École Nationale Des Sciences Appliquées Agadir

Rapport d'Analyse Des Données

Documentation

Élève :
Salma Elaissari

Enseignant :
Prof.BrahimEl Asri

Filière:
FID

24 avril 2024

Résumé

La première partie du rapport expose l'importance de la classification dans la structuration des données et la prise de décision. Ensuite, une introduction aux méthodes de classification hiérarchique et k-means est présentée, mettant en évidence leurs principes de fonctionnement et leurs applications.
Les objectifs du projet sont définis, notamment la création d'un programme offrant des fonctionnalités de classification, une interface utilisateur intuitive, une documentation complète du code, des exemples d'utilisation et une analyse des résultats obtenus.
Le rapport décrit ensuite en détail le projet, mettant en lumière les fonctionnalités du programme, l'interface utilisateur conviviale, la documentation du code et les exemples d'utilisation illustrant les méthodes CAH et k-means.
Une analyse comparative des méthodes CAH et k-means est présentée, discutant de leurs avantages et inconvénients respectifs. Le fonctionnement du code Python pour implémenter ces méthodes est expliqué, mettant en évidence les packages, classes et fonctions utilisés.
Enfin, une description de l'interface graphique du programme, tant pour k-means que pour CAH, est fournie, accompagnée d'une explication détaillée de son utilisation.

Table des matières

1 Introduction

La classification est l'une des tâches fondamentales en analyse de données et en apprentissage automatique. Elle consiste à regrouper des éléments similaires dans des catégories ou des classes distinctes en fonction de leurs caractéristiques ou de leurs attributs. L'objectif principal de la classification est de prédire l'appartenance d'un nouvel élément à une classe donnée, en se basant sur les caractéristiques observées dans les données d'entraînement.

1.1 Importance de la classification

L'importance de la classification réside dans sa capacité à fournir des informations utiles et exploitables à partir de données brutes. Voici quelques-unes des raisons pour lesquelles la classification est largement utilisée en analyse de données :

- **Organisation des données** : La classification permet de structurer et d'organiser des ensembles de données complexes en regroupant des éléments similaires dans des catégories distinctes.
- **Prédiction et reconnaissance** : En identifiant des modèles dans les données, la classification permet de prédire ou de reconnaître des éléments inconnus et de prendre des décisions éclairées sur leur classification.
- **Segmentation de marché** : Dans le domaine du marketing, la classification est utilisée pour segmenter les clients en groupes homogènes afin de mieux cibler les offres et les campagnes publicitaires.

En résumé, la classification joue un rôle crucial dans l'analyse de données en permettant d'extraire des informations significatives à partir de données complexes, ce qui facilite la prise de décisions éclairées dans de nombreux domaines d'application.

1.2 Introduction aux méthodes de classification hiérarchique et de k-means

Les méthodes de classification hiérarchique et de k-means sont deux techniques fondamentales largement utilisées dans divers domaines tels que la science des données,la finance, le marketing, etc., pour extraire des informations significatives à partir de données brutes.

La classification hiérarchique est une approche qui divise progressivement les données en clusters plus petits, formant ainsi une structure arborescente ou dendrogramme. Cette méthode peut être divisée en deux approches principales : la classification hiérarchique agglomérative et la classification hiérarchique divisive. La première commence par considérer chaque point de données comme un cluster séparé et fusionne ensuite progressivement les clusters les plus proches, tandis que la seconde commence par considérer tous les points de données comme un seul cluster et divise ensuite récursivement le cluster en clusters plus petits.

D'autre part, l'algorithme k-means est une méthode itérative de partitionnement de données en k clusters, où k est un nombre prédéfini. Cet algorithme commence par initialiser aléatoirement les centroïdes des clusters, puis il alterne entre deux étapes : attribution

des points de données au cluster dont le centroïde est le plus proche et mise à jour des centroïdes en calculant les moyennes des points de données dans chaque cluster. Ce processus est répété jusqu'à ce qu'une convergence soit atteinte, c'est-à-dire que les centroïdes ne changent pas de manière significative entre les itérations successives.

La classification hiérarchique est souvent préférée lorsque la structure hiérarchique des clusters est importante et que le nombre de clusters n'est pas prédéfini, tandis que k-means est plus adapté lorsque le nombre de clusters est connu à l'avance et que les clusters sont globulaires et bien séparés.

1.3 Objectifs du rapport

L'objectif principal de ce rapport est de présenter en détail le projet de développement d'un programme de classification utilisant les méthodes de classification hiérarchique (CAH) et de k-means en Python. Les objectifs spécifiques sont les suivants :

- Démontrer l'utilisation des fonctionnalités interactives de l'interface utilisateur pour faciliter l'analyse et la visualisation des données.
- Expliquer le choix des graphiques et des visualisations utilisés pour représenter les résultats de la classification, en mettant en évidence leur pertinence et leur utilité.

1.4 Description du projet

Le projet consiste à développer un programme de classification en Python, offrant aux utilisateurs une interface conviviale pour l'analyse de données à l'aide des méthodes de CAH et de k-means. Voici une description détaillée du projet :

- **Fonctionnalités du programme** : Le programme permet aux utilisateurs de charger des ensembles de données à partir de fichiers Excel, d'appliquer les algorithmes de classification (CAH et k-means) et d'explorer les résultats à travers des visualisations interactives.
- **Interface utilisateur intuitive** : L'interface utilisateur offre une expérience conviviale, permettant aux utilisateurs de paramétrer les algorithmes, d'interagir avec les données et de visualiser les résultats de manière intuitive.
- **Documentation du code** : Le rapport fournira une documentation complète du code source du programme, expliquant chaque fonction, classe et module utilisé, ainsi que leur contribution à la fonctionnalité globale du programme.
- **Exemples d'utilisation** : Le rapport présentera des exemples d'utilisation du programme avec différents ensembles de données, démontrant les capacités d'analyse et de visualisation offertes par les méthodes de CAH et de k-means.
- **Analyse des résultats** : En plus de présenter les fonctionnalités du programme, le rapport analysera les résultats obtenus avec différents ensembles de données, mettant en évidence les avantages et les limitations des méthodes de classification utilisées.

2 Explication des méthodes :

2.1 Explication détaillée de la CAH :

2.1.1 Principe de fonctionnement

La classification hiérarchique est une méthode de classification itérative qui permet de regrouper des individus en différentes classes selon un critère de ressemblance défini au préalable. Cette méthode est représentée par un dendrogramme, qui montre les différents niveaux de regroupement des individus.

La classification est ascendante car elle part des observations individuelles, et elle est hiérarchique car elle produit des classes ou groupes de plus en plus vastes, incluant des sous-groupes en leur sein. En découpant cet arbre à une certaine hauteur choisie, on produira la partition désirée.

Le critère de ressemblance, utilisé dans la classification hiérarchique, est calculé en se basant sur les mesures de distance ou de similarité entre les individus. Ces mesures peuvent varier en fonction du type de données et du contexte de l'analyse. Voici quelques exemples de calcul de critères de ressemblance couramment utilisés :

— **Distance euclidienne :** Pour des données numériques continues, la distance euclidienne entre deux individus i et j peut être calculée comme suit :

$$d_{ij} = \sqrt{\sum_{k=1}^{n}(x_{ik} - x_{jk})^2}$$

où x_{ik} et x_{jk} sont les valeurs des variables pour les individus i et j respectivement, et n est le nombre de variables.

Le coefficient de corrélation est une mesure statistique qui exprime à quel point deux ensembles de données sont liés linéairement les uns aux autres. Pour calculer le coefficient de corrélation, nous utilisons généralement le coefficient de corrélation de Pearson, qui est largement utilisé pour les données numériques continues. Voici comment il est calculé :

— **Calcul des moyennes** : Calculer les moyennes des ensembles de données X et Y, notées respectivement par $\bar{x}$ et $\bar{y}$:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

— **Calcul des écarts-types** : Calculer les écarts-types des ensembles de données X et Y, notés respectivement par s_x et s_y :

$$s_x = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$s_y = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

— **Calcul de la covariance** : Calculer la covariance entre les ensembles de données X et Y, notée $\text{cov}(X, Y)$:

$$\text{cov}(X, Y) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$$

— **Calcul du coefficient de corrélation de Pearson** : Utiliser la formule suivante pour calculer le coefficient de corrélation r :

$$r = \frac{\text{cov}(X, Y)}{s_x s_y}$$

Le coefficient de corrélation de Pearson varie entre -1 et 1. Un coefficient de 1 indique une corrélation linéaire positive parfaite, -1 indique une corrélation linéaire négative parfaite, et 0 indique l'absence de corrélation linéaire entre les ensembles de données.

Matrice de distance

Choix d'une mesure de distance : Sélectionnez une mesure de distance appropriée en fonction de la nature de vos données. Les mesures de distance couramment utilisées incluent :

— **Distance euclidienne**
— **Distance de Manhattan** (ou distance en "city block")

Calcul des distances entre les paires d'observations : Pour chaque paire d'observations i et j dans votre ensemble de données, calculez la distance selon la mesure choisie. Par exemple, si vous utilisez la distance euclidienne entre deux observations $\mathbf{x}_i$ et $\mathbf{x}_j$ dans un espace n-dimensionnel, la formule est :

Vous répétez le calcul de distance euclidienne pour toutes les paires d'observations pour construire la matrice de distance.

Construction de la matrice de distance : Une fois que vous avez calculé les distances entre toutes les paires d'observations, vous les stockez dans une matrice. Cette matrice est appelée matrice de distance et a la forme d'une matrice carrée $n \times n$, où n est le nombre d'observations dans votre ensemble de données. Chaque élément de la matrice représente la distance entre deux observations.

Construction de l'arbre de clustering :

Ensuite, un algorithme de clustering est appliqué pour regrouper les individus en clusters en fonction de leurs similarités mesurées par la matrice de distance. L'algorithme de clustering le plus couramment utilisé pour la création de dendrogrammes est l'algorithme de clustering hiérarchique.

Construction du dendrogramme :

Le dendrogramme est construit en utilisant l'arbre de clustering résultant. Chaque individu est représenté par une feuille dans le dendrogramme, et les branches de l'arbre indiquent les regroupements successifs des individus en clusters. La longueur des branches représente la distance entre les individus ou les clusters. Plus la distance est grande, moins les individus sont similaires.

Interprétation du dendrogramme :

Le dendrogramme peut être interprété en coupant les branches à différentes hauteurs pour former des clusters. En coupant le dendrogramme à différentes hauteurs, vous

pouvez obtenir différents niveaux de granularité dans le regroupement des individus. Le choix de la hauteur de coupe dépend souvent du contexte de l'analyse et des objectifs spécifiques de regroupement des données.

En résumé, le dendrogramme est une représentation graphique qui permet de visualiser les relations de regroupement entre les individus dans un ensemble de données, en utilisant une approche hiérarchique basée sur la distance entre les individus.

2.1.2 Avantages de la CAH :

La CAH présente plusieurs avantages par rapport à d'autres méthodes de classification :

- **Flexibilité dans le choix du type de dissimilarité :** La CAH permet le choix d'un type de dissimilarité adapté au sujet d'étude et à la nature des données. Cette flexibilité permet d'appliquer la CAH à différents types de données et contextes.
- **Visualisation de la progression du regroupement :** La CAH produit un dendrogramme qui permet de visualiser la progression du regroupement des données. Cette visualisation facilite la détermination d'un nombre approprié de classes dans lesquelles les données peuvent être regroupées.
- **Adaptabilité à différents types de données :** La CAH peut être appliquée à différents types de données, telles que les données quantitatives ou qualitatives, et peut être adaptée à des contextes spécifiques, tels que le marketing, les médias ou l'économie.
- **Représentation hiérarchique :** La CAH produit une représentation hiérarchique des données, ce qui permet d'identifier différents niveaux de granularité dans la classification. Cette représentation hiérarchique peut être utile pour identifier des sous-groupes au sein de groupes plus larges ou pour identifier des clusters à différents niveaux de similarité.
- **Choix de la méthode d'agrégation :** La CAH permet le choix de la méthode d'agrégation, qui peut être adaptée au contexte spécifique des données et aux objectifs de l'analyse. Le choix de la méthode d'agrégation peut avoir un impact sur les résultats de l'analyse et l'interprétation des données.

2.1.3 Inconvénients de la CAH :

La Classification Ascendante Hiérarchique (CAH) présente plusieurs avantages par rapport à d'autres méthodes de classification, mais elle présente également certains inconvénients :

- **Choix de la mesure de dissimilarité :** Bien que la CAH permette le choix d'une mesure de dissimilarité, ce choix peut être subjectif et peut affecter les résultats. D'autres méthodes de classification, telles que k-means, ne nécessitent pas le choix d'une mesure de dissimilarité.
- **Sensibilité aux valeurs aberrantes :** La CAH est sensible aux valeurs aberrantes, ce qui peut affecter significativement les résultats. D'autres méthodes de classification, telles que k-medoids, sont plus robustes aux valeurs aberrantes.

— **Complexité computationnelle :** La CAH peut être coûteuse en termes de calcul, surtout pour de grands ensembles de données. D'autres méthodes de classification, comme k-means, sont généralement plus rapides et plus évolutives.
— **Difficulté à choisir le nombre de clusters :** La CAH ne fournit pas de moyen clair de déterminer le nombre optimal de clusters. D'autres méthodes de classification, telles que la méthode du coude ou la méthode du silhouette, peuvent aider à déterminer le nombre optimal de clusters.
— **Absence d'interprétation probabiliste :** La CAH ne fournit pas d'interprétation probabiliste des clusters, ce qui peut limiter son utilité dans certaines applications. D'autres méthodes de classification, comme les modèles de mélange gaussien, fournissent une interprétation probabiliste des clusters.
— **Limitation aux structures hiérarchiques :** La CAH est limitée aux structures hiérarchiques, ce qui peut ne pas être approprié pour tous les ensembles de données. D'autres méthodes de classification, comme k-means, n'imposent pas une structure hiérarchique aux données.

2.1.4 Principes et Fonctionnement du code de CAH sur Python :

2.1.5 Packages utilisés :

```
import streamlit as st
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import linkage, dendrogram
from sklearn.metrics import pairwise_distances
```

Listing 1 – python script1

Ce code Python utilise plusieurs packages pour effectuer différentes tâches. Voici le rôle de chaque package utilisé :

numpy (np) : NumPy est une bibliothèque Python utilisée pour effectuer des calculs numériques. Dans ce code, NumPy est utilisé pour manipuler des tableaux et effectuer des calculs matriciels, tels que le calcul des distances euclidiennes entre les points de données. **matplotlib.pyplot (plt)** : Matplotlib est une bibliothèque de visualisation en Python. Ici, le sous-module pyplot de Matplotlib est utilisé pour créer des graphiques, notamment pour afficher le dendrogramme généré par le clustering hiérarchique.

pandas (pd) : Pandas est une bibliothèque Python utilisée pour la manipulation et l'analyse des données. Dans ce code, Pandas est utilisé pour importer les données à partir d'un fichier Excel et les manipuler sous forme de DataFrame, ce qui facilite le travail avec les données tabulaires.

from scipy.cluster.hierarchy import linkage, dendrogram : Scipy est une bibliothèque Python qui contient des outils pour le calcul scientifique et l'analyse de données.Elle offre plusieurs sous-modules, dont scipy.cluster.hierarchy, qui contient des fonctions pour le clustering hiérarchique,les fonctions linkage et dendrogram de Scipy effectuent le clustering hiérarchique et visualisent les résultats sous forme de

dendrogramme.
from sklearn.metrics import pairwise-distances : Scikit-learn est une bibliothèque d'apprentissage automatique et l'exploration des données,elle propose les algorithmes d'apprentissage automatique, ainsi que des outils pour évaluer les modèles, effectuer du prétraitement des données et plus encore, la fonction pairwise-distances de Scikit-learn calcule les distances entre les paires de points dans vos données, ce qui est nécessaire pour le clustering hiérarchique.

2.1.6 Classes utilisés :

```
class HierarchicalClustering:
    def __init__(self, k):
        self.k = k

    def fit(self, data, distances):
        n = len(data)
        clusters = [[i] for i in range(n)]

        while len(clusters) > self.k:
            min_dist = np.inf
            for i in range(len(clusters)):
                for j in range(i + 1, len(clusters)):
                    cluster1 = clusters[i]
                    cluster2 = clusters[j]
                    avg_dist = 0
                    count = 0
                    for idx1 in cluster1:
                        for idx2 in cluster2:
                            if not np.isnan(distances[idx1, idx2]):
                                avg_dist += distances[idx1, idx2]
                                count += 1
                    if count > 0:
                        avg_dist /= count
                        if avg_dist < min_dist:
                            min_dist = avg_dist
                            merge_index = (i, j)
            i, j = merge_index
            new_cluster = clusters[i] + clusters[j]
            clusters[i] = new_cluster
            clusters.pop(j)

        self.labels_ = np.zeros(n)
        for i, cluster in enumerate(clusters):
            for idx in cluster:
                self.labels_[idx] = i
```

Listing 2 – python script1

La classe HierarchicalClustering implémente l'algorithme de clustering hiérarchique. Voici le rôle et le fonctionnement détaillé de chaque méthode de cette classe :
init(self, k) : Cette méthode est le constructeur de la classe. Elle initialise un objet de la classe HierarchicalClustering avec un paramètre k qui représente le nombre de clusters souhaités.

fit(self, data, distance) : Cette méthode prend en entrée les données à clusteriser et applique l'algorithme de clustering hiérarchique. Voici son fonctionnement :

n = len(data : La ligne calcule le nombre d'observations ou de lignes dans les données pour obtenir une indication de la taille des données sur lesquelles l'algorithme de clustering hiérarchique sera appliqué.

self.labels = np.zeros(n) : assignant le tableau de zéros à self.labels, chaque point de données est initialement attribué au cluster 0. Cette initialisation est typique avant d'exécuter l'algorithme de clustering, où les étiquettes de cluster sont mises à jour au fur et à mesure de la convergence de l'algorithm.

Applique l'algorithme de clustering hiérarchique : Tant que le nombre de clusters est supérieur à k, fusionne les clusters les plus proches en termes de distance moyenne. Met à jour les distances entre les clusters fusionnés. Répète le processus jusqu'à ce que le nombre de clusters atteigne k.
Affecte des étiquettes de cluster à chaque point de données en fonction des clusters formés.
En résumé, la classe HierarchicalClustering encapsule l'algorithme de clustering hiérarchique, permettant de créer des clusters à partir de données et d'attribuer des étiquettes de cluster à chaque point de données en fonction de leur proximité.

2.1.7 Fonctions utilisés :

```
def import_data_from_excel(file_path):
    data = pd.read_excel(file_path)
    # Check if the first row or first column contains non-numeric values
    first_row_is_str = any(isinstance(val, str) for val in data.iloc[0])
    first_column_is_str = isinstance(data.iloc[:, 0].values[0], str)
    if first_row_is_str or first_column_is_str:
        data = data.iloc[1:, 1:]  # Exclude first row and first column if they contain non-numeric values
    return data
```

Listing 3 – python script1

import-data-from-excel(file-path) : Cette fonction importe les données à partir d'un fichier Excel. Voici son fonctionnement :

data = pd.read-excel(file-path) : Utilise la fonction readexcel de la bibliothèque Pandas pour lire les données à partir du fichier Excel spécifié par file-path. Et vérifie si la première ligne ou la première colonne contient des valeurs non numériques. Si la première ligne ou la première colonne contient des valeurs non numériques, les exclut du DataFrame en utilisant iloc[1 :, 1 :].

Return data : Renvoie le DataFrame contenant les données, avec les ajustements effectués si nécessaire.

Cette fonction est utile pour importer des données tabulaires à partir d'un fichier Excel tout en gérant les cas où les premières lignes ou colonnes peuvent contenir des en-têtes ou des valeurs non numériques

```python
def dendrogram_visualization(data, labels):
    X = data.values
    Z = linkage(X, method='ward')
    fig, ax = plt.subplots(figsize=(10, 5))
    dn = dendrogram(Z, ax=ax)
    ax.set_title("Dendrogram")
    ax.set_xlabel("Data Points")
    ax.set_ylabel("Distance")
    st.pyplot(fig)  # Display the plot in Streamlit
```

Listing 4 – python script1

dendrogram-visualization(data, labels) : La fonction est utilisée pour visualiser un dendrogramme à partir des données de clustering et des étiquettes de cluster associées. Voici un aperçu de ce que fait cette fonction :
Z = linkage(X, method='ward') : il calcule la matrice de liaison à partir des données X.
dn = dendrogram(Z, ax=ax) : il trace le dendrogramme à partir de la matrice de liaison Z sur les axes spécifiés ax.

2.2 Explication détaillée de la méthode kmeans :

2.2.1 Principe de fonctionnement :

K-means est un algorithme de clustering populaire qui vise à partitionner un ensemble de données en K clusters distincts et non chevauchants. L'algorithme affecte itérativement chaque point de données au centroïde le plus proche, puis met à jour les centroïdes en fonction de la moyenne des points de données assignés à chaque cluster. L'algorithme fonctionne comme suit :

— **1-Initialisation** :
  — K centroïdes sont initialisés de manière aléatoire à partir de l'ensemble de données.

— **2-Affectation** :
  — Chaque point de données est assigné au centroïde le plus proche en fonction d'une distance choisie, telle que la distance euclidienne et la distance de Manhattan. A noter : Soit un espace n-dimensionnel, la distance entre deux points p et q est comme suit :

$$\text{Distance euclidienne}(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

$$\text{Distance de Manhattan(City Block)}(p,q) = \sqrt{\sum_{i=1}^{n}|p_i - q_i|}$$

  — Assignation de l'observation : L'observation est assignée au cluster dont le centroid est le plus proche, en fonction de la distance calculée.

3-Mise à jour :

- Les centroïdes sont mis à jour en calculant la moyenne de tous les points de données assignés à chaque cluster.

4-Itération :

- Les étapes 2 et 3 sont répétées jusqu'à convergence, c'est-à-dire lorsque les centroïdes ne changent plus significativement ou qu'un nombre maximal d'itérations est atteint.

L'algorithme K-Means est sensible au choix initial des centroids et peut converger vers un optimum local. Pour atténuer cet effet, des techniques telles que l'initialisation K-Means++ peuvent être utilisées pour initialiser les centroids de manière plus intelligente. De plus, le choix du nombre de clusters k est souvent déterminé en utilisant des méthodes.

2.2.2 Avantages de K-means :

K-means est un algorithme de clustering qui présente plusieurs avantages qui en font un choix populaire pour l'analyse de données et les tâches d'apprentissage automatique.

— **Facilité d'utilisation :** K-means est un algorithme simple et facile à mettre en œuvre qui ne nécessite pas de calculs mathématiques complexes ou de compétences en programmation avancées.

— **Scalabilité :** K-means peut traiter de grands ensembles de données avec des milliers de points de données et de caractéristiques, ce qui le rend adapté aux applications de big data.

— **Efficacité :** K-means est un algorithme rapide qui peut converger rapidement vers une solution, même pour de grands ensembles de données.

— **Interprétabilité :** K-means produit des résultats facilement interprétables sous forme de clusters qui peuvent être visualisés et analysés pour obtenir des informations sur les données.

— **Flexibilité :** K-means peut être utilisé avec différents types de données, y compris numériques, catégorielles et binaires.

— **Robustesse :** K-means est robuste aux valeurs aberrantes et au bruit dans les données, ce qui peut améliorer la qualité des clusters.

— **Adaptabilité :** K-means peut être adapté à différents critères d'optimisation, tels que la minimisation de la somme des erreurs au carré ou la maximisation du score silhouette.

2.2.3 Inconvénients de K-means :

Les inconvénients de l'algorithme de clustering K-means comprennent :

— **Sensibilité aux Conditions Initiales :** K-means est sensible au placement initial des centroids, ce qui peut conduire à différentes affectations de clusters et résultats en fonction de la sélection aléatoire initiale des centroids.

— **Nécessité de Pré-définir le Nombre de Clusters (K) :**Une des limitations de K-means est que l'utilisateur doit spécifier le nombre de clusters au préalable, ce qui peut être difficile à déterminer avec précision en pratique.

— **Absence de Développement d'un Ensemble de Clusters Optimal :** K-means ne fournit pas intrinsèquement un ensemble optimal de clusters et nécessite que l'utilisateur pré-définisse le nombre de clusters, ce qui peut ne pas toujours correspondre à la structure sous-jacente des données.

— **Incohérence des Résultats :** K-means peut produire des résultats variables entre différentes exécutions de l'algorithme en raison de l'initialisation aléatoire des centroids, ce qui entraîne une incohérence dans les résultats du clustering.

— **Hypothèse de Clusters Sphériques :** K-means suppose que les clusters sont sphériques et isotropes, ce qui peut ne pas être vrai pour tous les types de distributions de données, limitant son applicabilité dans certaines situations.

— **Difficulté dans le Choix du Nombre de Clusters :** Sélectionner le nombre optimal de clusters (K) dans K-means peut être difficile, car cela nécessite une réflexion attentive et peut ne pas toujours avoir de solution simple.

Ces inconvénients mettent en lumière certaines des limitations et des défis associés à l'algorithme de clustering K-means, soulignant l'importance de comprendre ses contraintes et de considérer des méthodes de clustering alternatives lorsque c'est nécessaire.

2.2.4 Principes et Fonctionnement du code de kmeans sur Python :

2.2.5 Packages utilisés :

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
import plotly.graph_objs as go
import plotly.express as px
```

Listing 5 – python script1

pandas (pd) :Pandas est une bibliothèque Python qui fournit des structures de données et des outils d'analyse de données faciles à utiliser. La structure de données principale de pandas est le DataFrame, qui est une structure de données tabulaire bidimensionnelle avec des étiquettes d'axe (lignes et colonnes). Les fonctionnalités de pandas incluent le chargement de données depuis diverses sources, la manipulation et la transformation des données, ainsi que l'analyse exploratoire des données.

numpy (np) :NumPy est une bibliothèque Python pour le calcul numérique qui prend en charge les tableaux multidimensionnels et les fonctions mathématiques pour les manipuler. NumPy fournit des outils efficaces pour effectuer des opérations mathématiques et statistiques sur les tableaux, ce qui en fait une bibliothèque fondamentale pour la science des données et l'apprentissage automatique.

sklearn.decomposition.PCA :PCA (Principal Component Analysis) est une technique de réduction de dimensionnalité largement utilisée pour compresser les données en conservant les principales caractéristiques. L'objet PCA de scikit-learn permet d'effectuer une analyse en composantes principales sur les données pour projeter les observations dans un espace de dimension inférieure.

sklearn.preprocessing.StandardScaler :StandardScaler de scikit-learn est une méthode de prétraitement qui standardise les fonctionnalités en supprimant la moyenne et en mettant à l'échelle jusqu'à la variance unitaire. Cela permet de normaliser les caractéristiques des données, ce qui est souvent nécessaire pour de nombreux algorithmes d'apprentissage automatique qui supposent que les données sont centrées et ont des variances similaires.

sklearn.metrics.silhouette-score :La silhouette est une mesure de la cohésion et de la séparation des clusters dans un ensemble de données. silhouette-score de scikit-learn calcule la silhouette moyenne sur toutes les observations, fournissant ainsi une indication de la qualité du clustering.

plotly.graph-objs (go) :Plotly est une bibliothèque graphique interactive qui permet de créer des graphiques et des visualisations de données hautement personnalisables. plotly.graph-objs fournit une interface basée sur des objets pour créer des graphiques, permettant un contrôle précis sur les paramètres de visualisation tels que les axes, les couleurs et les annotations.

plotly.express (px) :Plotly Express est une interface de haut niveau pour créer des visualisations de données avec Plotly. Il offre une syntaxe simple pour créer rapidement des graphiques interactifs tels que des diagrammes à barres, des diagrammes circulaires, des nuages de points et bien d'autres, avec une prise en charge automatique de la mise en forme et de la configuration des graphiques.

2.2.6 Classes utilisés :

```python
class KMeansClustering:
    def __init__(self, k=3, distance_metric='euclidean', random_state=None):
        self.k = k
        self.distance_metric = distance_metric
        self.centroids = None
        self.random_state = random_state

    def calculate_distance(self, data_point, centroids):
        if self.distance_metric == 'euclidean':
            return np.sqrt(np.sum((centroids - data_point) ** 2, axis=1))
        elif self.distance_metric == 'city_block':
            return np.sum(np.abs(centroids - data_point), axis=1)
        else:
            raise ValueError("Invalid distance metric. Supported metrics are 'euclidean' and 'city_block'.")

    def fit(self, X, max_iterations=200):
        np.random.seed(self.random_state)  # Set the random seed

        # Initialize centroids randomly
        self.centroids = X[np.random.choice(X.shape[0], self.k, replace=False)]

        inertia_values = []

        for _ in range(max_iterations):
            # Assign each data point to the nearest centroid
            distances = np.vstack([self.calculate_distance(data_point, self.centroids) for data_point in X])
            cluster_assignment = np.argmin(distances, axis=1)

            # Update centroids
            new_centroids = np.array([X[cluster_assignment == i].mean(axis=0) for i in range(self.k)])

            # Calculate inertia
            inertia = np.sum((X - new_centroids[cluster_assignment]) ** 2)
            inertia_values.append(inertia)

            # Check convergence
            if np.allclose(self.centroids, new_centroids):
                break

            self.centroids = new_centroids

        # Calculate silhouette score
        labels = np.argmin(distances, axis=1)
        silhouette_avg = silhouette_score(X, labels)

        return cluster_assignment, self.centroids, inertia_values, silhouette_avg
```

Listing 6 – python script1

Classe `KMeansClustering` : La classe `KMeansClustering` représente l'algorithme de clustering K-means. Elle possède les méthodes suivantes :

- `__init__` : Initialise la classe avec les paramètres tels que le nombre de clusters, la métrique de distance, etc.
- `calculate_distance` : Calcule la distance entre un point de données donné et tous les centroids actuels.
- `fit` : Cette méthode effectue l'algorithme de clustering K-Means. Elle attribue itérativement chaque point de données au centroïde le plus proche, met à jour les centroïdes en fonction de la moyenne des points de données attribués, et répète le processus jusqu'à ce qu'il y ait convergence ou jusqu'à ce que le nombre maximal d'itérations (max-iterations) soit atteint. Elle calcule également les valeurs d'inertie (somme des distances au carré des échantillons jusqu'au centre de leur cluster le plus proche), retourne les affectations de cluster, les centroïdes finaux, les valeurs d'inertie au fil des itérations, et le score de silhouette comme mesure de qualité du clustering.

2.2.7 Fonctions utilisés :

```
def run_clustering(X, k, distance_metric, use_pca):
    if use_pca:
        pca = PCA(n_components=2)
        X = pca.fit_transform(X)

    kmeans = KMeansClustering(k=k, distance_metric=distance_metric,
   random_state=42)
    labels, centroids, inertia_values, silhouette_avg = kmeans.fit(X)
    return labels, centroids, inertia_values, silhouette_avg
```

Listing 7 – python script1

Fonction `run_clustering` : La fonction `run_clustering` exécute le processus complet de clustering K-means sur les données fournies. Elle prend en entrée les données, le nombre de clusters, la métrique de distance à utiliser et un indicateur pour indiquer si une PCA doit être utilisée.

```
def plot_result(X_reduced, labels, centroids):
    if X_reduced.shape[1] == 2:
        df = pd.DataFrame(X_reduced, columns=['Component 1', 'Component
   2'])
    else:
        df = pd.DataFrame(X_reduced[:, :2], columns=['Component 1', '
   Component 2'])
    df['Cluster'] = labels.astype(str)  # Convert labels to string for
   coloring purposes

    fig = px.scatter(df, x='Component 1', y='Component 2', color='
   Cluster',
                     title='Clustering Result', hover_name=df.index)
    fig.add_trace(go.Scatter(
        x=centroids[:, 0],
        y=centroids[:, 1],
        mode='markers',
        marker=dict(size=10, color='black', symbol='star'),
```

```
        name='Centroids'
    ))
    return fig
```

Listing 8 – python script1

Fonction `plot_result` : La fonction `plot_result` génère un graphique interactif de la visualisation des résultats de clustering. Elle prend en entrée les données réduites en dimensionnalité, les affectations de cluster et les centroids finaux.

```
def plot_convergence(inertia_values):
    fig = go.Figure()
    fig.add_trace(go.Scatter(
        x=list(range(1, len(inertia_values) + 1)),
        y=inertia_values,
        mode='lines+markers',
        marker=dict(color='blue'),
        name='Inertia'
    ))
    fig.update_layout(
        title='Convergence Plot',
        xaxis=dict(title='Iteration'),
        yaxis=dict(title='Inertia'),
        hovermode='closest'
    )
    return fig
```

Listing 9 – python script1

Fonction `plot_convergence` : La fonction `plot_convergence` génère un graphique interactif montrant la convergence de l'algorithme K-means au fil des itérations. Elle prend en entrée les valeurs d'inertie à chaque itération.

```
def plot_silhouette_score(silhouette_values):
    fig = go.Figure()
    fig.add_trace(go.Scatter(
        x=list(range(2, 12)),
        y=silhouette_values,
        mode='lines+markers',
        marker=dict(color='green'),
        name='Silhouette Score'
    ))
    fig.update_layout(
        title='Silhouette Score',
        xaxis=dict(title='Number of Clusters (K)'),
        yaxis=dict(title='Silhouette Score'),
        hovermode='closest'
    )
    return fig
```

Listing 10 – python script1

Fonction `plot_silhouette_score` : La fonction `plot_silhouette_score` génère un graphique interactif montrant l'évolution du score de silhouette en fonction du nombre de clusters. Elle prend en entrée les valeurs de score de silhouette pour différents nombres de clusters.

3 Interface graphique(kmeans, CAH) :

3.1 Code total sur python :

```
import streamlit as st
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
import plotly.graph_objs as go
import plotly.express as px
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import linkage, dendrogram  # Modified import statement
import base64
import requests

# Set background image
background_image_url = "https://img.freepik.com/free-vector/ai-technology-brain-background-vector-digital-transformation-concept_53876-117820.jpg?w=900&t=st=1713012632~exp=1713013232~hmac=05da65e2d9e6da77202ded9006cfecb86c0c11fbc70346fb0532307b18d8ac3f"

def get_base64_of_bin_file(url):
    data = requests.get(url).content
    return base64.b64encode(data).decode()

def set_png_as_page_bg(png_file):
    bin_str = get_base64_of_bin_file(png_file)
    st.markdown(
        f'<style>'
        f'.stApp {{'
        f'background-image: url("{bin_str}");'
        f'background-size: cover;'
        f'}}'
        f'</style>',
        unsafe_allow_html=True
    )

set_png_as_page_bg(background_image_url)

class KMeansClustering:
    def __init__(self, k=3, distance_metric='euclidean', random_state=None):
        self.k = k
        self.distance_metric = distance_metric
        self.centroids = None
        self.random_state = random_state

    def calculate_distance(self, data_point, centroids):
        if self.distance_metric == 'euclidean':
            return np.sqrt(np.sum((centroids - data_point) ** 2, axis=1))
        elif self.distance_metric == 'city_block':
            return np.sum(np.abs(centroids - data_point), axis=1)
```

```
        else:
            raise ValueError("Invalid distance metric. Supported metrics are 'euclidean' and 'city_block'.")

    def fit(self, X, max_iterations=200):
        np.random.seed(self.random_state)  # Set the random seed

        # Initialize centroids randomly
        self.centroids = X[np.random.choice(X.shape[0], self.k, replace=False)]

        inertia_values = []

        for _ in range(max_iterations):
            # Assign each data point to the nearest centroid
            distances = np.vstack([self.calculate_distance(data_point, self.centroids) for data_point in X])
            cluster_assignment = np.argmin(distances, axis=1)

            # Update centroids
            new_centroids = np.array([X[cluster_assignment == i].mean(axis=0) for i in range(self.k)])

            # Calculate inertia
            inertia = np.sum((X - new_centroids[cluster_assignment]) ** 2)
            inertia_values.append(inertia)

            # Check convergence
            if np.allclose(self.centroids, new_centroids):
                break

            self.centroids = new_centroids

        # Calculate silhouette score
        labels = np.argmin(distances, axis=1)
        silhouette_avg = silhouette_score(X, labels)

        return cluster_assignment, self.centroids, inertia_values, silhouette_avg

class HierarchicalClustering:
    def __init__(self, k):
        self.k = k

    def fit(self, data):
        self.data = data
        n = len(data)
        distances = np.zeros((n, n))

        # Calculate pairwise distances
        for i in range(n):
            for j in range(n):
                distances[i, j] = self.euclidean_distance(data[i], data[j])

        # Initialize clusters
```

```
97          clusters = [[i] for i in range(n)]
98
99          # Hierarchical clustering algorithm
100         while len(clusters) > self.k:
101             min_dist = np.inf
102             for i in range(len(clusters)):
103                 for j in range(i + 1, len(clusters)):
104                     cluster1 = clusters[i]
105                     cluster2 = clusters[j]
106                     avg_dist = 0
107                     count = 0
108                     for idx1 in cluster1:
109                         for idx2 in cluster2:
110                             if not np.isnan(distances[idx1, idx2]):
111                                 avg_dist += distances[idx1, idx2]
112                                 count += 1
113                     if count > 0:
114                         avg_dist /= count
115                         if avg_dist < min_dist:
116                             min_dist = avg_dist
117                             merge_index = (i, j)
118             i, j = merge_index
119             new_cluster = clusters[i] + clusters[j]
120             clusters[i] = new_cluster
121             clusters.pop(j)
122
123         self.labels_ = np.zeros(n)
124         for i, cluster in enumerate(clusters):
125             for idx in cluster:
126                 self.labels_[idx] = i
127
128     def euclidean_distance(self, x1, x2):
129         x1 = np.array(x1, dtype=np.float64)  # Convert x1 to numpy array
    of numbers
130         x2 = np.array(x2, dtype=np.float64)  # Convert x2 to numpy array
    of numbers
131         if np.any(np.isnan(x1)) or np.any(np.isnan(x2)):
132             return np.nan
133         return np.sqrt(np.sum((x1 - x2) ** 2))
134
135 def plot_dendrogram(data):
136     Z = linkage(data, method='ward')  # Use linkage from scipy.cluster.
    hierarchy directly
137     plt.figure(figsize=(10, 5))
138     dn = dendrogram(Z)
139     plt.title("Dendrogram")
140     plt.xlabel("Data Points")
141     plt.ylabel("Distance")
142     st.pyplot(plt.gcf())  # Pass the current figure as an argument to st
    .pyplot()
143
144 def plot_result(X_reduced, labels, centroids):
145     if X_reduced.shape[1] == 2:
146         df = pd.DataFrame(X_reduced, columns=['Component 1', 'Component
    2'])
147     else:
148         df = pd.DataFrame(X_reduced[:, :2], columns=['Component 1', '
```

```
    Component 2'])
    df['Cluster'] = labels.astype(str)  # Convert labels to string for
    coloring purposes

    fig = px.scatter(df, x='Component 1', y='Component 2', color='
    Cluster',
                         title='Clustering Result', hover_name=df.index)
    fig.add_trace(go.Scatter(
        x=centroids[:, 0],
        y=centroids[:, 1],
        mode='markers',
        marker=dict(size=10, color='black', symbol='star'),
        name='Centroids'
    ))
    return fig

def plot_convergence(inertia_values):
    fig = go.Figure()
    fig.add_trace(go.Scatter(
        x=list(range(1, len(inertia_values) + 1)),
        y=inertia_values,
        mode='lines+markers',
        marker=dict(color='blue'),
        name='Inertia'
    ))
    fig.update_layout(
        title='Convergence Plot',
        xaxis=dict(title='Iteration'),
        yaxis=dict(title='Inertia'),
        hovermode='closest'
    )
    return fig

def plot_silhouette_score(silhouette_values):
    fig = go.Figure()
    fig.add_trace(go.Scatter(
        x=list(range(2, 12)),
        y=silhouette_values,
        mode='lines+markers',
        marker=dict(color='green'),
        name='Silhouette Score'
    ))
    fig.update_layout(
        title='Silhouette Score',
        xaxis=dict(title='Number of Clusters (K)'),
        yaxis=dict(title='Silhouette Score'),
        hovermode='closest'
    )
    return fig

def main():
    st.title("Clustering")

    selected = st.radio("Select Option", ["Home", "KMeans Clustering", "
    Hierarchical Clustering"])

    if selected == "KMeans Clustering":
```

```
202         st.title("KMeans Clustering")
203         uploaded_file = st.file_uploader("Upload Excel file", type=["
    xlsx", "xls"])
204         if uploaded_file is not None:
205             data = pd.read_excel(uploaded_file, index_col=0)
206             st.write("Data imported successfully!")
207 
208             X = data.values
209             X = X[:, 2:]  # Adjust this according to your data
210 
211             scaler = StandardScaler()
212             X_scaled = scaler.fit_transform(X)
213 
214             k = st.sidebar.slider("Number of Clusters (K)", min_value=2,
     max_value=10, value=3)
215             distance_metric = st.sidebar.radio("Distance Metric", ["
    Euclidean", "City Block"], index=0)
216             distance_metric = 'euclidean' if distance_metric == '
    Euclidean' else 'city_block'
217 
218             use_pca = st.sidebar.checkbox("Use PCA")
219 
220             labels, centroids, inertia_values, silhouette_avg = None,
    None, None, None  # Initialize variables
221 
222             if st.sidebar.button("Cluster"):
223                 kmeans = KMeansClustering(k=k, distance_metric=
    distance_metric, random_state=42)
224                 labels, centroids, inertia_values, silhouette_avg =
    kmeans.fit(X_scaled)
225 
226                 st.success(f"Convergence achieved in {len(inertia_values
    )} iterations.")
227                 st.write(f"Number of clusters: {k}")
228                 st.write(f"Number of data points: {len(X)}")
229                 st.write(f"Number of features: {X.shape[1]}")
230                 st.write(f"Silhouette Score: {silhouette_avg:.4f}")
231 
232 
233             plot_choice = st.sidebar.selectbox("Select Plot", ["
    Clustering Result", "Convergence Plot", "Silhouette Score"])
234 
235             if plot_choice == "Clustering Result" and labels is not None
    :
236                 fig = plot_result(X_scaled, labels, centroids)
237                 st.plotly_chart(fig, use_container_width=True)
238             elif plot_choice == "Convergence Plot" and inertia_values is
     not None:
239                 fig = plot_convergence(inertia_values)
240                 st.plotly_chart(fig, use_container_width=True)
241             elif plot_choice == "Silhouette Score" and silhouette_avg is
     not None:
242                 silhouette_values = []
243                 for k in range(2, 12):
244                     kmeans = KMeansClustering(k=k, random_state=42)
245                     _, _, _, silhouette_avg = kmeans.fit(X_scaled)
246                     silhouette_values.append(silhouette_avg)
```

```
                fig = plot_silhouette_score(silhouette_values)
                st.plotly_chart(fig, use_container_width=True)

    elif selected == "Hierarchical Clustering":
        st.title("Hierarchical Clustering")
        uploaded_file = st.file_uploader("Upload Excel File", type=["xlsx", "xls"])
        if uploaded_file is not None:
            data = pd.read_excel(uploaded_file, index_col=0)
            clustering_algorithm = HierarchicalClustering(k=3)  # Initialize the clustering algorithm
            clustering_algorithm.fit(data.values)  # Fit the data
            labels = clustering_algorithm.labels_  # Get the cluster labels
            st.write(labels)  # Display the cluster labels
            plot_dendrogram(data.values)  # Visualize the dendrogram

if __name__ == "__main__":
    st.set_option('deprecation.showPyplotGlobalUse', False)  # Disable the warning about st.pyplot() usage
    main()
```

Listing 11 – python script1

3.2 Description du programme :

Mon programme est une application de clustering avec une interface utilisateur conviviale, construite avec Streamlit. Voici une description de ses principales fonctionnalités :

1. **Chargement des données :** Les utilisateurs peuvent charger leurs données à partir de fichiers Excel en utilisant l'interface de téléchargement de fichiers.
2. **K-means Clustering :** L'application permet aux utilisateurs d'effectuer le clustering K-means sur leurs données chargées. Ils peuvent spécifier le nombre de clusters (K) et la distance métrique (euclidienne ou de bloc de ville). Une fois les paramètres sélectionnés, les utilisateurs peuvent lancer le clustering et visualiser les résultats.
3. **Visualisation des résultats :** Les utilisateurs peuvent choisir parmi plusieurs types de visualisations pour explorer les résultats du clustering. Les options incluent un graphique de dispersion des données avec les clusters colorés et les centroids, un graphique de convergence montrant l'évolution de l'inertie au fil des itérations, et un graphique de score de silhouette pour évaluer la qualité du clustering.
4. **Interprétation des résultats :** L'application fournit des informations sur le nombre de clusters, la convergence du clustering, et le score de silhouette pour évaluer la qualité des clusters obtenus.

En utilisant cette application, les utilisateurs peuvent facilement explorer leurs données, identifier des structures de clustering, et prendre des décisions basées sur les insights générés par les méthodes de clustering K-means et CAH.

3.2.1 Packages utilisés :

Les packages utilisés dans le code Python pour l'application de clustering sont les suivants :

— `streamlit` : Streamlit est un framework pour créer des applications web interactives en Python. L'importation de `streamlit` vous permet d'utiliser les fonctionnalités de Streamlit pour développer des applications web conviviales et interactives avec facilité.
— `pandas` : Bibliothèque pour la manipulation et l'analyse de données tabulaires.
— `numpy` : Bibliothèque pour le calcul numérique avancé.
— `from sklearn.decomposition import PCA` : Ce package fournit des fonctionnalités pour effectuer une Analyse en Composantes Principales (PCA), une technique utilisée pour réduire la dimensionnalité des données.
— `from sklearn.preprocessing import StandardScaler` : Ce package offre des outils pour la mise à l'échelle des caractéristiques, ce qui est souvent nécessaire avant d'appliquer des méthodes de clustering ou d'apprentissage automatique.
— `from sklearn.metrics import silhouette-score` : Ce package fournit une fonction pour calculer le score de silhouette, une métrique utilisée pour évaluer la qualité des clusters obtenus à partir de méthodes de clustering.
— `plotly.graph-objs as go` : Plotly est une bibliothèque de visualisation interactive qui offre des fonctionnalités pour créer des graphiques interactifs. L'importation de `plotly.graph-objs as go` vous permet d'accéder à la classe `go`, qui contient des objets graphiques utilisés pour créer des visualisations à l'aide de Plotly
— `plotly.express as px` : Plotly Express est une interface haut niveau de Plotly qui permet de créer facilement des graphiques expressifs et des tableaux de bord interactifs. L'importation de `plotly.express as px` vous permet d'utiliser cette interface pour créer des visualisations de données de manière plus concise et intuitive.
— `matplotlib.pyplot as plt` : Matplotlib est une bibliothèque de traçage graphique en Python qui offre une grande variété de fonctionnalités pour créer des graphiques statiques. L'importation de `matplotlib.pyplot as plt` vous permet d'utiliser les fonctions de traçage de Matplotlib pour créer des graphiques dans votre application
— `from scipy.cluster import hierarchy` : SciPy est une bibliothèque Python pour les calculs scientifiques qui inclut des modules pour l'optimisation, l'algèbre linéaire, l'intégration, l'interpolation, la transformation de Fourier, le traitement du signal et bien d'autres encore. L'importation de `from scipy.cluster import hierarchy` vous permet d'accéder aux fonctionnalités de clustering hiérarchique disponibles dans le module `hierarchy` de SciPy.
— `base64` : Bibliothèque pour l'encodage et le décodage de données en base64.
— `requests` : Bibliothèque pour l'envoi de requêtes HTTP en Python.

Ces packages sont utilisés pour différentes fonctionnalités de l'application de clustering, telles que le chargement des données, la manipulation et l'analyse des données, la création de visualisations, le clustering et l'évaluation des modèles.

3.3 Interface et utilisation du programme :

Sélection de l'option : Lorsque vous ouvrez l'application, vous êtes invité à sélectionner une option parmi celles disponibles, telles que "KMeans Clustering" ou "Hierarchical Clustering".

3 Interface graphique(kmeans, CAH) :

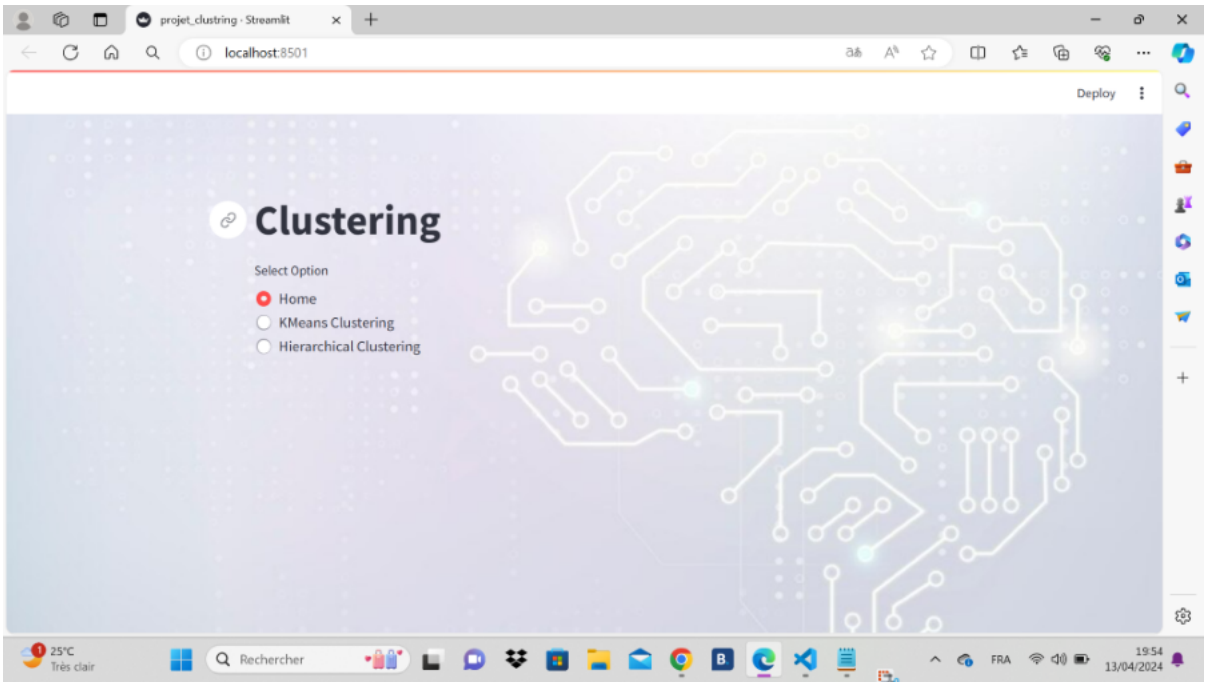

FIGURE 1 – étape 1

Chargement des données : Si vous choisissez l'option "KMeans Clustering", vous pouvez charger vos données en utilisant le bouton "Upload Excel file".

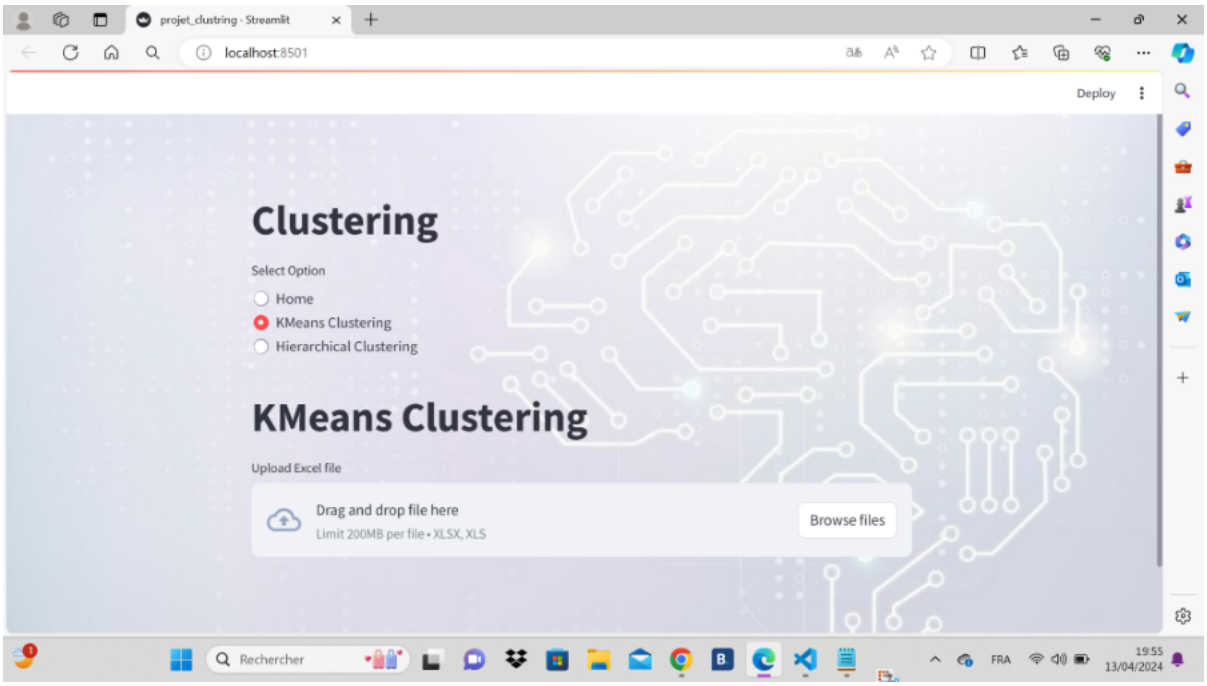

FIGURE 2 – étape 2

Vous pouvez télécharger un fichier Excel contenant vos données pour l'analyse

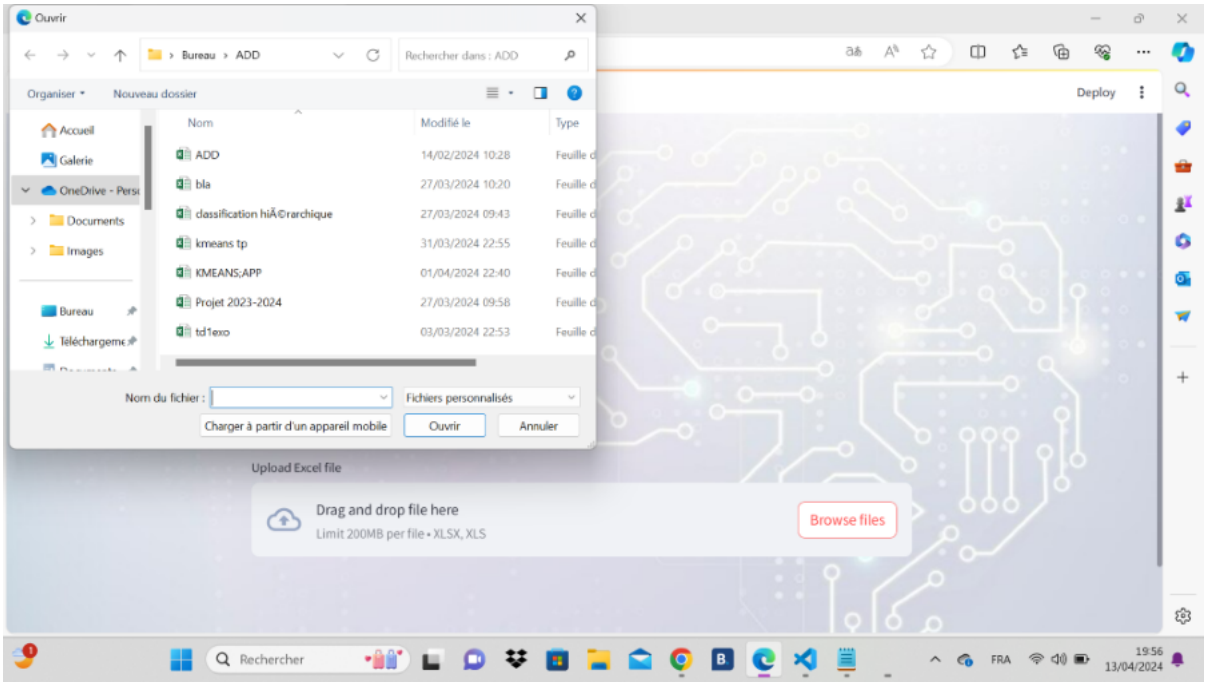

FIGURE 3 – étape 3

Une fois les données chargées, vous pouvez effectuer : Choix de la méthode de prétraitement : Vous avez la possibilité de spécifier la méthode de prétraitement des données. Par exemple, vous pouvez choisir d'appliquer une Analyse en Composantes Principales (ACP) pour réduire la dimensionnalité des données avant d'appliquer les méthodes de clustering.

Spécification des distances : Pour l'option "KMeans Clustering", vous pouvez spécifier la distance métrique à utiliser pour mesurer la similarité entre les points de données. Les distances les plus couramment utilisées sont la distance euclidienne et la distance de Manhattan.

Choix du nombre de clusters (k) : Vous pouvez spécifier le nombre de clusters (k) que vous souhaitez obtenir à partir de l'ensemble de données. Ce choix peut être basé sur des connaissances préalables du domaine, des analyses exploratoires des données ou des méthodes d'évaluation automatique du nombre optimal de clusters.

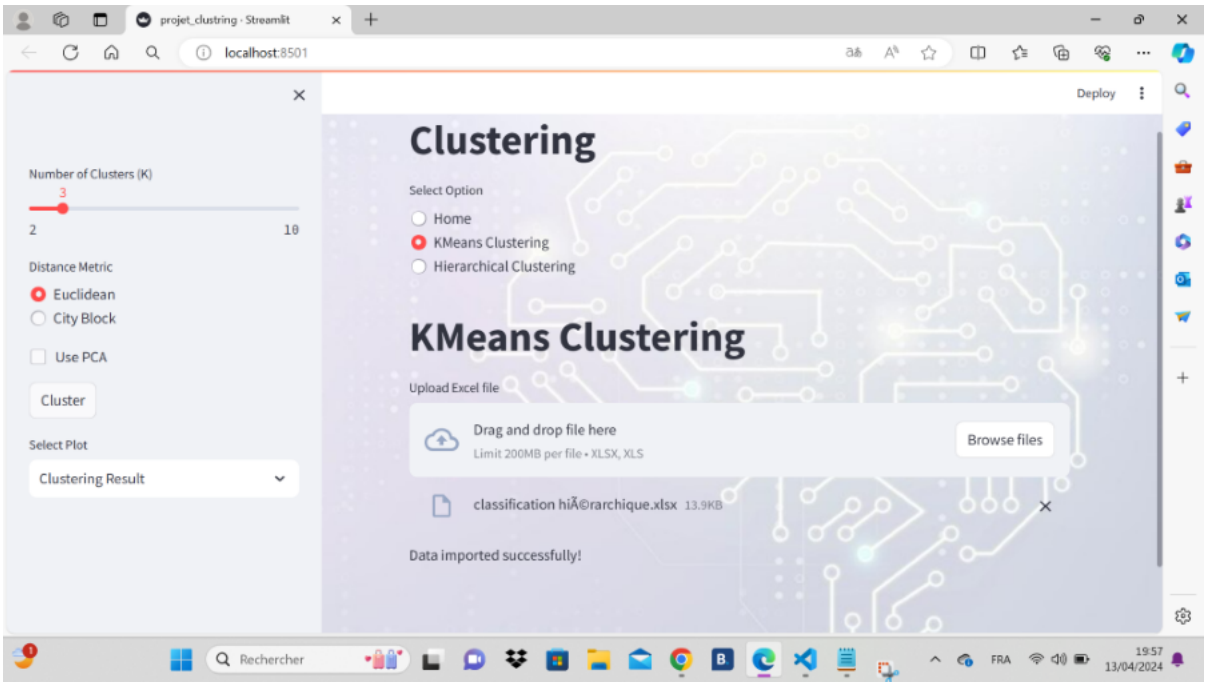

FIGURE 4 – étape 4

Sélection du type de plot : Une fois que le clustering est terminé et que les résultats sont prêts à être visualisés, vous pouvez choisir le type de plot que vous souhaitez afficher.

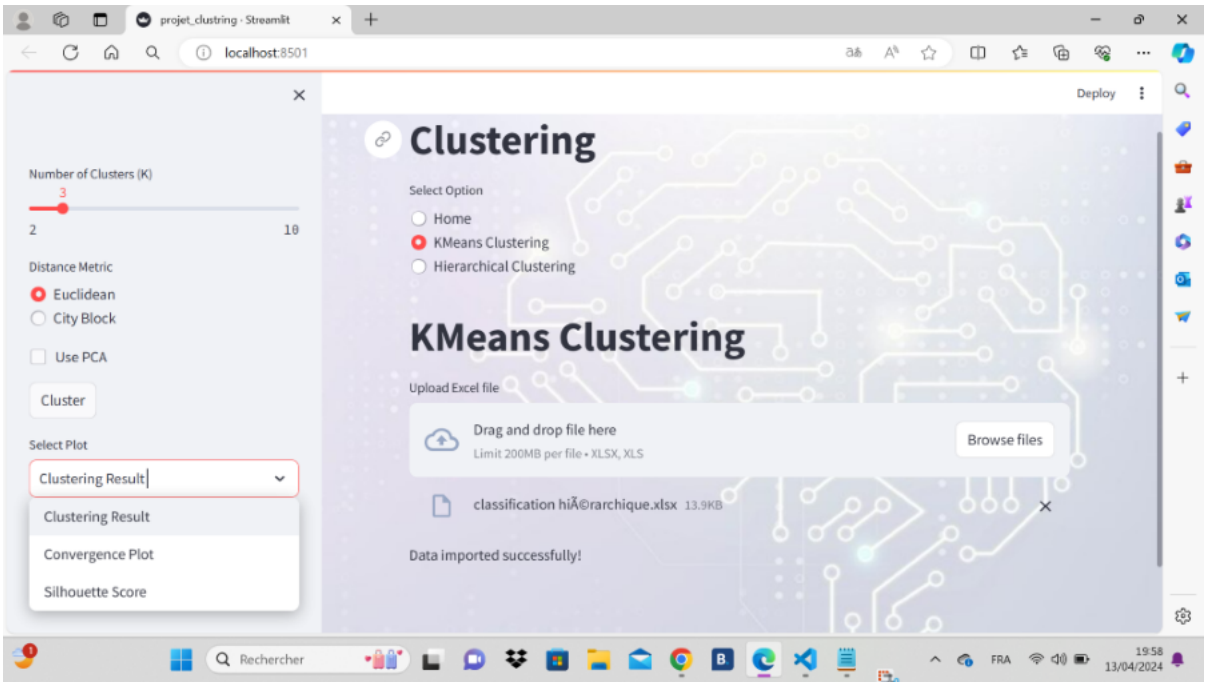

FIGURE 5 – étape 5

Si vous choisissez l'option "Clustering Result", vous obtiendrez un plot qui représente les résultats du clustering.

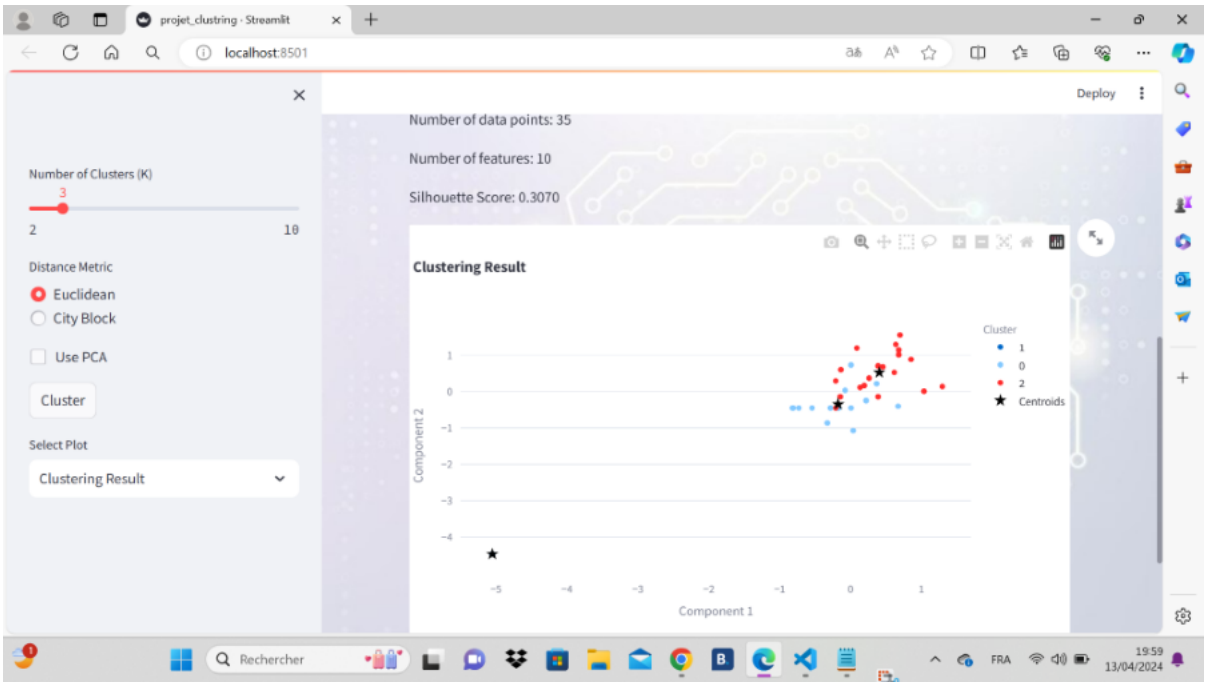

FIGURE 6 – étape 6

Si vous choisissez l'option "Convergence Plot", vous obtiendrez un graphique qui représente la convergence de clustering, dans lequelle l'inertie évolue au fil des itérations.

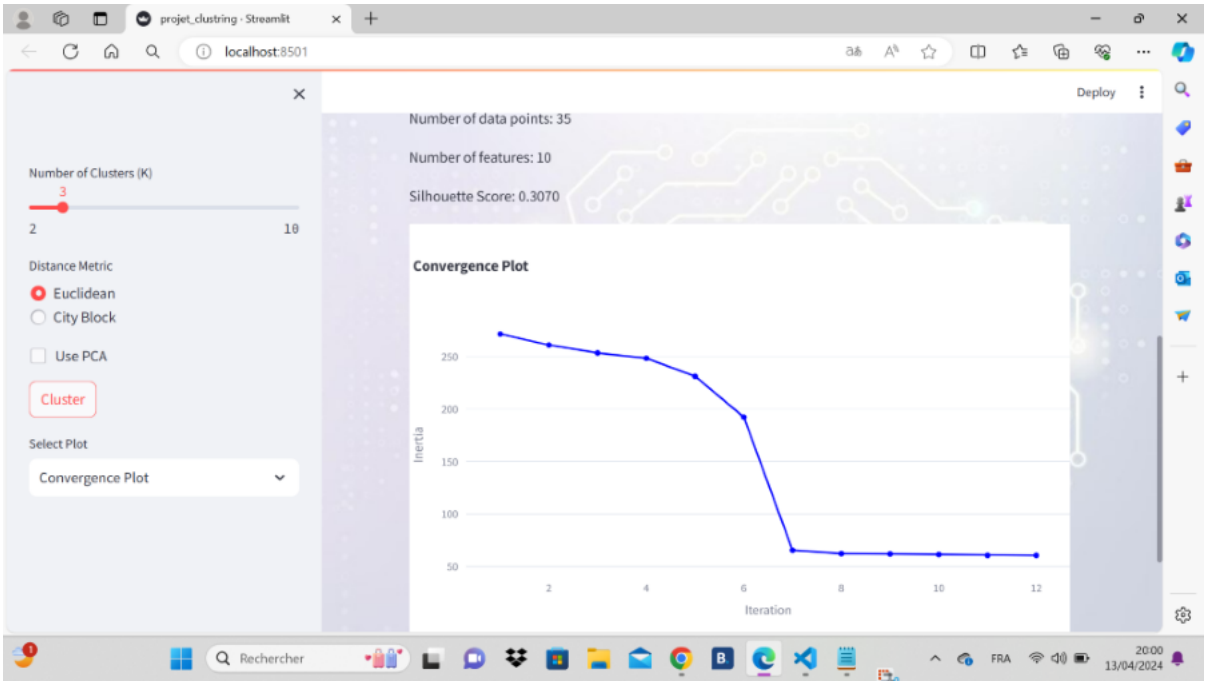

FIGURE 7 – étape 7

Si vous choisissez l'option "Silhouette Score", vous obtiendrez un graphique qui représente l'évaluation de la qualité du clustering à l'aide du score de silhouette.

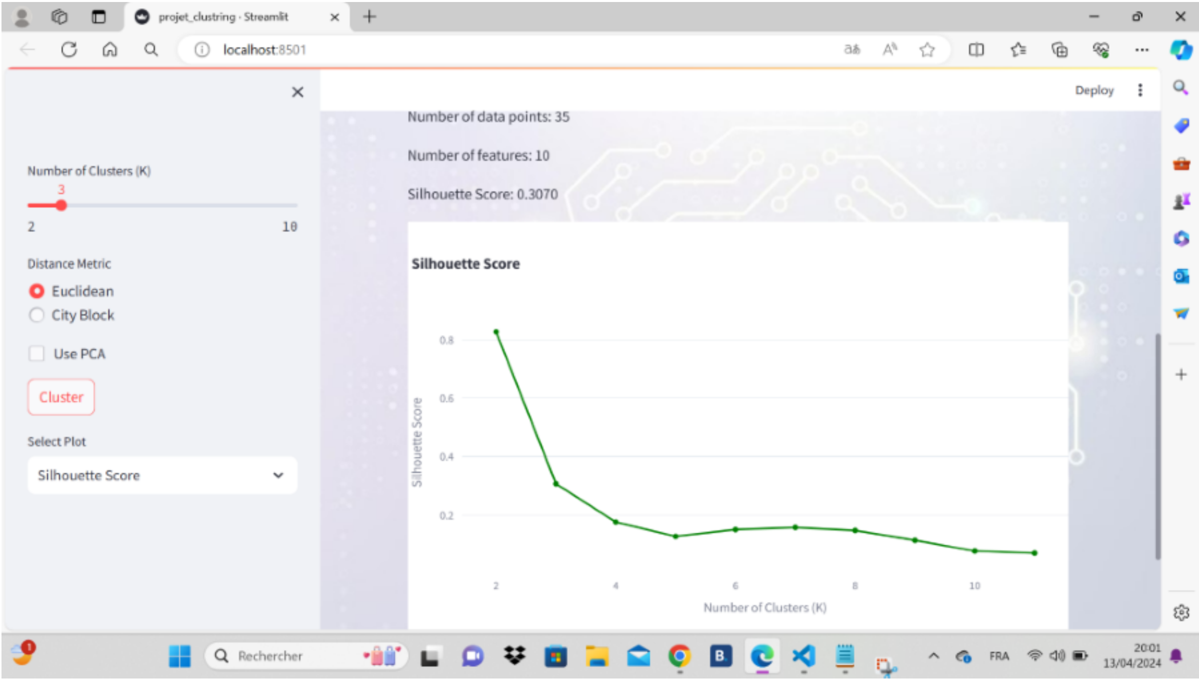

FIGURE 8 – étape 8

Si vous choisissez l'option "Hierarchical Clustring", vous pouvez charger vos données en utilisant le bouton "Upload Excel file"

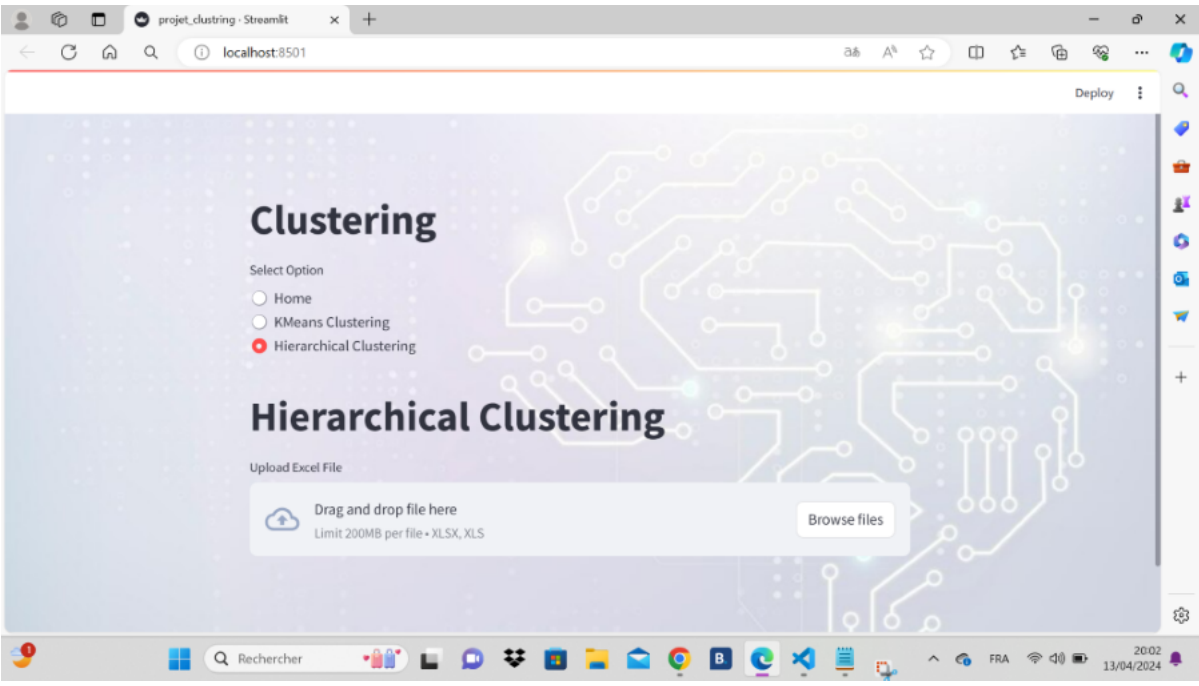

FIGURE 9 – étape 9

Après l'exécution du clustering, vous pouvez visualiser le dendrogramme résultant pour explorer la structure hiérarchique des clusters. Vous pouvez couper le dendrogramme à différentes hauteurs pour obtenir différents niveaux de granularité dans les clusters.

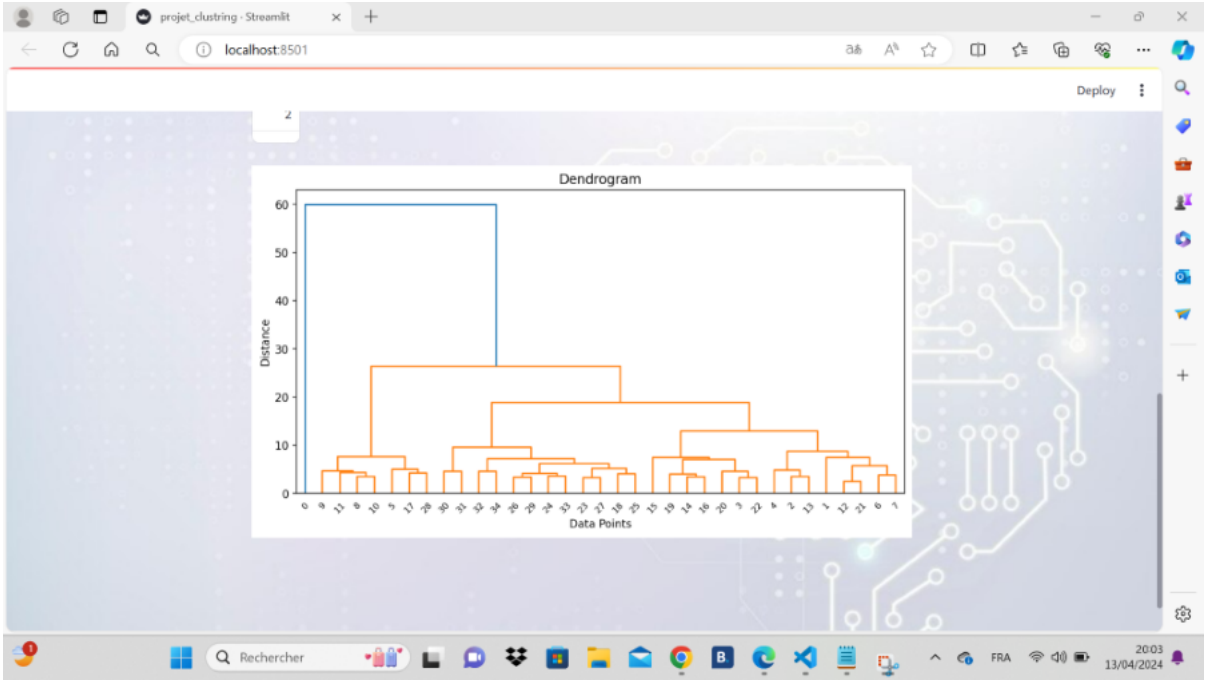

FIGURE 10 – étape 10

4 Conclusion :

Dans ce rapport, nous avons exploré les concepts de base de l'apprentissage non supervisé, en mettant l'accent sur la classification hiérarchique agglomérative (CAH) et l'algorithme K-Means. Nous avons abordé le prétraitement des données, les étapes de ces algorithmes, les mesures d'évaluation , ainsi que la visualisation des résultats à l'aide de dendrogrammes, ou les plots tel que convergence plot, clustring result. Cette exploration nous a permis de mieux comprendre l'application pratique de ces techniques dans l'analyse de données. Voici le lien vers mon référentiel GitHub où vous trouverez mon projet : **GitHub :**`https://github.com/sasoosaa`

5 Annexe :

Code de la methode kmeans :

```
1
2  class KMeansClustering:
3      def __init__(self, k=3, distance_metric='euclidean', random_state=
       None):
4          self.k = k
5          self.distance_metric = distance_metric
6          self.centroids = None
7          self.random_state = random_state
8
9      def calculate_distance(self, data_point, centroids):
10         if self.distance_metric == 'euclidean':
11             return np.sqrt(np.sum((centroids - data_point) ** 2, axis=1)
       )
12         elif self.distance_metric == 'city_block':
13             return np.sum(np.abs(centroids - data_point), axis=1)
14         else:
15             raise ValueError("Invalid distance metric. Supported metrics
        are 'euclidean' and 'city_block'.")
16
17     def fit(self, X, max_iterations=200):
18         np.random.seed(self.random_state)  # Set the random seed
19
20         # Initialize centroids randomly
21         self.centroids = X[np.random.choice(X.shape[0], self.k, replace=
       False)]
22
23         inertia_values = []
24
25         for _ in range(max_iterations):
26             # Assign each data point to the nearest centroid
27             distances = np.vstack([self.calculate_distance(data_point,
       self.centroids) for data_point in X])
28             cluster_assignment = np.argmin(distances, axis=1)
29
30             # Update centroids
31             new_centroids = np.array([X[cluster_assignment == i].mean(
       axis=0) for i in range(self.k)])
32
33             # Calculate inertia
34             inertia = np.sum((X - new_centroids[cluster_assignment]) **
       2)
35             inertia_values.append(inertia)
36
37             # Check convergence
38             if np.allclose(self.centroids, new_centroids):
39                 break
40
41             self.centroids = new_centroids
42
43         # Calculate silhouette score
44         labels = np.argmin(distances, axis=1)
45         silhouette_avg = silhouette_score(X, labels)
46
```

```
        return cluster_assignment, self.centroids, inertia_values,
    silhouette_avg

def run_clustering(X, k, distance_metric, use_pca):
    if use_pca:
        pca = PCA(n_components=2)
        X = pca.fit_transform(X)

    kmeans = KMeansClustering(k=k, distance_metric=distance_metric,
    random_state=42)
    labels, centroids, inertia_values, silhouette_avg = kmeans.fit(X)
    return labels, centroids, inertia_values, silhouette_avg

def plot_result(X_reduced, labels, centroids):
    if X_reduced.shape[1] == 2:
        df = pd.DataFrame(X_reduced, columns=['Component 1', 'Component
    2'])
    else:
        df = pd.DataFrame(X_reduced[:, :2], columns=['Component 1', '
    Component 2'])
    df['Cluster'] = labels.astype(str)  # Convert labels to string for
    coloring purposes

    fig = px.scatter(df, x='Component 1', y='Component 2', color='
    Cluster',
                     title='Clustering Result', hover_name=df.index)
    fig.add_trace(go.Scatter(
        x=centroids[:, 0],
        y=centroids[:, 1],
        mode='markers',
        marker=dict(size=10, color='black', symbol='star'),
        name='Centroids'
    ))
    return fig

def plot_convergence(inertia_values):
    fig = go.Figure()
    fig.add_trace(go.Scatter(
        x=list(range(1, len(inertia_values) + 1)),
        y=inertia_values,
        mode='lines+markers',
        marker=dict(color='blue'),
        name='Inertia'
    ))
    fig.update_layout(
        title='Convergence Plot',
        xaxis=dict(title='Iteration'),
        yaxis=dict(title='Inertia'),
        hovermode='closest'
    )
    return fig

def plot_silhouette_score(silhouette_values):
    fig = go.Figure()
    fig.add_trace(go.Scatter(
        x=list(range(2, 12)),
        y=silhouette_values,
```

```
        mode='lines+markers',
        marker=dict(color='green'),
        name='Silhouette Score'
    ))
    fig.update_layout(
        title='Silhouette Score',
        xaxis=dict(title='Number of Clusters (K)'),
        yaxis=dict(title='Silhouette Score'),
        hovermode='closest'
    )
    return fig

def main():
    st.title("KMeans Clustering")

    uploaded_file = st.file_uploader("Upload Excel file", type=["xlsx", "xls"])
    if uploaded_file is not None:
        data = pd.read_excel(uploaded_file, index_col=0)
        st.write("Data imported successfully!")

        X = data.values
        X = X[:, 2:]  # Adjust this according to your data

        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X)

        k = st.sidebar.slider("Number of Clusters (K)", min_value=2, max_value=10, value=3)
        distance_metric = st.sidebar.radio("Distance Metric", ["Euclidean", "City Block"], index=0)
        distance_metric = 'euclidean' if distance_metric == 'Euclidean' else 'city_block'

        use_pca = st.sidebar.checkbox("Use PCA")

        labels, centroids, inertia_values, silhouette_avg = None, None, None, None  # Initialize variables

        if st.sidebar.button("Cluster"):
            labels, centroids, inertia_values, silhouette_avg = run_clustering(X_scaled, k, distance_metric, use_pca)

            st.success(f"Convergence achieved in {len(inertia_values)} iterations.")
            st.write(f"Number of clusters: {k}")
            st.write(f"Number of data points: {len(X)}")
            st.write(f"Number of features: {X.shape[1]}")
            st.write(f"Silhouette Score: {silhouette_avg:.4f}")

        plot_choice = st.sidebar.selectbox("Select Plot", ["Clustering Result", "Convergence Plot", "Silhouette Score", "Elbow Method"])

        if plot_choice == "Clustering Result" and labels is not None:
            fig = plot_result(X_scaled, labels, centroids)
            st.plotly_chart(fig, use_container_width=True)
```

```
        elif plot_choice == "Convergence Plot" and inertia_values is not None:
            fig = plot_convergence(inertia_values)
            st.plotly_chart(fig, use_container_width=True)
        elif plot_choice == "Silhouette Score" and silhouette_avg is not None:
            silhouette_values = []
            for k in range(2, 12):
                kmeans = KMeansClustering(k=k, random_state=42)
                _, _, _, silhouette_avg = kmeans.fit(X_scaled)
                silhouette_values.append(silhouette_avg)
            fig = plot_silhouette_score(silhouette_values)
            st.plotly_chart(fig, use_container_width=True)

if __name__ == "__main__":
    main()
```

Listing 12 – python script1

Code de la méthode CAH :

```

class HierarchicalClustering:
    def __init__(self, k):
        self.k = k

    def fit(self, data, distances):
        n = len(data)
        clusters = [[i] for i in range(n)]

        while len(clusters) > self.k:
            min_dist = np.inf
            for i in range(len(clusters)):
                for j in range(i + 1, len(clusters)):
                    cluster1 = clusters[i]
                    cluster2 = clusters[j]
                    avg_dist = 0
                    count = 0
                    for idx1 in cluster1:
                        for idx2 in cluster2:
                            if not np.isnan(distances[idx1, idx2]):
                                avg_dist += distances[idx1, idx2]
                                count += 1
                    if count > 0:
                        avg_dist /= count
                        if avg_dist < min_dist:
                            min_dist = avg_dist
                            merge_index = (i, j)
            i, j = merge_index
            new_cluster = clusters[i] + clusters[j]
            clusters[i] = new_cluster
            clusters.pop(j)

        self.labels_ = np.zeros(n)
        for i, cluster in enumerate(clusters):
            for idx in cluster:
                self.labels_[idx] = i

```

```python
def import_data_from_excel(file_path):
    data = pd.read_excel(file_path)
    # Check if the first row or first column contains non-numeric values
    first_row_is_str = any(isinstance(val, str) for val in data.iloc[0])
    first_column_is_str = isinstance(data.iloc[:, 0].values[0], str)
    if first_row_is_str or first_column_is_str:
        data = data.iloc[1:, 1:]  # Exclude first row and first column if they contain non-numeric values
    return data

def dendrogram_visualization(data, labels):
    X = data.values
    Z = linkage(X, method='ward')
    fig, ax = plt.subplots(figsize=(10, 5))
    dn = dendrogram(Z, ax=ax)
    ax.set_title("Dendrogram")
    ax.set_xlabel("Data Points")
    ax.set_ylabel("Distance")
    st.pyplot(fig)  # Display the plot in Streamlit

# Example usage:
uploaded_file = st.file_uploader("Upload Excel File", type=["xlsx", "xls"])
if uploaded_file is not None:
    data = import_data_from_excel(uploaded_file)
    k = 3  # Number of clusters
    distances = pairwise_distances(data.values, metric='euclidean')
    clustering_algorithm = HierarchicalClustering(k=k)
    clustering_algorithm.fit(data, distances)
    labels = clustering_algorithm.labels_
    dendrogram_visualization(data, labels)
```

Listing 13 – python script1