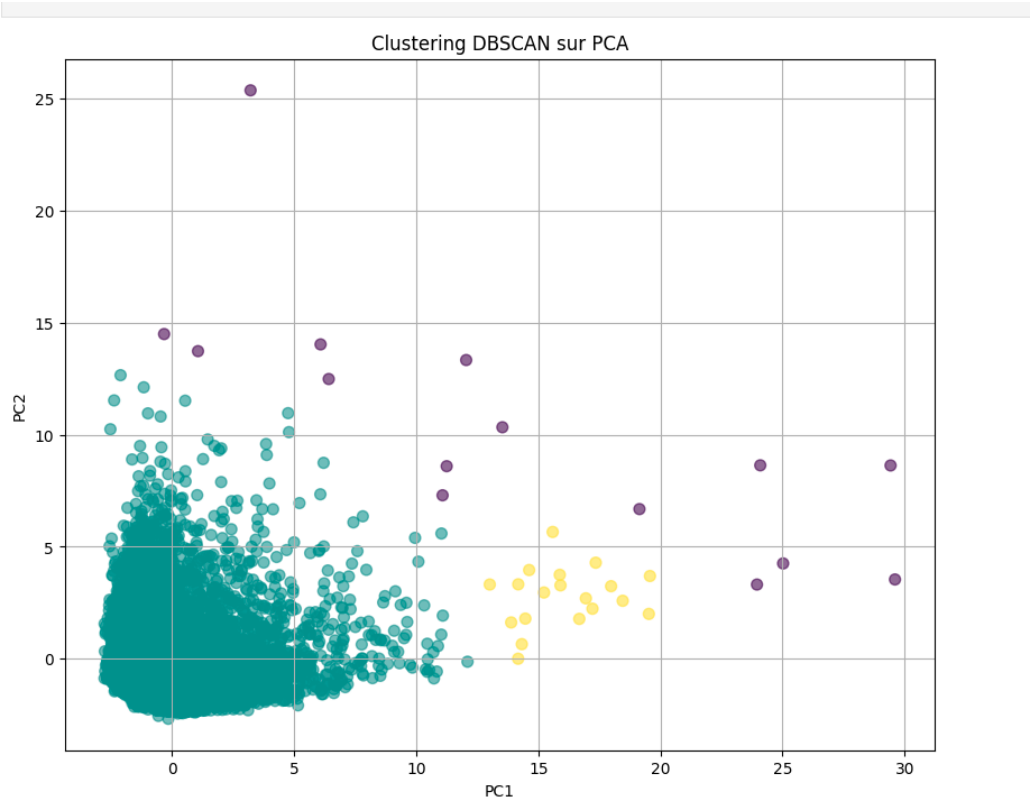


FIGURE 14 – Clustering DBSCAN sur PCA

Interprétation : DBSCAN identifie des clusters basés sur la densité et marque certains points comme des anomalies (bruit).

3.5 Clustering Hiérarchique

L'approche hiérarchique regroupe les données en formant un dendrogramme.

Code Python :

```
# Clustering Hi rarchique sur PCA
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
linked = linkage(df_pca, method='ward')
plt.figure(figsize=(10, 7))
dendrogram(linked)
plt.title("Dendrogramme Hi rarchique")
plt.show()
```

Résultat :

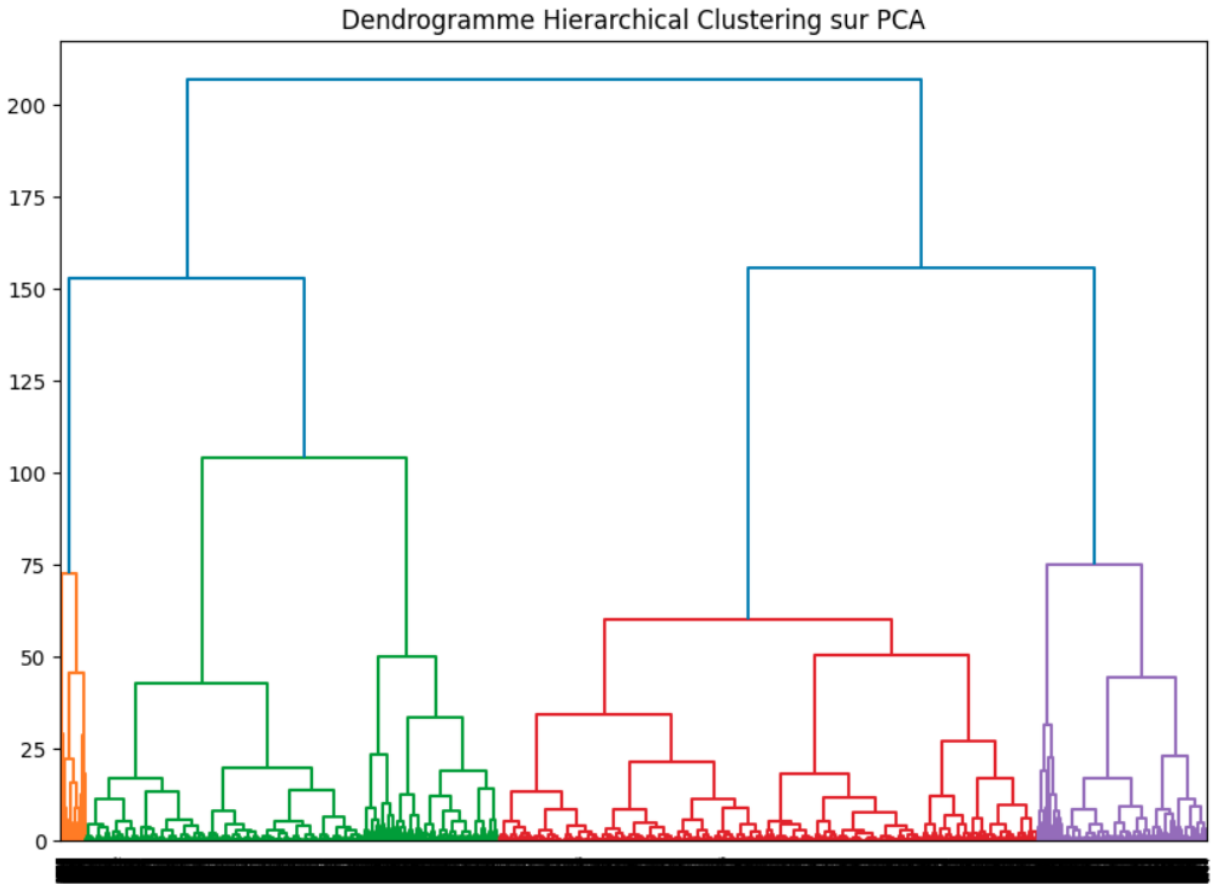


FIGURE 15 – Dendrogramme Hiérarchique

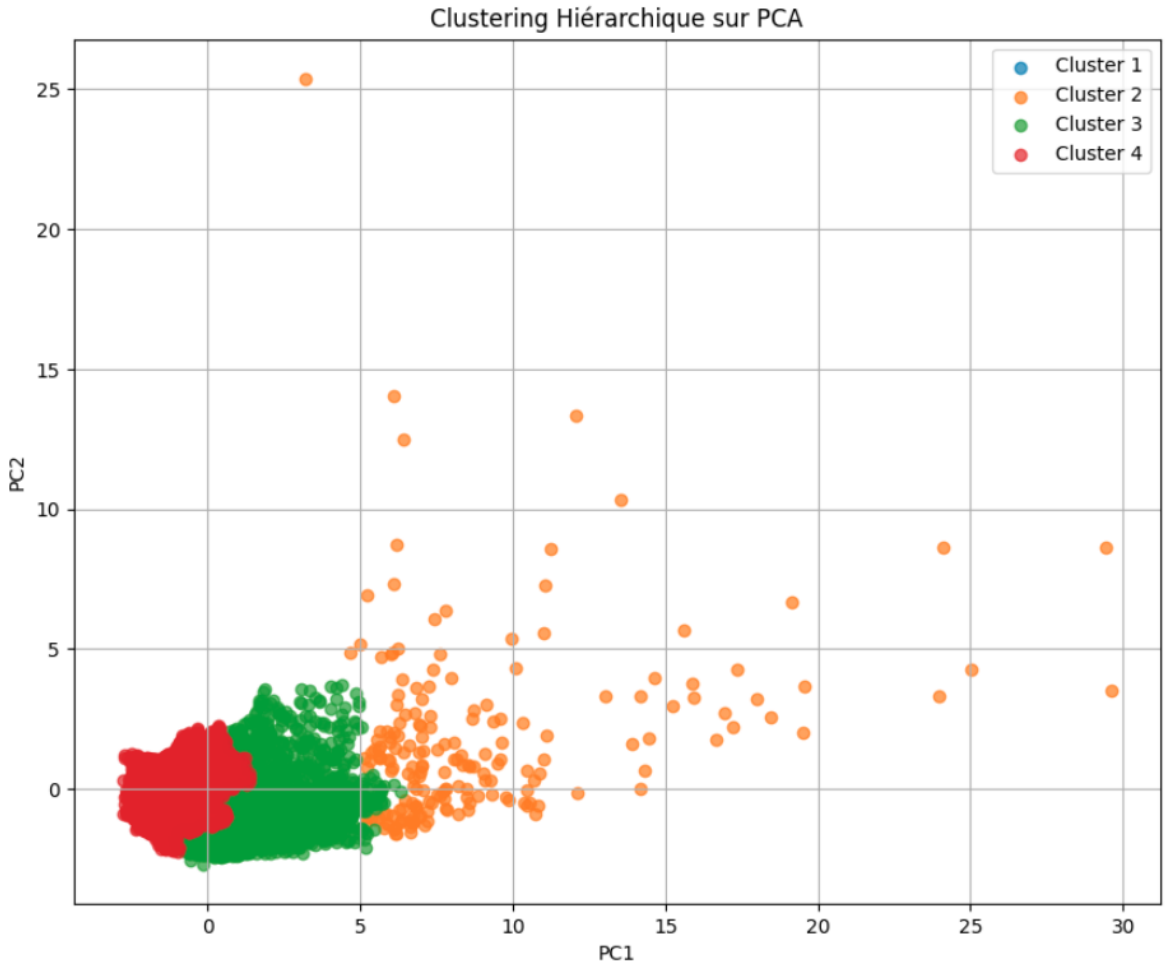


FIGURE 16 – Clustring Hiérarchique sur PCA

Interprétation : Le dendrogramme montre la structure des clusters et aide à définir un seuil pour le nombre optimal de groupes.

3.6 SOM (Self-Organizing Map)

Le SOM projette les données en une carte de caractéristiques auto-organisée.

Code Python :

```
# SOM sur PCA
from minisom import MiniSom
som = MiniSom(x=10, y=10, input_len=2, sigma=0.5, learning_rate=0.5)
som.random_weights_init(df_pca.to_numpy())
som.train_random(df_pca.to_numpy(), num_iteration=100)

plt.figure(figsize=(10, 8))
plt.pcolor(som.distance_map().T, cmap="Blues")
plt.title("Carte auto-organis e (SOM)")
```

```
plt.colorbar()
plt.show()
```

Résultat :

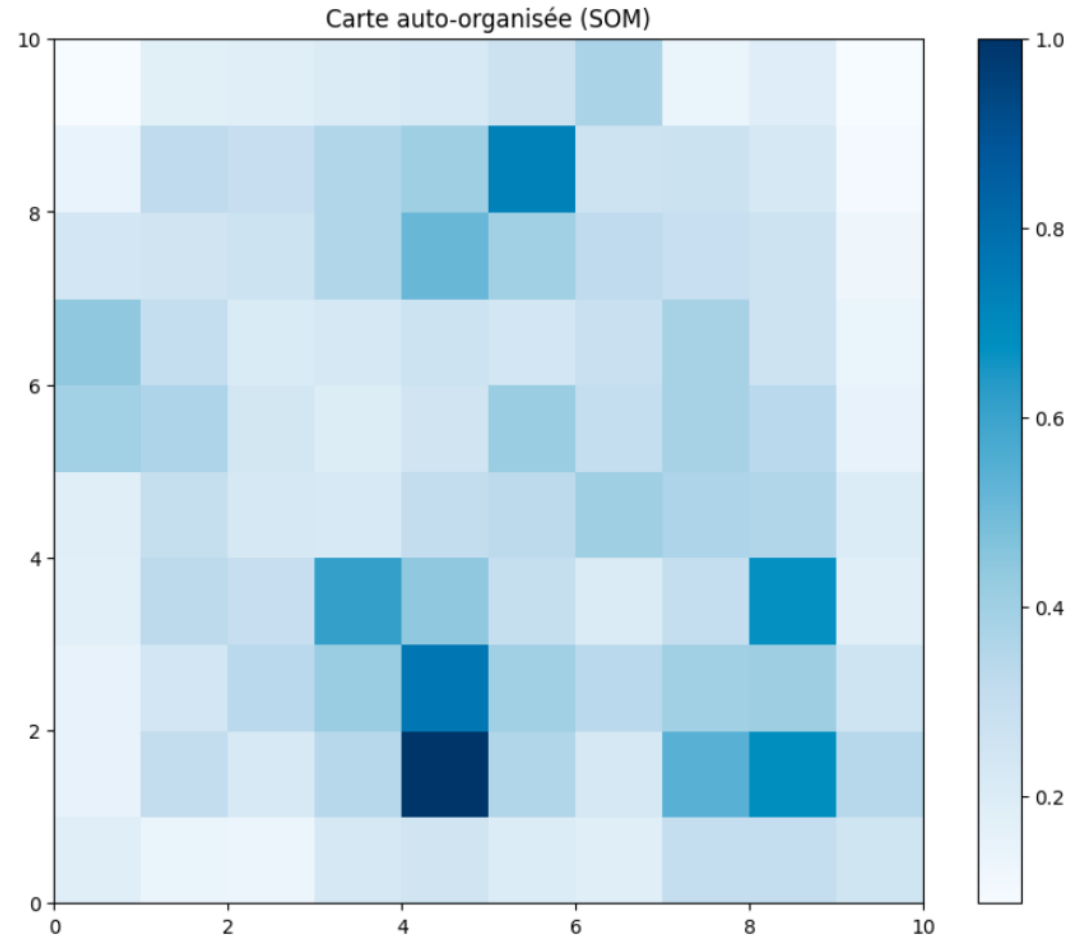


FIGURE 17 – Carte auto-organisée (SOM)

Interprétation : La carte SOM montre les distances entre les nœuds, aidant à visualiser les regroupements et anomalies.

3.7 Conclusion partielle

Chaque méthode de clustering offre des perspectives uniques :

- **KMeans** fonctionne bien pour les clusters bien définis, mais peut être sensible aux anomalies.
- **Fuzzy CMeans** offre une meilleure flexibilité pour les données floues.
- **DBSCAN** identifie les clusters denses et marque les points isolés comme bruit.
- **Clustering Hiérarchique** fournit une structure visuelle claire via le dendrogramme.
- **SOM** projette les données en une carte utile pour détecter les anomalies.

4 Conclusion

Cet atelier a permis une exploration approfondie des concepts fondamentaux de **clustering** et de réduction de dimensionnalité à travers des approches variées, en appliquant des outils modernes pour analyser un jeu de données complexe. Les étapes réalisées ont permis de répondre aux objectifs fixés en combinant une analyse qualitative et quantitative avec des interprétations détaillées.

Dans la **première partie**, nous avons procédé à une visualisation exploratoire des données en utilisant des techniques telles que la matrice de dispersion, le résumé statistique et le traitement des données manquantes. La réduction de dimensionnalité, réalisée grâce à **PCA** et **t-SNE**, a fourni des représentations compactes et visuellement informatives des données. Ces projections ont montré que les caractéristiques du dataset présentent une structure sous-jacente, rendant les données plus exploitables pour des techniques de clustering.

La **deuxième partie** s'est concentrée sur l'application de différentes méthodes de clustering :

- **KMeans** a démontré une capacité efficace à regrouper les données, particulièrement sur les projections PCA et t-SNE. La méthode d'Elbow a permis d'identifier un nombre optimal de clusters pour une partition cohérente.
- **Fuzzy CMeans**, avec son approche flexible et floue, a révélé une perspective plus nuancée des clusters, particulièrement utile dans des situations où les groupes se chevauchent.
- **DBSCAN**, basé sur la densité, a identifié les anomalies et les outliers tout en offrant une vue robuste des clusters denses.
- **Le clustering hiérarchique**, via le dendrogramme, a montré la hiérarchie et la structure naturelle des données, avec une capacité à regrouper dynamiquement en ajustant le seuil.
- Enfin, le **SOM (Self-Organizing Map)** a visualisé les regroupements et les distances entre les nœuds sous la forme d'une carte auto-organisée, apportant des informations complémentaires sur la distribution des données.

Comparaison des algorithmes :

- Les approches basées sur la partition, comme KMeans et Fuzzy CMeans, offrent des solutions rapides et exploitables pour des données bien segmentées.
- Les algorithmes basés sur la densité, comme DBSCAN, sont adaptés aux données non linéaires avec des anomalies.
- Les approches hiérarchiques et SOM apportent des perspectives globales et permettent

une interprétation visuelle enrichie.

Conclusion générale : L'analyse menée dans cet atelier démontre qu'aucune méthode unique ne peut répondre à tous les besoins. Le choix de l'algorithme dépend de la nature des données, des objectifs d'analyse et des contraintes spécifiques du problème. KMeans s'est avéré performant pour des données bien réparties, tandis que DBSCAN et SOM ont excellé dans l'identification des anomalies et des structures complexes. Les projections via PCA et t-SNE ont été cruciales pour optimiser les performances des algorithmes, en simplifiant les données tout en conservant leurs propriétés essentielles.

Cette étude met en évidence l'importance de combiner différentes techniques pour obtenir une vision complète des données. L'approche multidimensionnelle adoptée ici ouvre la voie à des applications pratiques dans divers domaines tels que l'analyse de clientèle, la segmentation de marché, ou encore la détection d'anomalies dans des systèmes complexes.