



ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES AGADIR

RAPPORT D'ANALYSE DES DONNÉES

Documentation

Élève :

Salma ELAISSARI

Enseignant :

Prof.BrahimEL ASRI

Filière:

FID

16 avril 2024

1	Introduction	2
1.1	Importance de la classification	2
1.2	Introduction aux méthodes de classification hiérarchique et de k-means	2
1.3	Objectifs du rapport	3
1.4	Description du projet	3
2	Explication des méthodes :	4
2.1	Explication détaillée de la CAH :	4
2.1.1	Principe de fonctionnement	4
2.1.2	Avantages de la CAH :	6
2.1.3	Inconvénients de la CAH :	6
2.1.4	Principes et Fonctionnement du code de CAH sur Python :	7
2.1.5	Packages utilisés :	7
2.1.6	Classes utilisés :	8
2.1.7	Fonctions utilisés :	9
2.2	Explication détaillée de la méthode kmeans :	11
2.2.1	Principe de fonctionnement :	11
2.2.2	Avantages de K-means :	11
2.2.3	Inconvénients de K-means :	12
2.2.4	Principes et Fonctionnement du code de kmeans sur Python :	13
2.2.5	Packages utilisés :	13
2.2.6	Classes utilisés :	14
2.2.7	Fonctions utilisés :	15
3	Interface graphique(kmeans, CAH :	17
3.1	Code total sur python :	17
3.2	Description du programme :	22
3.2.1	Packages utilisés :	22
3.3	Interface et utilisation du programme :	23
4	Conclusion :	29
5	Annexe :	30

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.

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.

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.

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.

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.

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.

2.1.6 Classes utilisés :

8

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.

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 .

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.

```

1 def import_data_from_excel(file_path):
2     data = pd.read_excel(file_path)
3     # Check if the first row or first column contains non-numeric
4     # values
5     first_row_is_str = any(isinstance(val, str) for val in data.
6                             iloc[0])
7     first_column_is_str = isinstance(data.iloc[:, 0].values[0], str
8                                     )
9     if first_row_is_str or first_column_is_str:
10         data = data.iloc[1:, 1:] # Exclude first row and first
11                                   # column if they contain non-numeric values
12     return data

```

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

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

```

1 def dendrogram_visualization(data, labels):
2     X = data.values
3     Z = linkage(X, method='ward')
4     fig, ax = plt.subplots(figsize=(10, 5))
5     dn = dendrogram(Z, ax=ax)
6     ax.set_title("Dendrogram")
7     ax.set_xlabel("Data Points")
8     ax.set_ylabel("Distance")
9     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 :

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

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

- Assignment 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 :

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.decomposition import PCA
4 from sklearn.preprocessing import StandardScaler
5 from sklearn.metrics import silhouette_score
6 import plotly.graph_objs as go
7 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.

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 :

```

1 def run_clustering(X, k, distance_metric, use_pca):
2     if use_pca:
3         pca = PCA(n_components=2)
4         X = pca.fit_transform(X)
5
6     kmeans = KMeansClustering(k=k, distance_metric=distance_metric,
7                               random_state=42)
8     labels, centroids, inertia_values, silhouette_avg = kmeans.fit(X)
9     return labels, centroids, inertia_values, silhouette_avg
10
11 def plot_result(X_reduced, labels, centroids):
12     if X_reduced.shape[1] == 2:
13         df = pd.DataFrame(X_reduced, columns=['Component 1', 'Component
14         2'])
15     else:
16         df = pd.DataFrame(X_reduced[:, :2], columns=['Component 1', '
17         Component 2'])
18     df['Cluster'] = labels.astype(str) # Convert labels to string for
19     coloring purposes
20
21     fig = px.scatter(df, x='Component 1', y='Component 2', color='
22     Cluster',
23                     title='Clustering Result', hover_name=df.index)
24     fig.add_trace(go.Scatter(
25         x=centroids[:, 0],
26         y=centroids[:, 1],
27         mode='markers',
28         marker=dict(size=10, color='black', symbol='star'),
29         name='Centroids'
30     ))
31     return fig
32
33 def plot_convergence(inertia_values):
34     fig = go.Figure()
35     fig.add_trace(go.Scatter(

```

```

31         x=list(range(1, len(inertia_values) + 1)),
32         y=inertia_values,
33         mode='lines+markers',
34         marker=dict(color='blue'),
35         name='Inertia'
36     ))
37     fig.update_layout(
38         title='Convergence Plot',
39         xaxis=dict(title='Iteration'),
40         yaxis=dict(title='Inertia'),
41         hovermode='closest'
42     )
43     return fig
44
45 def plot_silhouette_score(silhouette_values):
46     fig = go.Figure()
47     fig.add_trace(go.Scatter(
48         x=list(range(2, 12)),
49         y=silhouette_values,
50         mode='lines+markers',
51         marker=dict(color='green'),
52         name='Silhouette Score'
53     ))
54     fig.update_layout(
55         title='Silhouette Score',
56         xaxis=dict(title='Number of Clusters (K)'),
57         yaxis=dict(title='Silhouette Score'),
58         hovermode='closest'
59     )
60     return fig

```

Listing 7 – 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.

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.

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.

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 :

```
1 import streamlit as st
2 import pandas as pd
3 import numpy as np
4 from sklearn.decomposition import PCA
5 from sklearn.preprocessing import StandardScaler
6 from sklearn.metrics import silhouette_score
7 import plotly.graph_objs as go
8 import plotly.express as px
9 import matplotlib.pyplot as plt
10 from scipy.cluster.hierarchy import linkage, dendrogram # Modified
11     import statement
12 import base64
13 import requests
14
15 # Set background image
16 background_image_url = "https://img.freepik.com/free-vector/ai-
17     technology-brain-background-vector-digital-transformation-
18     concept_53876-117820.jpg?w=900&t=st=1713012632~exp=1713013232~hmac=05
19     da65e2d9e6da77202ded9006cfecb86c0c11fbc70346fb0532307b18d8ac3f"
20
21 def get_base64_of_bin_file(url):
22     data = requests.get(url).content
23     return base64.b64encode(data).decode()
24
25 def set_png_as_page_bg(png_file):
26     bin_str = get_base64_of_bin_file(png_file)
27     st.markdown(
28         f'<style>
29         f'.stApp {{
30         f'background-image: url("data:image/png;base64,{bin_str}");
31         f'background-size: cover;
32         f'}}
33         f'</style>',
34         unsafe_allow_html=True
35     )
36
37 set_png_as_page_bg(background_image_url)
38
39 class KMeansClustering:
40     def __init__(self, k=3, distance_metric='euclidean', random_state=
41         None):
42         self.k = k
43         self.distance_metric = distance_metric
44         self.centroids = None
45         self.random_state = random_state
46
47     def calculate_distance(self, data_point, centroids):
48         if self.distance_metric == 'euclidean':
49             return np.sqrt(np.sum((centroids - data_point) ** 2, axis=1))
50
51         elif self.distance_metric == 'city_block':
52             return np.sum(np.abs(centroids - data_point), axis=1)
```

```

47:         else:
48:             raise ValueError("Invalid distance metric. Supported metrics
are 'euclidean' and 'city_block'.")
49:
50:     def fit(self, X, max_iterations=200):
51:         np.random.seed(self.random_state) # Set the random seed
52:
53:         # Initialize centroids randomly
54:         self.centroids = X[np.random.choice(X.shape[0], self.k, replace=
False)]
55:
56:         inertia_values = []
57:
58:         for _ in range(max_iterations):
59:             # Assign each data point to the nearest centroid
60:             distances = np.vstack([self.calculate_distance(data_point,
self.centroids) for data_point in X])
61:             cluster_assignment = np.argmin(distances, axis=1)
62:
63:             # Update centroids
64:             new_centroids = np.array([X[cluster_assignment == i].mean(
axis=0) for i in range(self.k)])
65:
66:             # Calculate inertia
67:             inertia = np.sum((X - new_centroids[cluster_assignment])**
2)
68:             inertia_values.append(inertia)
69:
70:             # Check convergence
71:             if np.allclose(self.centroids, new_centroids):
72:                 break
73:
74:             self.centroids = new_centroids
75:
76:             # Calculate silhouette score
77:             labels = np.argmin(distances, axis=1)
78:             silhouette_avg = silhouette_score(X, labels)
79:
80:             return cluster_assignment, self.centroids, inertia_values,
silhouette_avg
81:
82: class HierarchicalClustering:
83:     def __init__(self, k):
84:         self.k = k
85:
86:     def fit(self, data):
87:         self.data = data
88:         n = len(data)
89:         distances = np.zeros((n, n))
90:
91:         # Calculate pairwise distances
92:         for i in range(n):
93:             for j in range(n):
94:                 distances[i, j] = self.euclidean_distance(data[i], data[
j])
95:
96:         # 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
130        x2 = np.array(x2, dtype=np.float64) # Convert x2 to numpy array
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.
137        hierarchy directly
138        plt.figure(figsize=(10, 5))
139        dn = dendrogram(Z)
140        plt.title("Dendrogram")
141        plt.xlabel("Data Points")
142        plt.ylabel("Distance")
143        st.pyplot(plt.gcf()) # Pass the current figure as an argument to st
144        .pyplot()
145
146    def plot_result(X_reduced, labels, centroids):
147        if X_reduced.shape[1] == 2:
148            df = pd.DataFrame(X_reduced, columns=['Component 1', 'Component
149            2'])
150        else:
151            df = pd.DataFrame(X_reduced[:, :2], columns=['Component 1', '

```

```

149         df['Cluster'] = labels.astype(str) # Convert labels to string for
150         coloring purposes
151
152         fig = px.scatter(df, x='Component 1', y='Component 2', color='
153         Cluster',
154                             title='Clustering Result', hover_name=df.index)
155     fig.add_trace(go.Scatter(
156         x=centroids[:, 0],
157         y=centroids[:, 1],
158         mode='markers',
159         marker=dict(size=10, color='black', symbol='star'),
160         name='Centroids'
161     ))
162     return fig
163
164 def plot_convergence(inertia_values):
165     fig = go.Figure()
166     fig.add_trace(go.Scatter(
167         x=list(range(1, len(inertia_values) + 1)),
168         y=inertia_values,
169         mode='lines+markers',
170         marker=dict(color='blue'),
171         name='Inertia'
172     ))
173     fig.update_layout(
174         title='Convergence Plot',
175         xaxis=dict(title='Iteration'),
176         yaxis=dict(title='Inertia'),
177         hovermode='closest'
178     )
179     return fig
180
181 def plot_silhouette_score(silhouette_values):
182     fig = go.Figure()
183     fig.add_trace(go.Scatter(
184         x=list(range(2, 12)),
185         y=silhouette_values,
186         mode='lines+markers',
187         marker=dict(color='green'),
188         name='Silhouette Score'
189     ))
190     fig.update_layout(
191         title='Silhouette Score',
192         xaxis=dict(title='Number of Clusters (K)'),
193         yaxis=dict(title='Silhouette Score'),
194         hovermode='closest'
195     )
196     return fig
197
198 def main():
199     st.title("Clustering")
200
201     selected = st.radio("Select Option", ["Home", "KMeans Clustering", "
202     Hierarchical Clustering"])
203
204     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)

```



```

247         fig = plot_silhouette_score(silhouette_values)
248         st.plotly_chart(fig, use_container_width=True)
249
250     elif selected == "Hierarchical Clustering":
251         st.title("Hierarchical Clustering")
252         uploaded_file = st.file_uploader("Upload Excel File", type=["
xls", "xls"])
253         if uploaded_file is not None:
254             data = pd.read_excel(uploaded_file, index_col=0)
255             clustering_algorithm = HierarchicalClustering(k=3)  #
Initialize the clustering algorithm
256             clustering_algorithm.fit(data.values)  # Fit the data
257             labels = clustering_algorithm.labels_  # Get the cluster
labels
258             st.write(labels)  # Display the cluster labels
259             plot_dendrogram(data.values)  # Visualize the dendrogram
260
261 if __name__ == "__main__":
262     st.set_option('deprecation.showPyplotGlobalUse', False)  # Disable
the warning about st.pyplot() usage
263     main()

```

Listing 8 – 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".

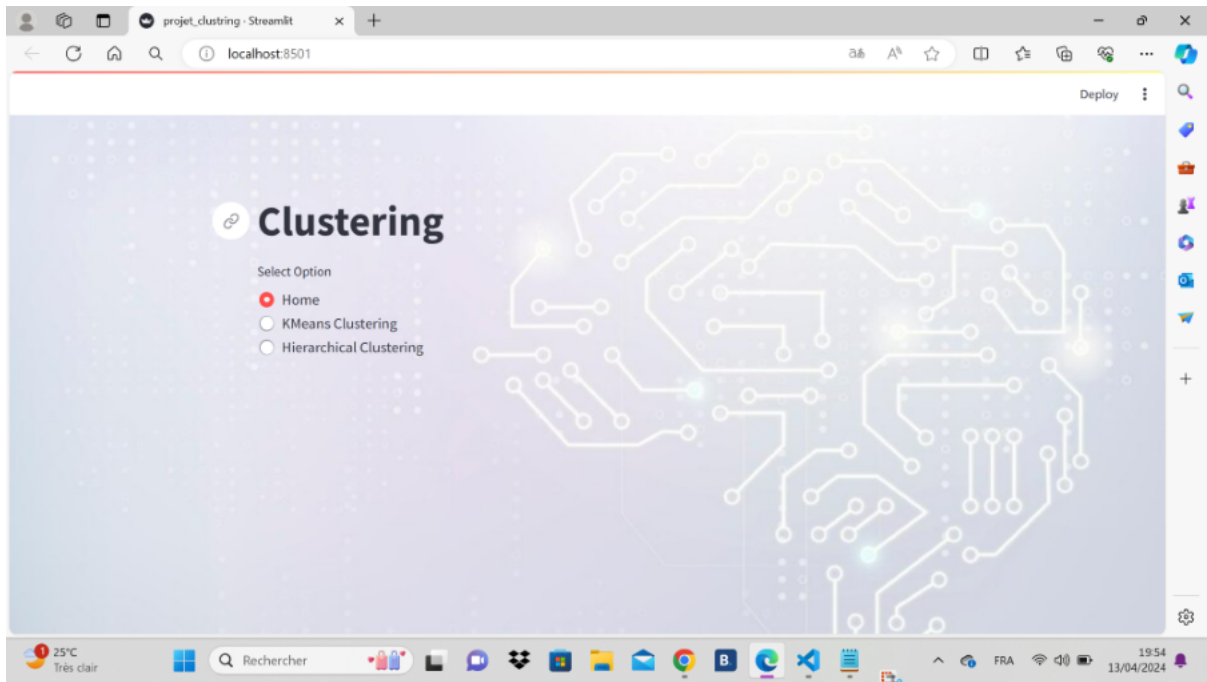


FIGURE 1 – étape 1

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

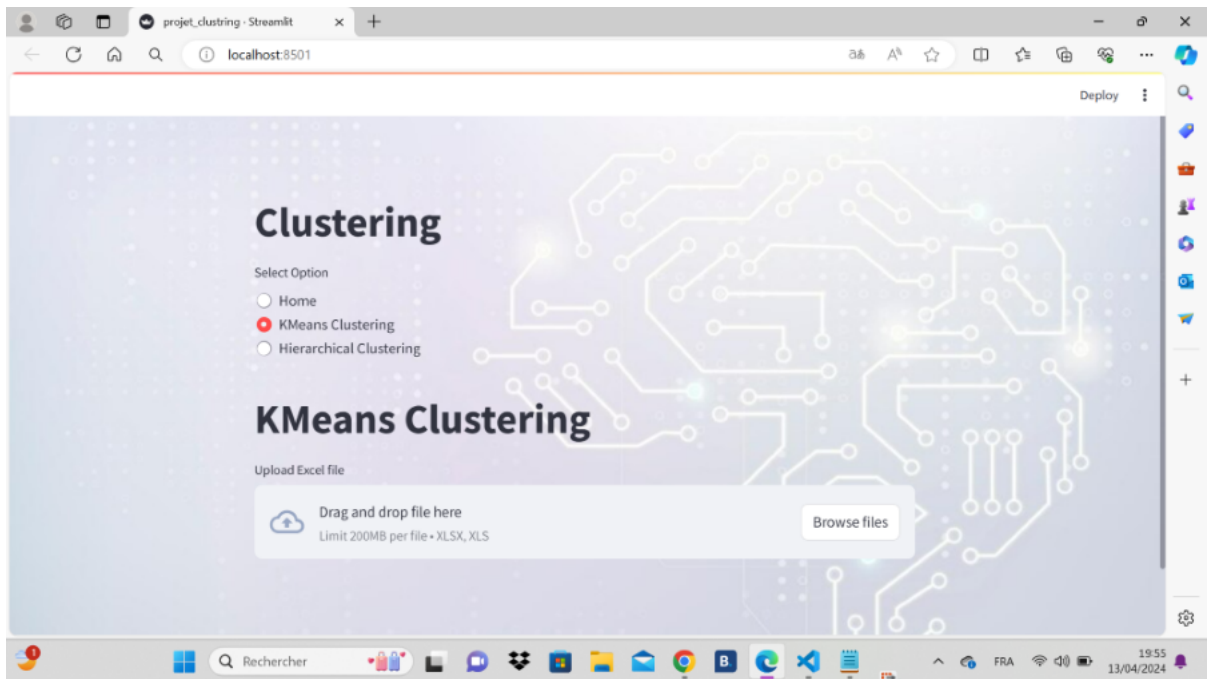


FIGURE 2 – étape 2

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

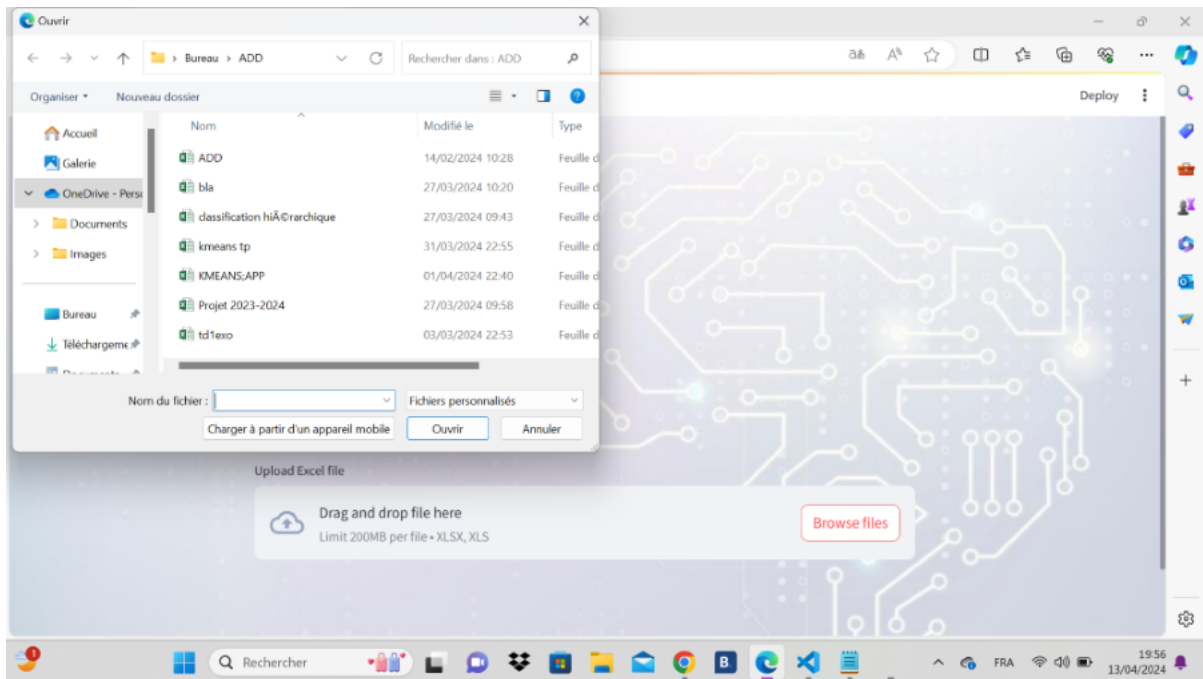


FIGURE 3 – étape 3

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.

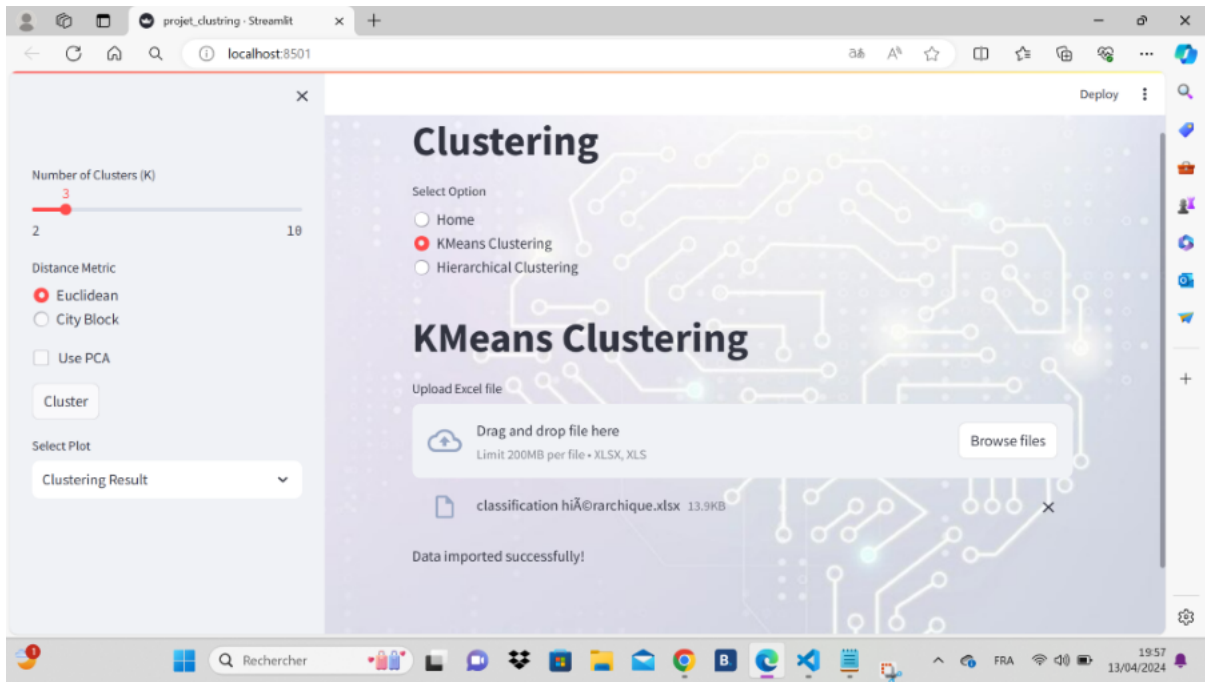


FIGURE 4 – étape 4

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.

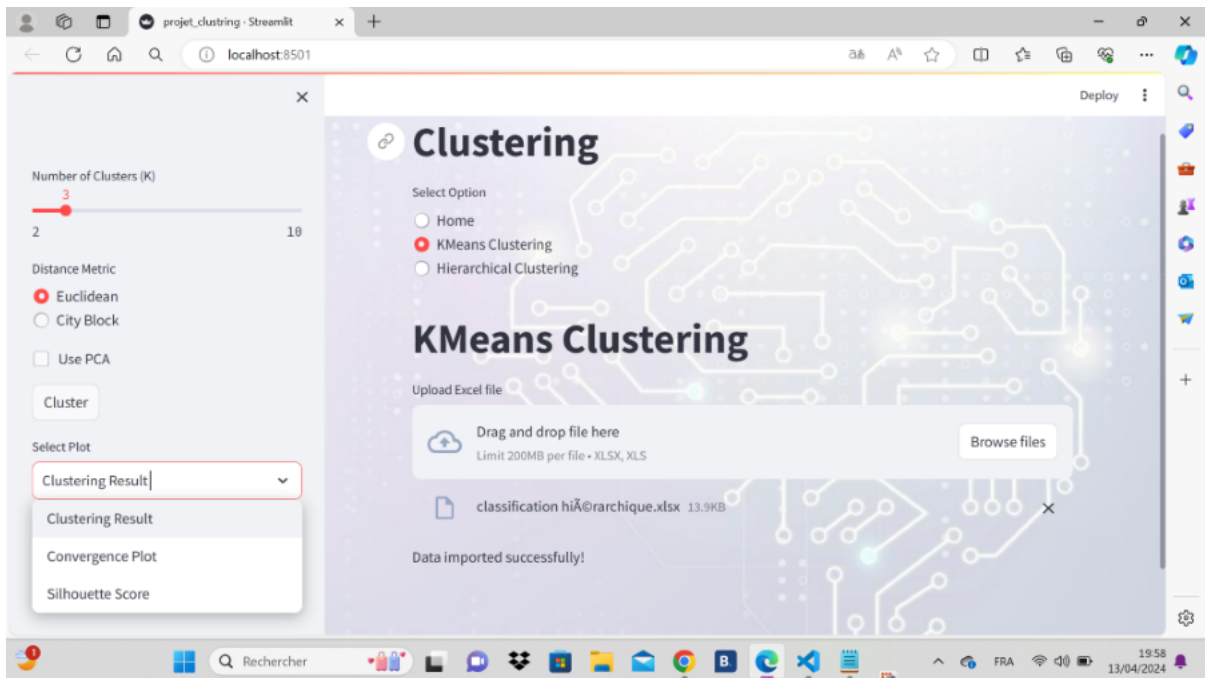


FIGURE 5 – étape 5

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

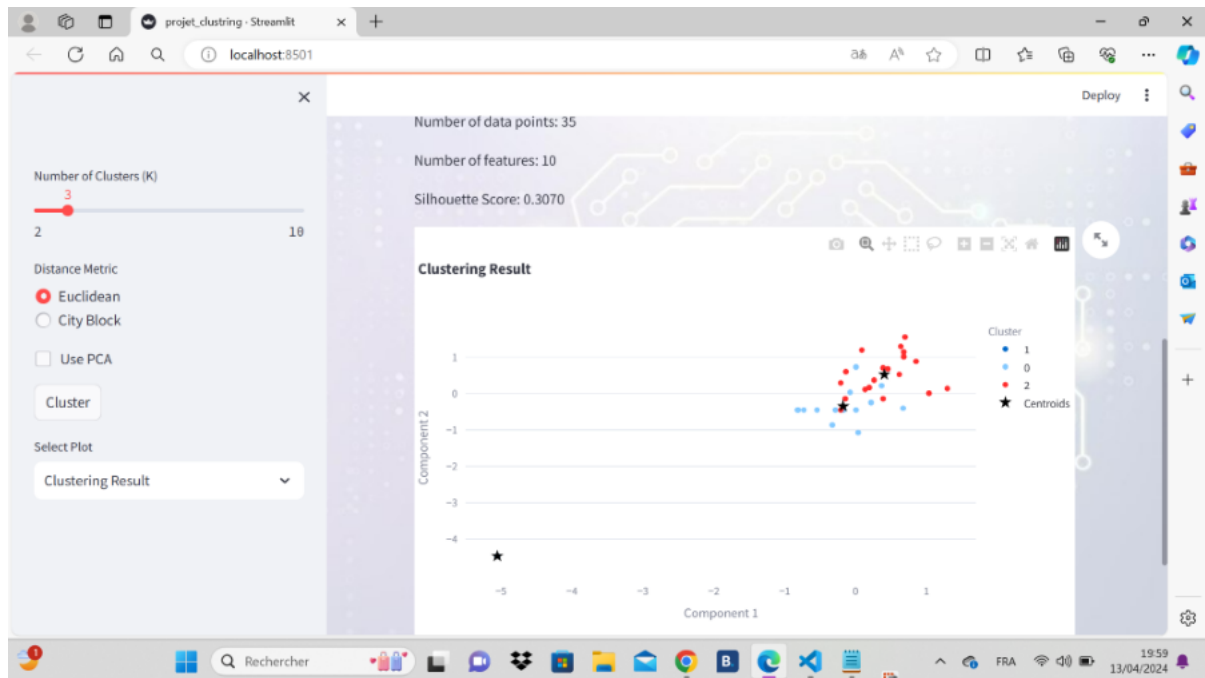


FIGURE 6 – étape 6

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

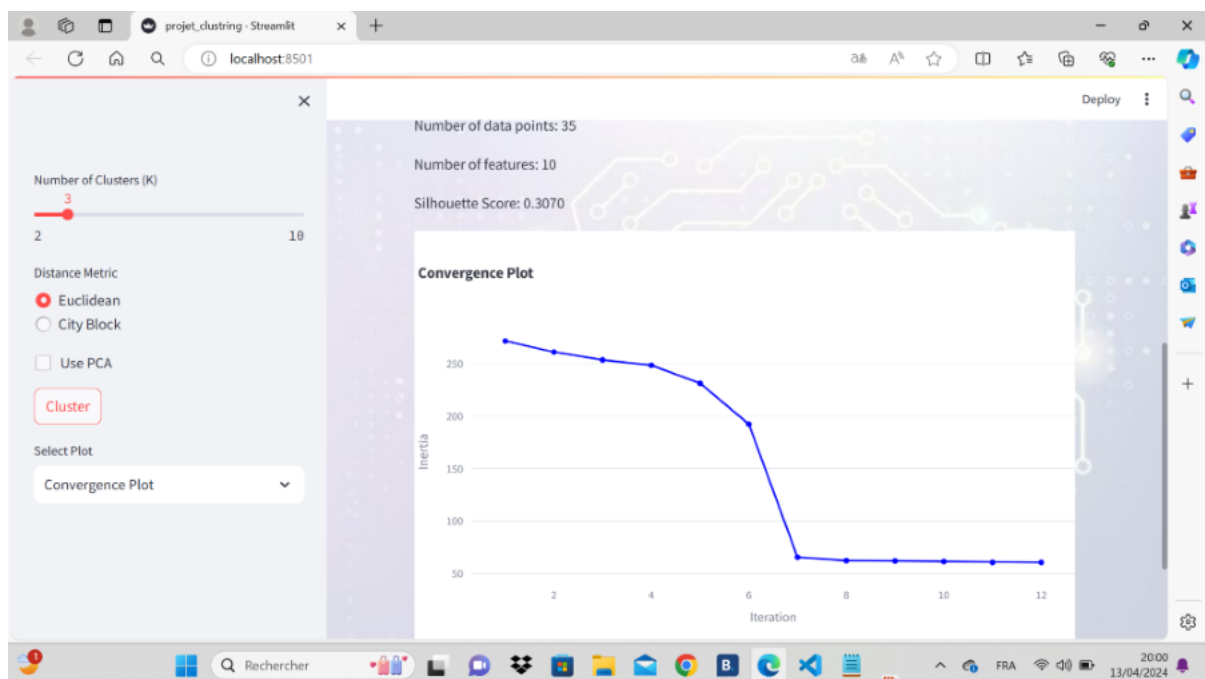


FIGURE 7 – étape 7

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.

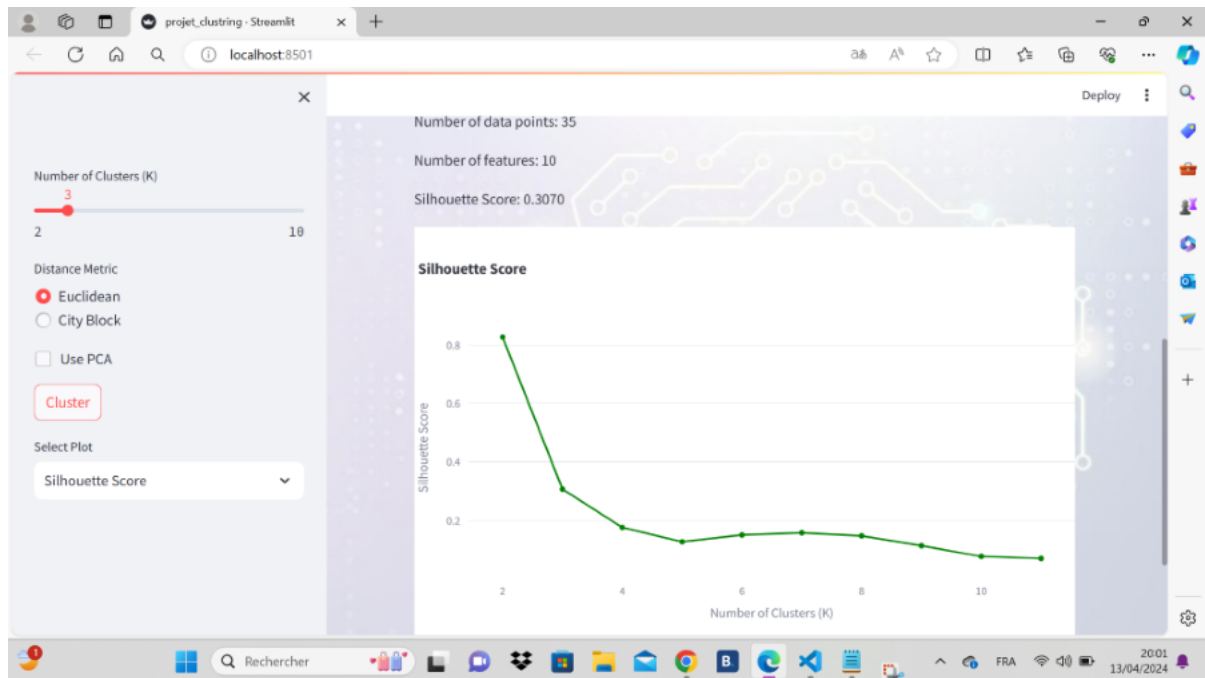


FIGURE 8 – étape 8

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

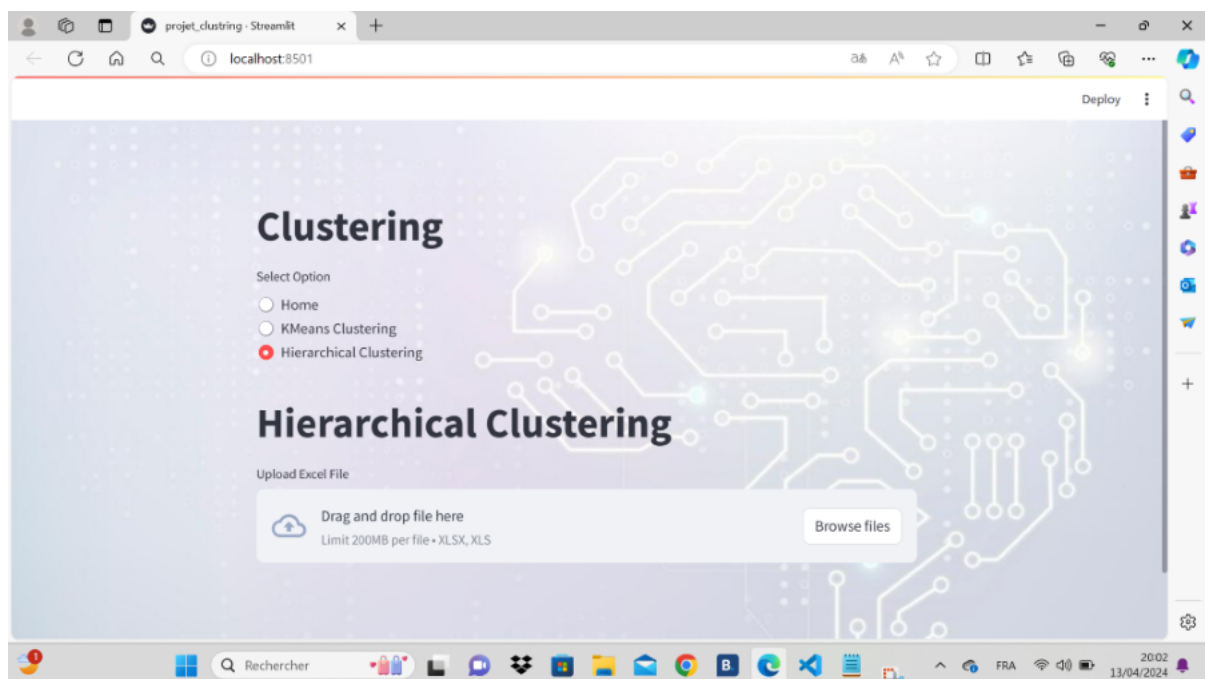


FIGURE 9 – étape 9

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, clustering 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=
4         None):
5         self.k = k
6         self.distance_metric = distance_metric
7         self.centroids = None
8         self.random_state = random_state
9
10    def calculate_distance(self, data_point, centroids):
11        if self.distance_metric == 'euclidean':
12            return np.sqrt(np.sum((centroids - data_point) ** 2, axis=1)
13        )
14        elif self.distance_metric == 'city_block':
15            return np.sum(np.abs(centroids - data_point), axis=1)
16        else:
17            raise ValueError("Invalid distance metric. Supported metrics
18                are 'euclidean' and 'city_block'.")
19
20    def fit(self, X, max_iterations=200):
21        np.random.seed(self.random_state) # Set the random seed
22
23        # Initialize centroids randomly
24        self.centroids = X[np.random.choice(X.shape[0], self.k, replace=
25            False)]
26
27        inertia_values = []
28
29        for _ in range(max_iterations):
30            # Assign each data point to the nearest centroid
31            distances = np.vstack([self.calculate_distance(data_point,
32                self.centroids) for data_point in X])
33            cluster_assignment = np.argmin(distances, axis=1)
34
35            # Update centroids
36            new_centroids = np.array([X[cluster_assignment == i].mean(
37                axis=0) for i in range(self.k)])
38
39            # Calculate inertia
40            inertia = np.sum((X - new_centroids[cluster_assignment]) **
41                2)
42            inertia_values.append(inertia)
43
44            # Check convergence
45            if np.allclose(self.centroids, new_centroids):
46                break
47
48            self.centroids = new_centroids
49
50            # Calculate silhouette score
51            labels = np.argmin(distances, axis=1)
52            silhouette_avg = silhouette_score(X, labels)

```

```

47     return cluster_assignment, self.centroids, inertia_values,
    silhouette_avg
48
49 def run_clustering(X, k, distance_metric, use_pca):
50     if use_pca:
51         pca = PCA(n_components=2)
52         X = pca.fit_transform(X)
53
54         kmeans = KMeansClustering(k=k, distance_metric=distance_metric,
    random_state=42)
55         labels, centroids, inertia_values, silhouette_avg = kmeans.fit(X)
56         return labels, centroids, inertia_values, silhouette_avg
57
58 def plot_result(X_reduced, labels, centroids):
59     if X_reduced.shape[1] == 2:
60         df = pd.DataFrame(X_reduced, columns=['Component 1', 'Component
    2'])
61     else:
62         df = pd.DataFrame(X_reduced[:, :2], columns=['Component 1', '
    Component 2'])
63     df['Cluster'] = labels.astype(str) # Convert labels to string for
    coloring purposes
64
65     fig = px.scatter(df, x='Component 1', y='Component 2', color='
    Cluster',
66                     title='Clustering Result', hover_name=df.index)
67     fig.add_trace(go.Scatter(
68         x=centroids[:, 0],
69         y=centroids[:, 1],
70         mode='markers',
71         marker=dict(size=10, color='black', symbol='star'),
72         name='Centroids'
73     ))
74     return fig
75
76 def plot_convergence(inertia_values):
77     fig = go.Figure()
78     fig.add_trace(go.Scatter(
79         x=list(range(1, len(inertia_values) + 1)),
80         y=inertia_values,
81         mode='lines+markers',
82         marker=dict(color='blue'),
83         name='Inertia'
84     ))
85     fig.update_layout(
86         title='Convergence Plot',
87         xaxis=dict(title='Iteration'),
88         yaxis=dict(title='Inertia'),
89         hovermode='closest'
90     )
91     return fig
92
93 def plot_silhouette_score(silhouette_values):
94     fig = go.Figure()
95     fig.add_trace(go.Scatter(
96         x=list(range(2, 12)),
97         y=silhouette_values,

```

```

98     mode='lines+markers',
99     marker=dict(color='green'),
100     name='Silhouette Score'
101 ))
102 fig.update_layout(
103     title='Silhouette Score',
104     xaxis=dict(title='Number of Clusters (K)'),
105     yaxis=dict(title='Silhouette Score'),
106     hovermode='closest'
107 )
108 return fig
109
110 def main():
111     st.title("KMeans Clustering")
112
113     uploaded_file = st.file_uploader("Upload Excel file", type=["xlsx",
114 "xls"])
115     if uploaded_file is not None:
116         data = pd.read_excel(uploaded_file, index_col=0)
117         st.write("Data imported successfully!")
118
119         X = data.values
120         X = X[:, 2:] # Adjust this according to your data
121
122         scaler = StandardScaler()
123         X_scaled = scaler.fit_transform(X)
124
125         k = st.sidebar.slider("Number of Clusters (K)", min_value=2,
126 max_value=10, value=3)
127         distance_metric = st.sidebar.radio("Distance Metric", ["
128 Euclidean", "City Block"], index=0)
129         distance_metric = 'euclidean' if distance_metric == 'Euclidean'
130 else 'city_block'
131
132         use_pca = st.sidebar.checkbox("Use PCA")
133
134         labels, centroids, inertia_values, silhouette_avg = None, None,
135 None, None # Initialize variables
136
137         if st.sidebar.button("Cluster"):
138             labels, centroids, inertia_values, silhouette_avg =
139 run_clustering(X_scaled, k, distance_metric, use_pca)
140
141             st.success(f"Convergence achieved in {len(inertia_values)}
142 iterations.")
143             st.write(f"Number of clusters: {k}")
144             st.write(f"Number of data points: {len(X)}")
145             st.write(f"Number of features: {X.shape[1]}")
146             st.write(f"Silhouette Score: {silhouette_avg:.4f}")
147
148             plot_choice = st.sidebar.selectbox("Select Plot", ["Clustering
149 Result", "Convergence Plot", "Silhouette Score", "Elbow Method"])
150
151             if plot_choice == "Clustering Result" and labels is not None:
152                 fig = plot_result(X_scaled, labels, centroids)
153                 st.plotly_chart(fig, use_container_width=True)

```

```

146         elif plot_choice == "Convergence Plot" and inertia_values is not
None:
147             fig = plot_convergence(inertia_values)
148             st.plotly_chart(fig, use_container_width=True)
149         elif plot_choice == "Silhouette Score" and silhouette_avg is not
None:
150             silhouette_values = []
151             for k in range(2, 12):
152                 kmeans = KMeansClustering(k=k, random_state=42)
153                 _, _, _, silhouette_avg = kmeans.fit(X_scaled)
154                 silhouette_values.append(silhouette_avg)
155             fig = plot_silhouette_score(silhouette_values)
156             st.plotly_chart(fig, use_container_width=True)
157
158 if __name__ == "__main__":
159     main()

```

Listing 9 – python script1

Code de la méthode CAH :

```

1
2 class HierarchicalClustering:
3     def __init__(self, k):
4         self.k = k
5
6     def fit(self, data, distances):
7         n = len(data)
8         clusters = [[i] for i in range(n)]
9
10        while len(clusters) > self.k:
11            min_dist = np.inf
12            for i in range(len(clusters)):
13                for j in range(i + 1, len(clusters)):
14                    cluster1 = clusters[i]
15                    cluster2 = clusters[j]
16                    avg_dist = 0
17                    count = 0
18                    for idx1 in cluster1:
19                        for idx2 in cluster2:
20                            if not np.isnan(distances[idx1, idx2]):
21                                avg_dist += distances[idx1, idx2]
22                                count += 1
23                    if count > 0:
24                        avg_dist /= count
25                        if avg_dist < min_dist:
26                            min_dist = avg_dist
27                            merge_index = (i, j)
28            i, j = merge_index
29            new_cluster = clusters[i] + clusters[j]
30            clusters[i] = new_cluster
31            clusters.pop(j)
32
33        self.labels_ = np.zeros(n)
34        for i, cluster in enumerate(clusters):
35            for idx in cluster:
36                self.labels_[idx] = i
37

```

```

38 def import_data_from_excel(file_path):
39     data = pd.read_excel(file_path)
40     # Check if the first row or first column contains non-numeric values
41     first_row_is_str = any(isinstance(val, str) for val in data.iloc[0])
42     first_column_is_str = isinstance(data.iloc[:, 0].values[0], str)
43     if first_row_is_str or first_column_is_str:
44         data = data.iloc[1:, 1:] # Exclude first row and first column
45         if they contain non-numeric values
46     return data
47
48 def dendrogram_visualization(data, labels):
49     X = data.values
50     Z = linkage(X, method='ward')
51     fig, ax = plt.subplots(figsize=(10, 5))
52     dn = dendrogram(Z, ax=ax)
53     ax.set_title("Dendrogram")
54     ax.set_xlabel("Data Points")
55     ax.set_ylabel("Distance")
56     st.pyplot(fig) # Display the plot in Streamlit
57
58 # Example usage:
59 uploaded_file = st.file_uploader("Upload Excel File", type=["xlsx", "xls
60 ])
61 if uploaded_file is not None:
62     data = import_data_from_excel(uploaded_file)
63     k = 3 # Number of clusters
64     distances = pairwise_distances(data.values, metric='euclidean')
65     clustering_algorithm = HierarchicalClustering(k=k)
66     clustering_algorithm.fit(data, distances)
67     labels = clustering_algorithm.labels_
68     dendrogram_visualization(data, labels)
    
```

Listing 10 – python script1