المدرسة الوطنية للعلوم التطبيقية - أكادير
ⵜⵉⵎⵙⵙⵓⵜ ⵏ ⵜⵎⵓⵔⵉⵡⵉⵏ ⵜⵉⵖⵓⵔⵉⵖⵉⵏ - ⴰⵣⴰⵏⵣⵓ
ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES - AGADIR

Rapport de projet d'analyse des données :

Réalisé par :

KERDOUN Wassim

filière : Finance et ingénierie décisionelle (FID1)

Documentation de la web application

Encadré par :

Prof. EL ASRI Brahim

A déposé le : **17/04/2024**

Résumé

Le projet d'analyse de données avec les techniques de clustering K-Means et Classification Hiérarchique Ascendante (CHA) offre une solution interactive et polyvalente pour l'exploration et la compréhension des structures sous-jacentes des ensembles de données non étiquetés. À travers cette application, les utilisateurs sont équipés d'outils puissants pour segmenter leurs données en clusters significatifs, facilitant ainsi l'identification de motifs, de tendances et de groupes intrinsèques.

L'utilisation de l'algorithme K-Means dans l'application permet aux utilisateurs de spécifier le nombre de clusters souhaités et la distance métrique à utiliser pour mesurer la similarité entre les observations. Cette approche de clustering partitionne l'espace des données en clusters compacts, représentés par des centroids, offrant ainsi une vue claire et concise de la structure des données. De plus, l'application propose des fonctionnalités avancées telles que la réduction de la dimensionnalité par l'analyse en composantes principales (PCA) et l'initialisation des centroids avec l'algorithme K-Means++, enrichissant ainsi l'analyse des données.

En complément du clustering avec K-Means, l'application offre également la possibilité d'explorer la Classification Hiérarchique Ascendante (CHA), une approche qui hiérarchise les données en construisant un dendrogramme basé sur les distances entre les observations. Cette méthode permet aux utilisateurs d'explorer les relations de similarité entre les observations à différents niveaux d'agrégation, offrant ainsi une perspective plus globale de la structure des données.

L'objectif principal de ce projet est de fournir aux utilisateurs un outil convivial et puissant pour l'analyse de données, leur permettant de découvrir des insights précieux à partir de leurs ensembles de données. En mettant l'accent sur l'interactivité, la visualisation et la personnalisation, cette application vise à démocratiser l'accès à des techniques avancées d'analyse de données, ouvrant ainsi la voie à de nouvelles découvertes et compréhensions dans une variété de domaines d'application.

Liste des figures :

Listings

Table des matières:

1 Packages :

Dans cette section, nous discutons des packages Python que nous avons utilisés pour développer notre application de clustering hiérarchique. Chaque package a été sélectionné en fonction de sa capacité à répondre à nos besoins spécifiques et à faciliter le processus de développement.

Streamlit

Nous avons utilisé Streamlit pour créer une interface utilisateur conviviale et interactive. Streamlit est un framework de développement rapide qui nous a permis de créer des applications web à partir de scripts Python. Nous avons utilisé Streamlit pour permettre aux utilisateurs de télécharger leurs données, de choisir les paramètres de l'algorithme de clustering et de visualiser les résultats. Vous pouvez en savoir plus sur Streamlit dans la documentation officielle : `https://docs.streamlit.io/`.

```
import streamlit as st
```

Pandas et NumPy

Nous avons utilisé Pandas et NumPy pour la manipulation et l'analyse des données. Pandas est une bibliothèque Python puissante pour la manipulation de données tabulaires, tandis que NumPy est une bibliothèque pour le calcul numérique en Python. Vous pouvez en savoir plus sur Pandas dans la documentation officielle : `https://pandas.pydata.org/docs/` et sur NumPy dans la documentation officielle : `https://numpy.org/doc/`.

```
import pandas as pd
import numpy as np
```

Matplotlib et Plotly

Nous avons utilisé Matplotlib pour la création de visualisations statiques et Plotly pour les visualisations interactives. Matplotlib est une bibliothèque Python pour la création de graphiques en 2D, tandis que Plotly est une bibliothèque de graphiques interactive. Vous pouvez en savoir plus sur Matplotlib dans la documentation officielle : `https://matplotlib.org/stable/contents.html` et sur Plotly dans la documentation officielle : `https://plotly.com/python/`.

```
import matplotlib.pyplot as plt
import plotly.graph_objs as go
```

SciPy

Nous avons utilisé SciPy pour la hiérarchisation des clusters et la création du dendrogramme. SciPy est une bibliothèque Python open-source pour les mathématiques, les sciences et l'ingénierie. Vous pouvez en savoir plus sur SciPy dans la documentation officielle : `https://docs.scipy.org/doc/scipy/reference/`.

```
from scipy.cluster.hierarchy import linkage, dendrogram
```

Scikit-learn

Nous avons utilisé Scikit-learn pour le calcul des distances pairwise, la réduction de dimension avec PCA et la standardisation des données. Scikit-learn est une bibliothèque Python pour l'apprentissage automatique et l'analyse de données. Vous pouvez en savoir plus sur Scikit-learn dans la documentation officielle : `https://scikit-learn.org/stable/`.

```
from sklearn.metrics import pairwise_distances
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

Autres Packages

Nous avons également utilisé d'autres packages tels que streamlit_option_menu pour la création de menus déroulants dans Streamlit, base64 pour le traitement des fichiers, et BytesIO pour la manipulation des flux d'octets. Vous pouvez en savoir plus sur ces packages dans leur documentation respective.

```
from streamlit_option_menu import option_menu
import base64
from io import BytesIO
```

2

K-Means :

1 Introduction à K-Means :

L'algorithme K-Means est l'une des techniques de clustering les plus couramment utilisées en analyse de données et en apprentissage automatique. Son objectif principal est de regrouper un ensemble de données non étiquetées en un nombre prédéfini de clusters, où chaque observation appartient au cluster dont le centroïde est le plus proche.

- Contexte et Importance : Dans le domaine de l'analyse de données, la capacité à identifier des structures significatives au sein de grands ensembles de données est essentielle pour extraire des informations précieuses et prendre des décisions éclairées. Le clustering, et en particulier l'algorithme K-Means, joue un rôle crucial dans cette tâche en permettant la segmentation automatique des données en groupes homogènes.
- Fonctionnement de l'Algorithme : L'algorithme K-Means fonctionne de manière itérative pour répartir les observations en K clusters, où K est un paramètre défini par l'utilisateur. Initialement, il sélectionne aléatoirement K centroids, qui servent de points de départ pour chaque cluster. Ensuite, l'algorithme alterne entre deux étapes principales : l'affectation des observations au cluster le plus proche et la mise à jour des centroids en calculant la moyenne des observations de chaque cluster. Ce processus se répète jusqu'à ce qu'une convergence soit atteinte, c'est-à-dire que les centroids ne bougent plus ou que le critère d'arrêt spécifié soit satisfait.
- Applications de l'Algorithme :

  L'algorithme K-Means est largement utilisé dans divers domaines, y compris l'analyse des clients pour la segmentation du marché, la catégorisation des documents dans le traitement du langage naturel, la détection des anomalies dans les données industrielles, et bien plus encore. Sa simplicité, son efficacité et sa capacité à gérer de grands ensembles de données en font un outil polyvalent pour l'exploration et l'analyse de données.
- Objectifs de ce Chapitre :

  Dans ce chapitre, nous explorerons en détail l'algorithme K-Means, en fournissant une explication approfondie de son fonctionnement, de ses avantages et de ses limitations. Nous examinerons également comment l'algorithme est mis en œuvre dans notre application d'analyse de données, en mettant l'accent sur les fonctionnalités avancées et les bonnes pratiques pour obtenir des résultats précis et significatifs. Enfin, nous illustrerons l'utilisation de l'algorithme à travers des exemples concrets et des analyses détaillées de ses performances.

2 Déscription de l'algorithme :

L'algorithme K-Means est l'une des techniques de clustering les plus populaires utilisées en analyse de données. Il vise à partitionner un ensemble de données en un nombre prédéfini de clusters, où chaque observation appartient au cluster dont le centroid est le plus proche. Voici une description détaillée de l'algorithme K-Means :

1. **Initialisation des centroids** :
   - Choix initial des centroids : Les centroids initiaux sont généralement sélectionnés de manière aléatoire parmi les observations de l'ensemble de données.
2. **Assignation des observations aux clusters** :
   - Calcul de la distance : Pour chaque observation de l'ensemble de données, la distance entre cette observation et chaque centroid est calculée. Les distances peuvent être mesurées en utilisant différentes métriques, telles que la distance euclidienne ou la distance de Manhattan.

     A noter : Soit un espace n-dimensionnel, la distance entre deux points p et q est comme suit :

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

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

   - 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 des centroids** :
   - Calcul des nouveaux centroids : Une fois que toutes les observations ont été assignées à des clusters, les nouveaux centroids sont calculés en prenant la moyenne des observations appartenant à chaque cluster.
   - Mise à jour des centroids : Les centroids sont mis à jour avec les nouvelles valeurs calculées.
4. **Répétition des étapes 2 et 3** :
   - Itérations : Les étapes d'assignation des observations aux clusters et de mise à jour des centroids sont répétées jusqu'à ce qu'un critère d'arrêt soit atteint. Ce critère peut être le nombre maximal d'itérations, la convergence des centroids, ou une diminution de l'inertie intra-cluster.
5. **Critère d'arrêt** :
   - Convergence : L'algorithme converge lorsque les centroids ne changent plus ou lorsque la variation de l'inertie intra-cluster entre deux itérations successives est négligeable. Cette convergence peut être décrite mathématiquement comme le cite le théorème suivant :

     Definition 2.1 (Convergence)**.** La convergence de l'algorithme K-Means peut être définie mathématiquement de la manière suivante :
     Soit $C = \{C_1, C_2, ..., C_k\}$ l'ensemble des clusters, où k est le nombre de clusters prédéfini. Soit $\mu = \{\mu_1, \mu_2, ..., \mu_k\}$ l'ensemble des centroids correspondants.
     L'algorithme K-Means converge lorsque les centroids ne changent plus ou lorsque

la variation de l'inertie intra-cluster entre deux itérations successives est négligeable.

La convergence peut être exprimée par les conditions suivantes :

Variation des centroids : Les centroids ne changent plus entre deux itérations successives, c'est-à-dire que $\mu_i(t) = \mu_i(t+1)$ pour tout i, où t représente l'itération actuelle.

Variation de l'inertie intra-cluster : La variation de l'inertie intra-cluster entre deux itérations successives est négligeable. L'inertie intra-cluster $J(C)$ est définie comme la somme des distances au carré entre chaque observation et le centroid de son cluster, en d'autres termes :

$$J(C) = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

où :

- k est le nombre de clusters,
- C_i représente le i-ème cluster,
- x est une observation dans le cluster C_i,
- μ_i est le centroid du cluster C_i,
- $\|x - \mu_i\|^2$ est la distance euclidienne au carré entre l'observation x et le centroid μ_i.

Ainsi, la variation de l'inertie intra-cluster peut être exprimée comme suit :

$$\frac{|J(C^{(t+1)}) - J(C^{(t)}|}{J(C^{(t)})}$$

Où $J(C^{(t)})$ et $J(C^{(t+1)})$ représentent l'inertie intra-cluster à l'itération t et t+1 respectivement.

L'algorithme K-Means converge lorsque l'une de ces conditions est satisfaite.

6. **Évaluation des clusters** :
   - Inertie intra-cluster : Une fois que les clusters sont formés, l'inertie intra-cluster est calculée comme la somme des distances au carré entre chaque observation et le centroid de son cluster. Une faible inertie intra-cluster indique des clusters compacts et bien séparés.

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 telles que la méthode du coude (Elbow method) ou la validation croisée.

⚠ Remarque :

— Méthode du coude (Elbow method) : La méthode du coude est une approche heuristique utilisée pour déterminer le nombre optimal de clusters dans un ensemble de données. Elle consiste à exécuter l'algorithme K-Means pour différents nombres de clusters et à calculer la somme des carrés des distances de chaque point à son centroid (inertie intra-cluster) pour chaque solution. Ensuite, un graphique est tracé en fonction du nombre de clusters et de l'inertie intra-cluster.

L'idée principale est de sélectionner le nombre de clusters au point où l'augmentation de la qualité de clustering (diminution de l'inertie intra-cluster) décroît fortement,

formant ainsi un coude dans le graphique. Ce point est souvent interprété comme le nombre optimal de clusters à utiliser.

— Initialisation K-Means++ : L'initialisation K-Means++ est une méthode utilisée pour sélectionner les centroids initiaux de manière plus intelligente que l'initialisation aléatoire dans l'algorithme K-Means. Plutôt que de choisir les centroids initiaux de manière aléatoire, cette méthode les sélectionne en suivant une approche probabiliste.

  L'algorithme d'initialisation K-Means++ fonctionne en deux étapes. Dans la première étape, un centroid est choisi de manière aléatoire parmi les données d'entrée. Ensuite, pour chaque point restant, la probabilité de sélection comme centroid est proportionnelle au carré de la distance minimale entre ce point et les centroids déjà sélectionnés. Ce processus est répété jusqu'à ce que tous les centroids soient sélectionnés.

  L'initialisation K-Means++ conduit généralement à une meilleure convergence de l'algorithme K-Means et à des résultats de clustering plus stables par rapport à l'initialisation aléatoire, en particulier pour des ensembles de données complexes.

3 Avantages et Limitations de l'Algorithme K-Means :

L'algorithme K-Means est largement utilisé pour ses nombreux avantages, mais il présente également des limitations à prendre en considération lors de son application. Cette section examine en détail les avantages et les limitations de K-Means, offrant ainsi un aperçu complet de ses capacités et de ses défis.

3.1 Avantages :

1. **Simplicité et Facilité d'Implémentation** :
   — K-Means est simple à comprendre et à mettre en œuvre, ce qui en fait un choix populaire pour de nombreux praticiens. Son processus itératif est facile à suivre, même pour les utilisateurs débutants en analyse de données.
2. **Efficacité en Temps et en Espace** :
   — L'algorithme K-Means est relativement rapide et efficace, ce qui lui permet de traiter rapidement de grands ensembles de données. Il est donc adapté à une utilisation dans des applications nécessitant une réponse rapide.
3. **Évolutivité** :
   — K-Means peut être facilement parallélisé, ce qui permet de l'utiliser efficacement sur des ensembles de données massifs. Cette capacité d'évolutivité le rend approprié pour l'analyse de données à grande échelle.
4. **Interprétabilité des Résultats** :
   — Les clusters générés par K-Means sont généralement faciles à interpréter. Chaque cluster est représenté par son centroid, ce qui permet une visualisation claire et une interprétation intuitive des résultats.

3.2 Limitations :

1. **Sensibilité aux Centroids Initiaux** :
   — Les performances de K-Means dépendent fortement du choix initial des centroids. Une mauvaise initialisation peut entraîner une convergence vers un optimum local plutôt que global, affectant ainsi la qualité des clusters obtenus.
2. **Dépendance à la Distance Euclidienne** :
   — K-Means utilise la distance euclidienne pour mesurer la similarité entre les points, ce qui le rend sensible aux valeurs aberrantes et aux données non linéaires. Il peut donc produire des résultats suboptimaux dans des cas où la distance euclidienne n'est pas appropriée.
3. **Nécessité de Spécifier le Nombre de Clusters** :
   — Un inconvénient majeur de K-Means est la nécessité de spécifier le nombre de clusters à l'avance. Cette information peut ne pas être évidente dans certains cas, ce qui rend difficile le choix du nombre optimal de clusters.
4. **Incapacité à Gérer les Clusters de Formes et Tailles Variées** :
   — K-Means fonctionne mieux lorsque les clusters ont une forme sphérique et une taille similaire. Il peut avoir du mal à détecter des clusters de formes complexes ou de tailles très différentes, ce qui peut entraîner une segmentation inappropriée des données.

En résumé, bien que l'algorithme K-Means présente plusieurs avantages significatifs, il est important de prendre en compte ses limitations lors de sa mise en œuvre dans des projets d'analyse de données. En comprenant ses forces et ses faiblesses, les praticiens peuvent utiliser K-Means de manière plus efficace et en tirer le meilleur parti pour leurs applications spécifiques.

4 Implémentation de l'Algorithme K-Means

4.1 Implémentation de la Classe `KMeansClustering`

⚠ Le code de la classe se trouve dans la partie Annexe (Kmeans class code).

Dans cette section, nous détaillons l'implémentation de la classe `KMeansClustering`, qui est responsable de l'exécution de l'algorithme K-Means pour le clustering des données.

Attributs de la Classe

La classe `KMeansClustering` possède plusieurs attributs qui sont utilisés pour l'exécution de l'algorithme K-Means.

- `k` : Le nombre de clusters à former.
- `distance_metric` : La métrique de distance utilisée pour mesurer la similarité entre les points.
- `centroids` : Les centroids des clusters.
- `random_state` : La graine aléatoire pour la reproductibilité des résultats.
- `inertia_history` : Une liste contenant l'inertie du clustering à chaque itération.

Méthodes de la Classe

La classe `KMeansClustering` possède plusieurs méthodes pour l'exécution de l'algorithme K-Means.

- `__init__` : Le constructeur de la classe qui initialise les attributs.
- `fit` : La méthode principale pour exécuter l'algorithme K-Means. Elle prend en entrée les données `X` et retourne les étiquettes de cluster, les centroids et l'historique de l'inertie.
- `calculate_distances` : Une méthode privée pour calculer les distances entre les points et les centroids.

Utilisation de la Classe

Pour utiliser la classe `KMeansClustering`, il suffit d'instancier un objet avec les paramètres appropriés et d'appeler la méthode `fit` avec les données à clusteriser. Voici un exemple d'utilisation de la classe :

```
# Instanciation de l'objet KMeansClustering
kmeans = KMeansClustering(k=3, distance_metric='euclidean', random_state=42)

# Execution de l'algorithme K-Means sur les donnees X
labels, centroids, inertia_history = kmeans.fit(X)
```

Listing 2.1 – Appel à la classe

En utilisant la classe `KMeansClustering`, nous pouvons facilement exécuter l'algorithme K-Means sur nos données et obtenir les résultats du clustering.

4.2 Fonctionnalités Supplémentaires

Les fonctionnalités supplémentaires sont des extensions de l'algorithme de clustering K-Means qui améliorent sa performance, sa flexibilité et sa polyvalence. Elles ont été ajoutées pour répondre à des besoins spécifiques et pour offrir aux utilisateurs des outils avancés pour l'analyse de leurs données.

Les fonctionnalités supplémentaires comprennent la réduction de dimension avec PCA, la méthode du coude (Elbow method), le tracé de convergence et l'initialisation K-Means++. Chacune de ces fonctionnalités apporte une valeur ajoutée à l'algorithme K-Means et permet aux utilisateurs d'explorer et d'analyser leurs données de manière plus approfondie.

— **Réduction de Dimension avec PCA**

Description de l'Algorithme :
La Réduction de Dimension avec PCA (Principal Component Analysis) est une technique largement utilisée pour réduire la dimensionnalité des données tout en préservant au mieux leur structure. Elle permet de représenter les données dans un espace de dimension réduite en projetant les données sur les composantes principales qui capturent la variance maximale dans les données d'origine. PCA est souvent utilisée avant l'application de méthodes de clustering comme K-Means pour améliorer la performance et l'interprétabilité des résultats.

Explication du Code :
Le code Python pour la réduction de dimension avec PCA utilise la bibliothèque scikit-learn, qui fournit une implémentation efficace de l'algorithme PCA. Voici comment cela est généralement implémenté :

```
from sklearn.decomposition import PCA

# Appliquer PCA pour reduire la dimension des donnees
pca = PCA(n_components=2)  # Specifier le nombre de composantes principales a
    conserver
X_pca = pca.fit_transform(X)  # Reduire la dimension des donnees

# Afficher les donnees reduites dans l'espace de dimension reduite
plot_result(X_pca, labels, centroids_pca, display_statistics=False)
```

Listing 2.2 – PCA dimensionality reduction code

⚠ **Remarque :** Dans la librairie scikit-learn (sklearn), la méthode PCA (Principal Component Analysis) est déjà implémentée de manière efficace et optimisée. Elle offre des fonctionnalités complètes pour effectuer la réduction de dimensionnalité et est largement utilisée dans la communauté de l'apprentissage automatique.

Ce code utilise la classe 'PCA' de scikit-learn pour effectuer la réduction de dimension. La méthode 'fit_transform()' est utilisée pour ajuster le modèle PCA aux données et transformer les données d'origine en un nouvel espace de dimension réduite. Les données réduites sont ensuite affichées dans l'espace de dimension réduite à l'aide de la fonction 'plot_result()'.

— **Méthode du Coude**

Description de la Méthode :
La méthode du coude est une technique utilisée pour déterminer le nombre optimal de clusters dans un ensemble de données pour les algorithmes de clustering comme K-Means. Elle consiste à tracer la variation de l'inertie intra-cluster en fonction du nombre de clusters et à identifier le point où la courbe présente un coude ou un changement significatif de pente. Ce point indique le nombre optimal de clusters à choisir.

Explication du Code :
Dans le code Python, la méthode du coude est implémentée pour visualiser la variation de l'inertie intra-cluster en fonction du nombre de clusters. Voici comment cela est généralement réalisé :

```
    def plot_elbow_method():
    st.title("Methode du Coude")
    cluster_data = st.session_state.get('cluster_data')
    if cluster_data:
        X_scaled = cluster_data['X_scaled']
        inertia = calculate_inertia(X_scaled)
        st.plotly_chart(plot_inertia(inertia), use_container_width=True)
    else:
        st.info("Veuillez executer le clustering sur la page d'accueil d'abord."
    )
```

Listing 2.3 – Elbow method code

Ce code utilise les données de clustering pour calculer l'inertie intra-cluster pour différentes valeurs de k (nombre de clusters). La fonction 'calculate_inertia()' calcule l'inertie pour chaque valeur de k en utilisant l'algorithme K-Means. Ensuite, la fonction 'plot_inertia()' est utilisée pour tracer la courbe d'inertie par rapport au nombre de clusters. Le point où la courbe présente un coude indique le nombre optimal de clusters.

— **Plot de Convergence**

Description de la Convergence :

Le plot de convergence est un outil visuel utilisé pour suivre la convergence de l'algorithme K-Means au fil des itérations. Il affiche l'inertie, qui est la somme des distances au carré des points par rapport à leurs centroids, en fonction du nombre d'itérations. L'inertie diminue à chaque itération jusqu'à ce que l'algorithme converge vers une solution stable. Le plot de convergence permet de vérifier si l'algorithme a convergé et à quel point.

Explication du Code :

Dans le code Python, la convergence est vérifiée et tracée à l'aide des données de clustering. Voici un exemple de code pour réaliser cela :

```
def plot_convergence(cluster_data):
    st.title("Plot de Convergence")
    if cluster_data:
        inertia_history = cluster_data['inertia_history']
        convergence_iteration = len(inertia_history)
        final_inertia = inertia_history[-1]
        fig = go.Figure(data=go.Scatter(x=np.arange(1, len(inertia_history) + 1)
    ,
                                        y=inertia_history, mode='lines+markers')
    )
        fig.update_layout(title="Convergence: Inertie vs Iterations",
                          xaxis_title="Iterations",
                          yaxis_title="Inertie")

        fig.update_traces(line=dict(color='#eb4034'))

        # Affichage des informations supplementaires
        st.plotly_chart(fig)
        st.info(f"Convergence atteinte a l'iteration {convergence_iteration}.")
        st.info(f"Inertie finale : {final_inertia}")

    else:
        st.info("Veuillez executer le clustering sur la page d'accueil d'abord."
    )
```

Listing 2.4 – Convergence Plot code

Ce code utilise l'historique de l'inertie intra-cluster pour tracer la courbe de convergence. Il affiche également des informations sur le point de convergence et l'inertie finale.

— **Initialisation K-Means++**

Description de l'Algorithme :

L'initialisation K-Means++ est une méthode pour sélectionner les centroids initiaux de manière déterministe et intelligente afin d'améliorer la convergence et la qualité des clusters dans l'algorithme K-Means. Voici comment fonctionne l'algorithme K-Means++ :

1. **Sélection du Premier Centroid :** Un centroid est choisi au hasard parmi les données.
2. **Calcul des Probabilités de Sélection :** Pour chaque point de données restant, la distance au centroid le plus proche est calculée. Les probabilités de sélection de chaque point comme prochain centroid sont calculées proportionnellement à la distance au carré.
3. **Sélection des Centroids Suivants :** Les centroids suivants sont sélectionnés en tirant aléatoirement un point de données selon les probabilités calculées précédemment. Ce processus est répété jusqu'à ce que tous les centroids soient sélectionnés.

Explication du Code :

Le code Python pour l'initialisation K-Means++ comprend principalement deux étapes :

1. **Sélection du Premier Centroid :** Un point de données est choisi au hasard comme premier centroid.
2. **Calcul des Probabilités de Sélection :** Pour chaque point de données restant, la distance au carré du centroid le plus proche est calculée. Les probabilités de sélection de chaque point sont calculées proportionnellement à ces distances au carré.

Le code correspondant pour l'initialisation K-Means++ dans la classe `KMeansClustering` serait le suivant :

```
def _initialize_centroids(self, X):
    centroids = [X[np.random.randint(X.shape[0])]]  # Selection du premier centroid
    for _ in range(1, self.k):
        distances = np.array([min([np.linalg.norm(x - c)**2 for c in centroids])
                              for x in X])  # Calcul des distances au carre
        probabilities = distances / distances.sum()  # Calcul des probabilites
        centroids.append(X[np.random.choice(len(X), p=probabilities)])  # Selection du prochain centroid
    return np.array(centroids)
```

Listing 2.5 – K-Means++ code

Ce code illustre la manière dont l'initialisation K-Means++ est implémentée dans la méthode `_initialize_centroids()` de la classe `KMeansClustering`.

Après l'exécution de l'algorithme de clustering, il est essentiel d'analyser les résultats obtenus pour comprendre la structure des clusters formés. Pour cela, des statistiques des clusters, telles que la taille de chaque cluster et la distance entre les centroids, sont souvent fournies. Ces statistiques offrent un aperçu précieux de la répartition des données dans les clusters ainsi que de la séparation entre ces derniers. Elles aident les utilisateurs à évaluer la qualité et la signification des clusters identifiés, ce qui peut orienter les décisions ultérieures en matière d'analyse ou de prise de décision.

— **Statistiques des clusters**

La fonction de traçage pour les statistiques des clusters affiche la taille de chaque cluster ainsi que la distance entre les centroids. Elle fournit une vue d'ensemble des caractéristiques de chaque cluster et de leur répartition dans l'espace des données.

```python
def plot_cluster_statistics(X, labels, centroids):
    # Calcul de la taille de chaque cluster
    cluster_sizes = {}
    for label in np.unique(labels):
        cluster_sizes[label] = np.sum(labels == label)

    # Calcul de la distance entre les centroids
    centroid_distances = {}
    for i, centroid1 in enumerate(centroids):
        for j, centroid2 in enumerate(centroids):
            if i < j:
                distance = np.linalg.norm(centroid1 - centroid2)
                centroid_distances[f"Centroid {i} - Centroid {j}"] = distance

    # Affichage des statistiques
    st.subheader("Cluster Sizes:")
    for label, size in cluster_sizes.items():
        st.write(f"Cluster {label}: {size} points")

    st.subheader("Distance Between Centroids:")
    for key, distance in centroid_distances.items():
        st.write(f"{key}: {distance:.2f}")
```

Listing 2.6 – Statistics code

Explication :
Cette fonction prend en entrée les données (`X`), les étiquettes des clusters (`labels`) et les centroids calculés (`centroids`). Elle calcule ensuite la taille de chaque cluster en comptant le nombre de points attribués à chaque étiquette de cluster. Ensuite, elle calcule la distance entre chaque paire de centroids à l'aide de la norme euclidienne. Enfin, elle affiche les statistiques des clusters, y compris la taille de chaque cluster et la distance entre les centroids.

Cette fonction est utile car elle fournit des informations détaillées sur la répartition des données dans les clusters, permettant ainsi une meilleure compréhension de la structure des données.

4.3 Traçage des clusters dans K-Means :

La fonction `plot_result` est utilisée pour tracer les clusters résultants d'une opération de clustering K-Means. Voici une explication détaillée de son fonctionnement :
Entrées :

- `X` : Les données sur lesquelles le clustering a été effectué, généralement dans un espace de dimension réduit.
- `labels` : Les étiquettes de cluster attribuées à chaque point de données dans `X`.
- `centroids` : Les coordonnées des centroïdes de chaque cluster.
- `display_statistics` : Un paramètre booléen indiquant si les statistiques des clusters doivent être affichées.

La fonction crée des traces pour chaque cluster à partir des données `X`, des étiquettes `labels`, et des couleurs définies. Chaque trace représente un cluster sous forme d'un nuage de points sur le graphique. Les coordonnées des points de données dans chaque cluster sont extraites de `X` en fonction des étiquettes `labels`. Chaque trace est configurée pour afficher le nom du cluster dans la légende du graphique.

Les statistiques des clusters sont calculées pour chaque cluster, en stockant la taille du cluster dans un dictionnaire. Une trace supplémentaire est créée pour représenter les centroïdes de chaque cluster. Les coordonnées des centroïdes sont fournies dans `centroids`. Les centroïdes sont affichés comme des étoiles blanches sur le graphique.
La mise en page du graphique est définie avec un titre, des étiquettes d'axes et un mode de survol. La largeur et la hauteur du graphique sont spécifiées pour assurer qu'il s'affiche correctement dans l'application.
Le graphique est ensuite affiché à l'aide de la fonction `plotly_chart` de Streamlit.

```
    def plot_result(X, labels, centroids, display_statistics=True):
    # Define colors for clusters
    colors = ['rgba(31, 119, 180, 0.5)', 'rgba(255, 127, 14, 0.5)', 'rgba(44, 160, 44, 0.5)', 'rgba(214, 39, 40, 0.5)',
              'rgba(148, 103, 189, 0.5)', 'rgba(140, 86, 75, 0.5)', 'rgba(227, 119, 194, 0.5)', 'rgba(127, 127, 127, 0.5)',
              'rgba(188, 189, 34, 0.5)', 'rgba(23, 190, 207, 0.5)']

    # Create trace for each cluster
    traces = []
    for i, cluster in enumerate(sorted(np.unique(labels))):
        data = X[labels == cluster]
        trace = go.Scatter(
            x=data[:, 0],
            y=data[:, 1],
            mode='markers',
            name=f'Cluster {cluster}',
            marker=dict(size=10, color=colors[i]),
            line=dict(color=colors[i].replace('0.5', '1'), width=2)
        )
        traces.append(trace)

    # Add centroids trace
    centroid_trace = go.Scatter(
        x=centroids[:, 0],
        y=centroids[:, 1],
        mode='markers',
        name='Centroids',
        marker=dict(size=10, color='white', symbol='star'),
    )
    traces.append(centroid_trace)

    # Create layout with wider parameters
    layout = go.Layout(
        title='K-Means Clustering',
        xaxis=dict(title='Component 1'),
        yaxis=dict(title='Component 2'),
        hovermode='closest',
        width=1085,  # Set the width to 1200 pixels
        height=457.59,  # Set the height to 800 pixels
    )

    # Create figure
    fig = go.Figure(data=traces, layout=layout)

    # Display figure
    st.plotly_chart(fig)
```

Listing 2.7 – Plot result code

Classification hiérarchique ascendante : 3

1 Introduction :

La classification hiérarchique ascendante (CAH) est une méthode de regroupement de données fondamentale dans le domaine de l'apprentissage non supervisé. Son objectif principal est de découvrir la structure inhérente des données en les organisant dans une hiérarchie de clusters, offrant ainsi des perspectives précieuses sur les relations entre les observations. Contrairement à d'autres approches de clustering qui séparent les données en un nombre prédéfini de groupes, la CAH offre une vue granulaire des similitudes et des différences à différentes échelles, ce qui en fait un outil puissant pour l'exploration et la compréhension des ensembles de données complexes.

Le but de ce chapitre est d'explorer en profondeur les concepts, les techniques et les applications de la classification hiérarchique ascendante, en mettant l'accent sur son implémentation pratique dans une application web interactive utilisant Streamlit. Nous nous engageons à fournir une compréhension complète de l'algorithme de CAH, de ses différentes approches et de ses implications dans la résolution de problèmes du monde réel. En outre, nous visons à fournir aux lecteurs les outils et les connaissances nécessaires pour utiliser efficacement la CAH dans leurs propres projets, en mettant en évidence les meilleures pratiques et les pièges à éviter.

Dans ce chapitre, nous entamerons notre exploration de la CAH en présentant d'abord le fonctionnement de l'algorithme, en détaillant ses différentes étapes et en discutant de ses variantes. Nous examinerons ensuite l'implémentation pratique de la CAH dans une application web interactive, en mettant en œuvre une interface utilisateur conviviale pour le clustering des données. Nous aborderons également les avantages et les limitations de la CAH, ainsi que ses fonctionnalités et ses applications dans divers domaines. Enfin, nous explorerons les techniques de visualisation du dendrogramme, qui permettent de représenter graphiquement la structure hiérarchique des clusters.

En résumé, ce chapitre vise à démystifier la classification hiérarchique ascendante, à la rendre accessible à un large public et à démontrer son potentiel en tant qu'outil essentiel pour l'analyse exploratoire des données et la découverte de connaissances dans divers domaines d'application.

2 Déscription de l'algorithme :

L'algorithme de Classification Hiérarchique Ascendante (CAH) commence par initialiser chaque point de données comme un cluster individuel. Ensuite, il calcule la matrice des distances entre toutes les paires de points de données, généralement en utilisant des mesures telles que la distance euclidienne ou la distance de Manhattan. Cette matrice de distances est essentielle pour déterminer quels clusters fusionner à chaque étape.
Ensuite, l'algorithme itère à travers les étapes suivantes :

1. **Fusion des clusters** : À chaque étape, les deux clusters les plus similaires sont fusionnés pour former un nouveau cluster. La similarité entre les clusters peut être mesurée de différentes manières, telles que la distance minimale (lien simple), la distance maximale (lien complet), la distance moyenne ou la méthode de Ward, qui utilise un indice d'agrégation pour mesurer la similarité entre les clusters avant et après leur fusion. L'indice de Ward est calculé en comparant la somme des carrés des différences entre chaque point dans un cluster et le centroid du cluster, avant et après la fusion.
2. **Mise à jour de la matrice des distances** : Après chaque fusion de clusters, la matrice des distances est mise à jour pour refléter la similarité entre le nouveau cluster et les autres clusters. Cette mise à jour implique généralement le recalcul des distances entre le nouveau cluster et tous les autres clusters.
3. **Construction du dendrogramme** : Pendant le processus de fusion, un dendrogramme est construit pour représenter la hiérarchie des clusters. Chaque nœud du dendrogramme représente un cluster, et la distance entre les branches du dendrogramme indique la similarité entre les clusters.

Le processus continue jusqu'à ce qu'il ne reste qu'un seul cluster contenant tous les points de données, et le dendrogramme final peut être utilisé pour choisir le nombre optimal de clusters en coupant l'arbre à un niveau approprié.
En utilisant des formules mathématiques telles que la distance euclidienne, la distance de Manhattan et l'indice de Ward, l'algorithme de CAH peut regrouper efficacement les données en clusters hiérarchiques tout en préservant les relations de similarité entre les individus.

Généralement, il existe 2 méthodes pour la mesure de similiarité utilisée dans la Classification Hiérarchique Ascendante (CAH) :

Definition 3.1 (Méthode du Saut Minimum)**.** La méthode du saut minimum mesure la similarité entre deux clusters en prenant la distance minimale entre tous les points des deux clusters. Mathématiquement, cette mesure est définie comme la distance minimale entre les points i et j des clusters A et B :

$$\Delta(A, B) = \min_{i \in A, j \in B} d(i, j)$$

Cette mesure de similarité favorise la fusion de clusters dont les points les plus proches sont les plus similaires, ce qui peut conduire à des clusters allongés.

Definition 3.2 (Méthode de Ward)**.** La méthode de Ward cherche à minimiser la variation intra-cluster après la fusion. Elle mesure la similarité entre deux clusters en calculant la variation de la somme des carrés des distances intra-cluster avant et après la fusion. Mathématiquement, la similarité $d_{\text{Ward}}(C_1, C_2)$ entre les clusters C_1 et C_2 est définie comme :

$$d_{\text{Ward}}(C_1, C_2) = V(C) - (V(C_1) + V(C_2))$$

avec :
$V(C_1) = \sum_{i=1}^{n_1} \|x_i - m_1\|^2$ et $V(C_2) = \sum_{i=1}^{n_1} \|x_i - m_2\|^2$
où :
$V(C_i)$ représente la variation intra-cluster avant la fusion du cluster C_i
$V(C)$ représente la variation intra-cluster du nouveau cluster formé par la fusion de C_1 et C_2.
Cette mesure de similarité favorise la fusion de clusters qui contribuent le moins à la variance intra-cluster globale, favorisant ainsi la formation de clusters compacts et homogènes.

3 Avantages et Limitations :

3.1 Avantages

1. **Interprétabilité :** La CAH produit des résultats faciles à interpréter sous forme de dendrogrammes, permettant une visualisation claire de la structure hiérarchique des données.
2. **Flexibilité du nombre de clusters :** Contrairement à certains algorithmes de clustering qui nécessitent la spécification préalable du nombre de clusters, la CAH ne nécessite pas cette information, offrant ainsi une flexibilité dans le choix du nombre de clusters.
3. **Robustesse aux formes de clusters :** La CAH peut détecter des clusters de formes complexes, y compris les clusters non sphériques ou de tailles différentes, ce qui en fait une méthode robuste pour divers types de données.
4. **Utilisation de différentes métriques de distance :** La CAH peut utiliser diverses mesures de distance pour calculer la similarité entre les clusters, permettant ainsi une adaptation aux spécificités des données et des contextes d'application.

3.2 Limitations

1. **Sensibilité aux valeurs aberrantes :** Étant basée sur des mesures de distance, la CAH peut être sensible aux valeurs aberrantes, ce qui peut entraîner des distorsions dans les résultats du clustering.
2. **Complexité computationnelle :** La CAH peut être coûteuse en termes de calcul, surtout avec un grand nombre d'observations, car elle nécessite le calcul de la matrice de distance pour toutes les paires d'observations.
3. **Manque de scalabilité :** La CAH peut rencontrer des difficultés de scalabilité avec de grandes quantités de données, ce qui peut rendre son application inefficace pour de grands ensembles de données.
4. **Difficulté à traiter les données de grande dimension :** La CAH peut être moins efficace pour traiter les données de grande dimension en raison du phénomène de la malédiction de la dimensionnalité, où l'espace de recherche devient trop dispersé dans de nombreuses dimensions.

En considérant ces avantages et limitations, il est important de choisir la méthode de clustering en fonction des caractéristiques spécifiques des données et des objectifs de l'analyse.

4 Implémentation de l'algorithme (CAH) :

4.1 Implémentation de la Classe HierarchicalClustering

⚠ Le code de la classe se trouve dans la partie Annexe (Hierarchical clustering class code).

Dans cette section, nous détaillons l'implémentation de la classe `HierarchicalClustering`, qui est responsable de l'exécution de l'algorithme de clustering hiérarchique.

Attributs de la Classe

La classe `HierarchicalClustering` possède plusieurs attributs qui sont utilisés pour l'exécution de l'algorithme de clustering hiérarchique :

- `k` : Le nombre de clusters à former.
- `distances` : La matrice de distances précalculée entre les points de données.
- `labels_` : Les étiquettes de cluster attribuées à chaque point de données.

Méthodes de la Classe

La classe `HierarchicalClustering` possède plusieurs méthodes pour l'exécution de l'algorithme de clustering hiérarchique :

- `__init__` : Le constructeur de la classe qui initialise les attributs.
- `fit` : La méthode principale pour exécuter l'algorithme de clustering hiérarchique. Elle prend en entrée les données et retourne les étiquettes de cluster.
- `_calculate_distances` : Une méthode privée pour calculer les distances entre les points et les centroids.

Utilisation de la Classe

Pour utiliser la classe `HierarchicalClustering`, il suffit d'instancier un objet avec les paramètres appropriés et d'appeler la méthode `fit` avec les données à clusteriser. Voici un exemple d'utilisation de la classe :

```
# Instanciation de l'objet HierarchicalClustering
hierarchical_clustering = HierarchicalClustering(k=3)

# Execution de l'algorithme de clustering hierarchique sur les donnees
labels = hierarchical_clustering.fit(data, distances)
```

Listing 3.1 – Appel à la classe

4.2 Fonctionnalités Supplémentaires :

- **Choix de la distance et de la méthode de clustering** :
  L'utilisateur peut sélectionner la distance à utiliser pour le calcul des distances entre les points. Si la distance sélectionnée est euclidienne, la méthode de clustering utilisée est Ward. Sinon, la méthode de clustering utilisée est la moyenne (average).
- **Construction du dendrogramme** :
  Une fois la distance et la méthode de clustering déterminées, le dendrogramme est construit à l'aide de la fonction `linkage` de SciPy. Cette fonction prend les données `X` et la méthode de clustering comme entrée et renvoie une matrice qui représente les fusions de clusters successives.

— **Calcul de la ligne de coupe** :
Le seuil pour la ligne de coupe est calculé en fonction du nombre de clusters souhaité (k). Il est défini comme la distance maximale entre les $(k-1)$-ième et k-ième clusters dans le dendrogramme. En utilisant la méthode de Ward, le seuil est calculé mathématiquement comme suit :

$$\text{threshold} = Z[-k, 2]$$

où :

— Z est la matrice de liaison (*linkage matrix*) générée par la fonction `linkage` du module `scipy.cluster.hierarchy`.
— k est le nombre de clusters souhaité.

La formule ci-dessus sélectionne la distance maximale entre les fusions de clusters qui produisent exactement k clusters dans le dendrogramme.

> ⚠ **Matrice de Liaison et Calcul du Seuil**
> La matrice de liaison (*linkage matrix*) est une composante fondamentale de l'analyse de clustering hiérarchique. Elle contient des informations sur les fusions successives de clusters lors de la construction du dendrogramme. Mathématiquement, la matrice de liaison est une matrice $(n-1) \times 4$, où n est le nombre d'observations initiales. Chaque ligne de cette matrice représente une fusion de clusters avec les éléments suivants :
>
> 1. Identifiant du Cluster 1 (Index 1) : Indice du premier cluster fusionné.
> 2. Identifiant du Cluster 2 (Index 2) : Indice du deuxième cluster fusionné.
> 3. Distance ou Similarité (Index 3) : Valeur représentant la distance ou la similarité entre les clusters fusionnés. Cette valeur dépend de la méthode de linkage utilisée.
> 4. Nombre d'Observations dans le Cluster Fusionné (Index 4) : Nombre total d'observations dans le cluster fusionné.
>
> La matrice de liaison est utilisée pour construire le dendrogramme, qui est un arbre binaire représentant graphiquement les étapes de fusion des clusters. Chaque nœud du dendrogramme correspond à un cluster, et la hauteur à laquelle les deux branches d'un nœud fusionnent indique la distance ou la similarité entre ces clusters.

— **Coloration des clusters dans le dendrogramme** :
Les clusters dans le dendrogramme sont colorés en fonction de leurs étiquettes de cluster. Cela permet une visualisation claire des regroupements de données à chaque niveau de la hiérarchie.

— **Affichage des distances inter et intra-cluster** :
On calcule les distances inter-cluster, qui représentent la différence de hauteur entre les fusions de clusters consécutives dans le dendrogramme. On calcule également les distances intra-cluster, qui mesurent la cohésion des clusters en calculant la moyenne des distances de chaque point au centroid de son cluster.

Voici le code contenant toutes ces fonctionnalités combinées :

```
if clustering_result:
    X = clustering_result['X']
    labels = clustering_result['labels']
    distance_metric = st.session_state.get('distance_metric', 'euclidean').lower()
    if distance_metric == 'euclidean':
        Z = linkage(X, method='ward')
    elif distance_metric == 'city block':
        Z = linkage(X, method='average')
```

```
    else:
        st.error("Invalid distance metric selected. Choose either 'Euclidean' or 'City Block'.")
        return

    # Calculate threshold based on k
    k = len(np.unique(labels))
    if k <= 1:
        st.warning("Please choose k > 1.")
        return
    threshold = Z[-k, 2]  # Maximum distance between (k-1) and k clusters

    # Add a small offset to the threshold for better visualization
    offset = 0.05 * (max(Z[:, 2]) - min(Z[:, 2]))  # Adjust this factor as needed
    threshold += offset

    # Define color palette based on k
    if k <= 2:
        colors = ['tab:blue', 'tab:orange']
    else:
        colors = plt.cm.tab10(np.linspace(0, 1, k))

    # Plot dendrogram
    fig, ax = plt.subplots(figsize=(10, 5))
    dn = dendrogram(Z, ax=ax, color_threshold=threshold)

    # Change color of dendrogram based on labels
    for i, d in zip(dn['icoord'], dn['dcoord']):
        x = 0.5 * sum(i[1:3])
        y = d[1]
        ax.plot(x, y, color=colors[int(labels[int(i[1]-5)//10])])

    # Draw threshold line with adjusted position
    ax.axhline(y=threshold, color='r', linestyle='--', label='Threshold')

    ax.set_title("Dendrogram")
    ax.set_xlabel("Data Points")
    ax.set_ylabel("Distance")
    ax.legend()  # Show legend

    # Display the plot
    st.pyplot(fig)

    # Calculate inter-cluster distance
    inter_cluster_distances = np.diff(Z[:, 2])

    # Calculate intra-cluster distance
    intra_cluster_distances = []
    for i, d in zip(dn['icoord'], dn['dcoord']):
        x = 0.5 * sum(i[1:3])
        y = d[1]
        cluster_label = labels[int(i[1] - 5) // 10]  # Assuming dendrogram plot settings
        cluster_indices = np.where(labels == cluster_label)[0]
        cluster_points = X[cluster_indices]
        centroid = np.mean(cluster_points, axis=0)
        intra_cluster_distance = np.mean(np.linalg.norm(cluster_points - centroid, axis=1))
        intra_cluster_distances.append(intra_cluster_distance)
```

Listing 3.2 – Fonctionnalités

4 Interface graphique :

1 Interface Graphique de K-Means :

L'interface graphique de l'algorithme K-Means permet aux utilisateurs de charger un ensemble de données à partir d'un fichier Excel, de choisir le nombre de clusters (K), ainsi que la distance métrique à utiliser pour le calcul de la similarité entre les données. Elle offre également la possibilité de visualiser les résultats du clustering, y compris les centroids et les statistiques des clusters.

1.1 Page d'Accueil

La page d'accueil permet à l'utilisateur de charger un fichier Excel contenant les données à analyser. Il peut également sélectionner le nombre de clusters (K) et la distance métrique à utiliser. En cliquant sur le bouton "Run Clustering", l'algorithme K-Means est exécuté sur les données chargées.

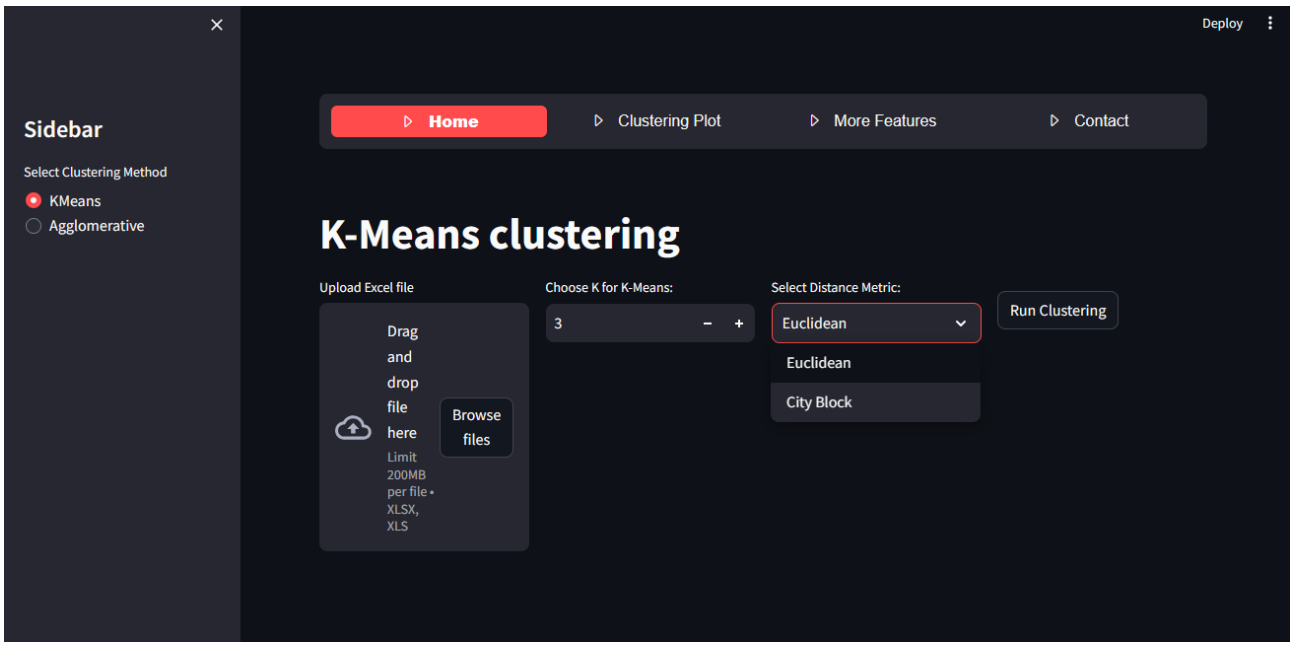

FIGURE 4.1 – Home page

1.2 Page de Visualisation du Clustering

Une fois le clustering effectué, l'utilisateur est redirigé vers la page de visualisation du clustering. Cette page affiche un graphique interactif où chaque point de données est coloré en fonction du cluster auquel il appartient. Les centroids des clusters sont également affichés. L'utilisateur peut exporter les données clusterisées au format CSV en cliquant sur le bouton "Export Data to CSV".

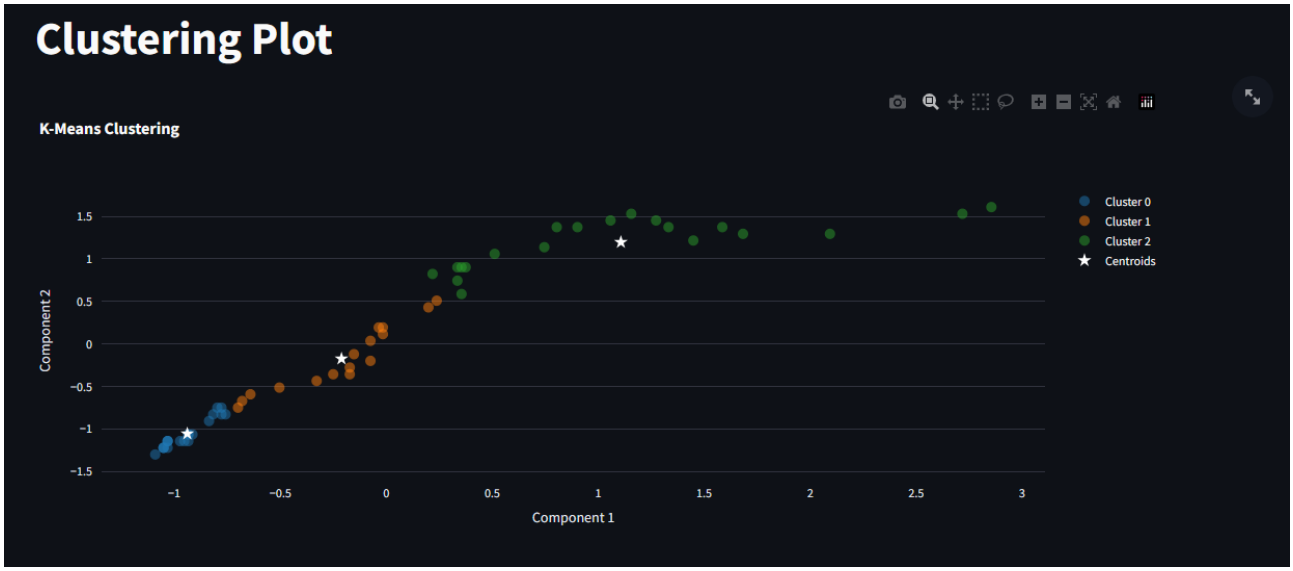

FIGURE 4.2 – Plot visualization

FIGURE 4.3 – Clusters statistics

1.3 Page de Plus de Fonctionnalités

La page de plus de fonctionnalités offre plusieurs options supplémentaires, telles que la réduction de dimensionnalité PCA, la méthode du coude, la visualisation de la convergence, et l'initialisation des centroids avec la méthode K-Means++.

Page de PCA

La page de PCA affiche un graphique de clustering après réduction de dimensionnalité à l'aide de l'analyse en composantes principales (PCA). Cela permet à l'utilisateur de visualiser les clusters dans un espace de dimension réduite.

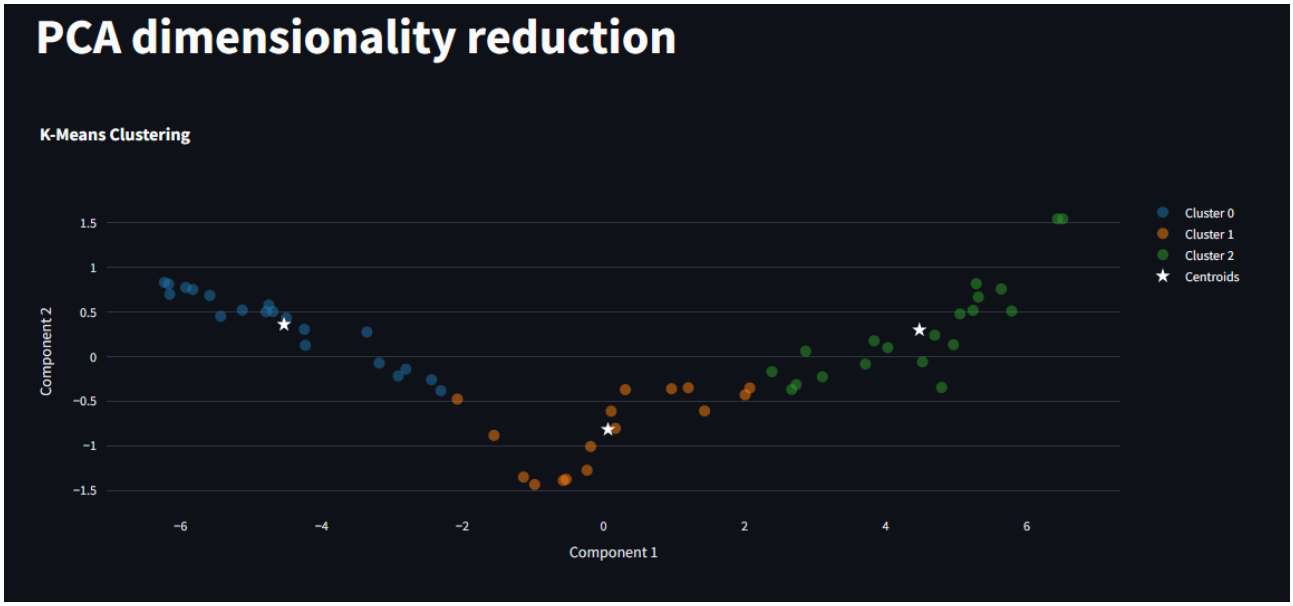

FIGURE 4.4 – PCA dimensionality reduction visualization

Page de Méthode du Coude (Elbow Method)

La page de la méthode du coude affiche un graphique de l'inertie en fonction du nombre de clusters (K). Cela permet à l'utilisateur de sélectionner un nombre approprié de clusters en identifiant le point à partir duquel l'inertie diminue de manière significative.

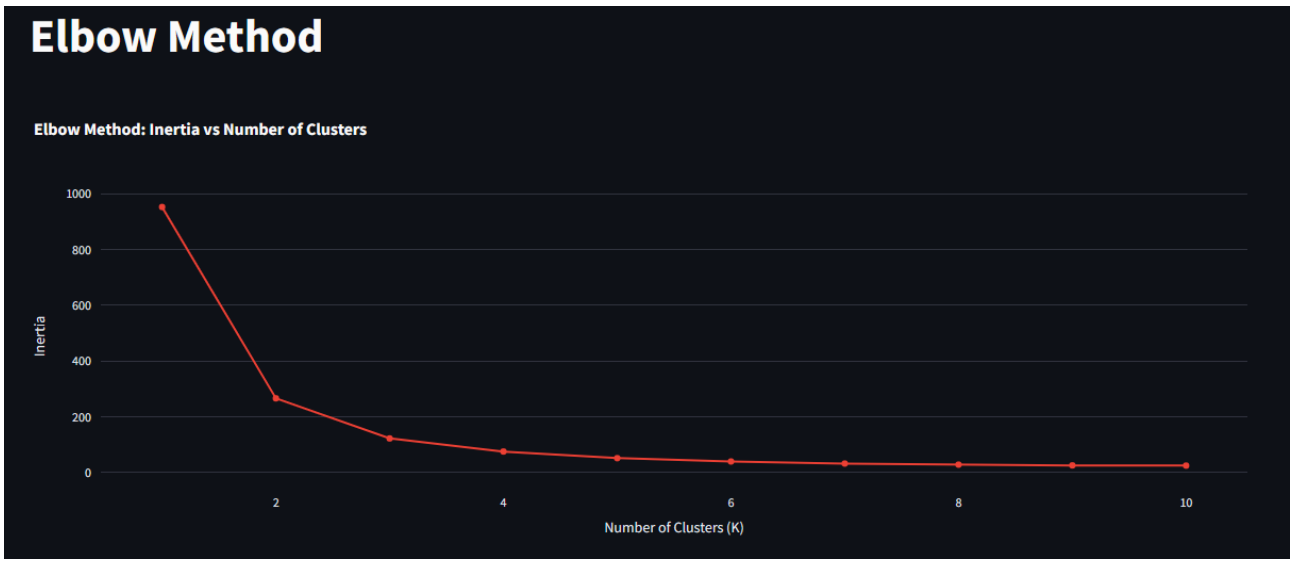

FIGURE 4.5 – Elbow Method visualization

Page de Convergence

La page de convergence affiche un graphique de l'inertie en fonction du nombre d'itérations de l'algorithme K-Means. Cela permet à l'utilisateur de vérifier si l'algorithme a convergé et à quel point.

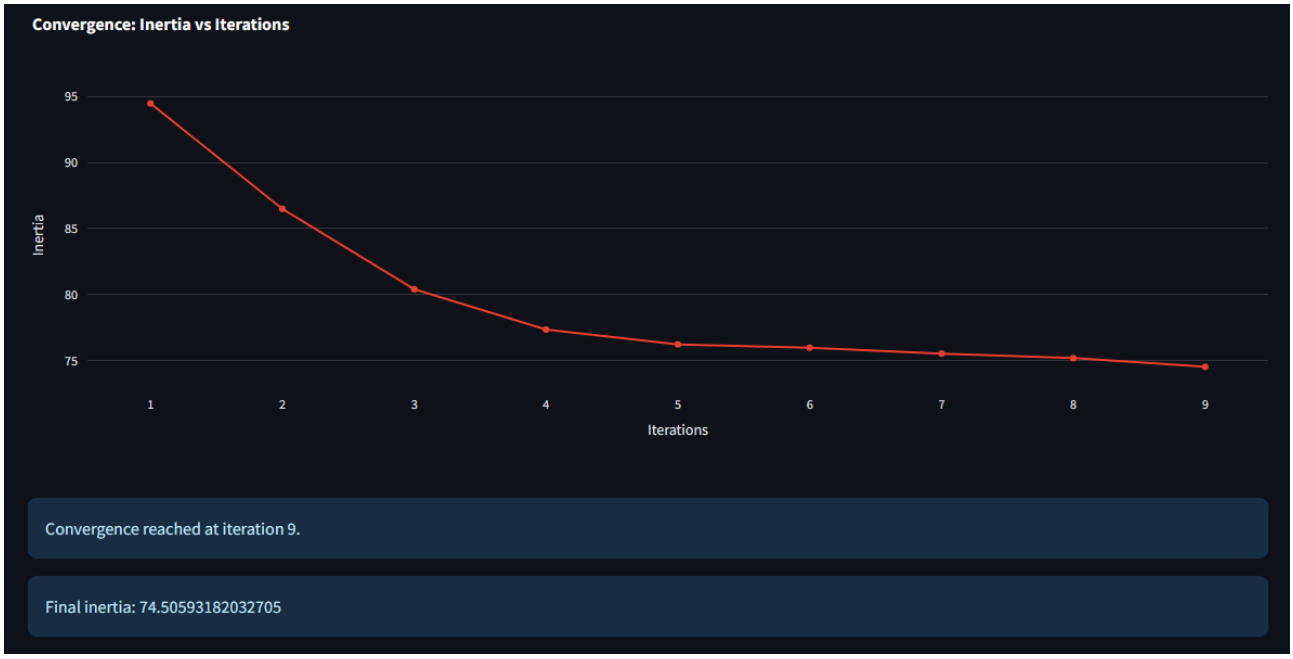

FIGURE 4.6 – Convergence visualization

1.4 Page de Contact

La page de contact fournit des liens vers le profil LinkedIn et GitHub du développeur, ainsi que vers la documentation du projet.
LinkedIn :`https://www.linkedin.com/in/wassim-kerdoun-494a89246/`
GitHub : `https://github.com/Riemann222/project.git`

2 Interface Graphique de (CAH) :

L'interface graphique propose plusieurs fonctionnalités pour l'analyse en classification ascendante hiérarchique (CAH).

2.1 Page d'accueil :

Sur cette page, l'utilisateur peut télécharger un fichier Excel contenant les données à utiliser pour l'analyse en CAH. Il peut également spécifier le nombre de clusters (k) à obtenir à partir de l'analyse à l'aide d'un champ numérique. Un menu déroulant permet à l'utilisateur de choisir entre deux métriques de distance : euclidienne ou de Manhattan. Enfin, un bouton "Run Clustering" lance l'algorithme de CAH après que l'utilisateur a saisi les paramètres requis.

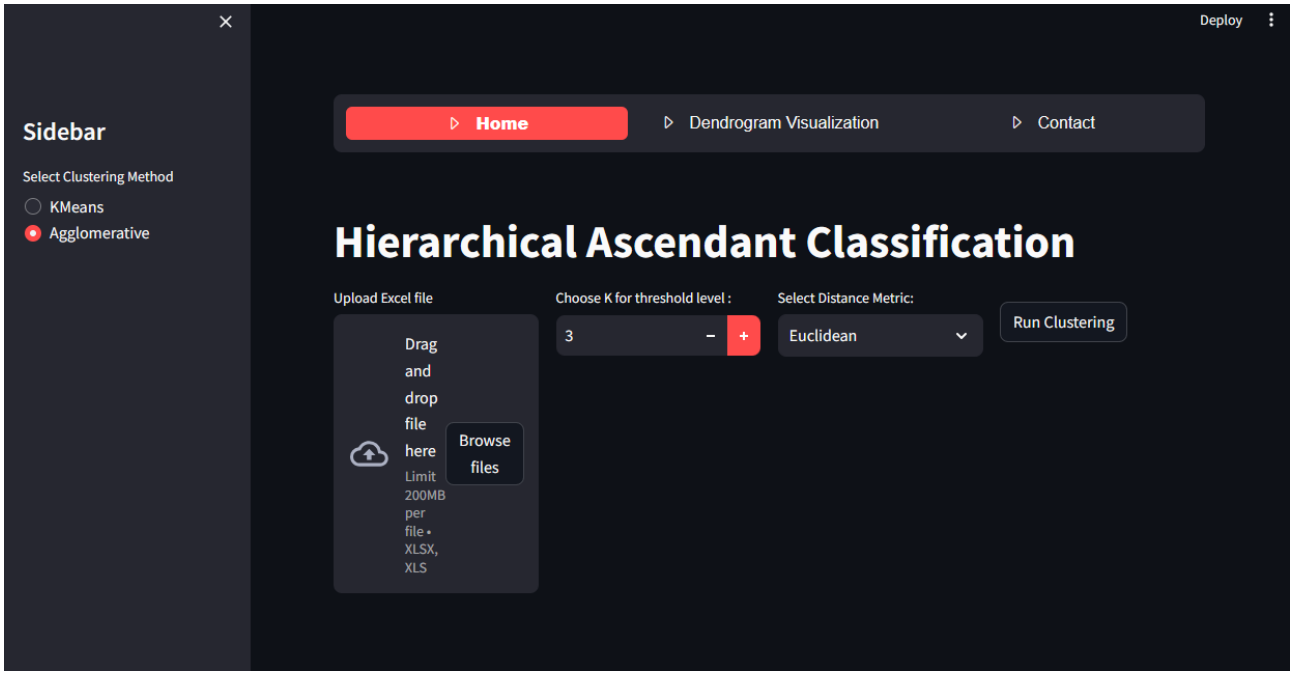

FIGURE 4.7 – Home Page

2.2 Page de visualisation du dendrogramme :

Cette page affiche le dendrogramme résultant de l'analyse en CAH. Le dendrogramme est accompagné d'une ligne de seuil indiquant la distance maximale entre les clusters. Deux tableaux horizontaux présentent également les distances inter- et intra-cluster, permettant à l'utilisateur d'examiner les propriétés des clusters obtenus.

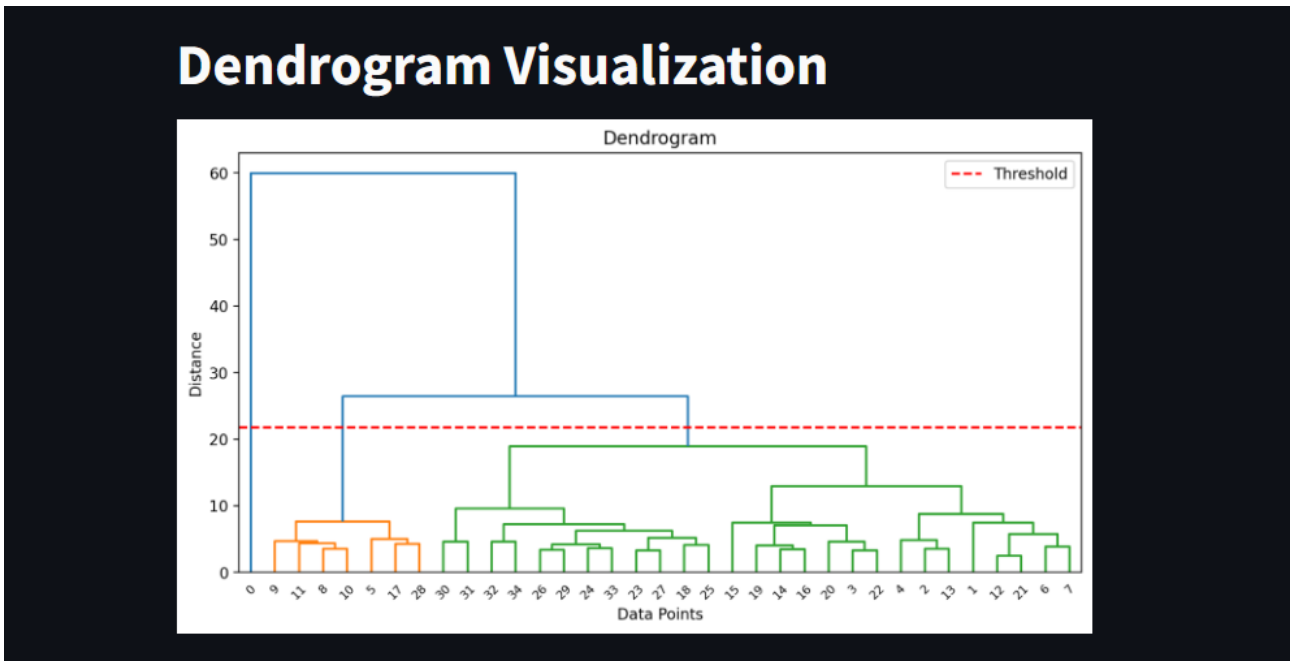

FIGURE 4.8 – Dendrogram visualization

2.3 Page de contact :

Sur cette page, l'utilisateur peut trouver des liens vers le profil LinkedIn et GitHub du développeur de l'application. Il y a également un lien vers la documentation du projet, qui peut fournir des informations supplémentaires sur son utilisation et sa mise en œuvre.
LinkedIn :`https://www.linkedin.com/in/wassim-kerdoun-494a89246/`
GitHub : `https://github.com/Riemann222/project.git`

5 Conclusion :

La classification hiérarchique ascendante (CAH) et l'analyse des k-moyennes (K-Means) sont deux techniques puissantes de clustering utilisées dans le domaine de l'apprentissage non supervisé. La CAH offre une approche ascendante qui permet de regrouper les données de manière hiérarchique, offrant ainsi une vision en profondeur de la structure des données grâce aux dendrogrammes. En revanche, les K-Means sont efficaces pour diviser les données en clusters compacts en minimisant l'inertie intra-cluster, mais ils nécessitent la spécification préalable du nombre de clusters. L'implémentation pratique de ces algorithmes dans une application web interactive offre aux utilisateurs une plateforme conviviale pour explorer et comprendre leurs données. L'utilisation de bibliothèques comme NumPy et SciPy facilite le traitement des données en Python, tandis que l'intégration de fonctionnalités supplémentaires telles que la réduction de dimensionnalité PCA et la méthode du coude permet aux utilisateurs d'affiner leurs analyses. Malgré leurs avantages, ces techniques restent sensibles aux valeurs aberrantes et à la sélection des paramètres, ce qui souligne l'importance d'une compréhension approfondie des données et des algorithmes. En envisageant des améliorations futures, une documentation exhaustive et l'intégration de techniques de clustering avancées pourraient enrichir davantage l'expérience utilisateur et étendre les possibilités d'exploration des données.

6 Annexe :

Code de la classe `KMeansClustering` avec la méthode d'initialisation intelligentes des centroids utilisées uniquement au choix de l'utilisateur :

```
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
        self.inertia_history = []

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

        self.centroids = self._initialize_centroids(X)  # Initialize centroids using
    k-means++

        for iteration in range(max_iterations):
            # Assign each data point to the nearest centroid
            distances = self.calculate_distances(X)
            cluster_assignment = np.argmin(distances, axis=0)

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

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

            self.centroids = new_centroids

            # Calculate inertia and store it in history
            inertia = np.sum(np.min(distances, axis=0))
            self.inertia_history.append(inertia)

        return cluster_assignment, self.centroids, self.inertia_history

    def calculate_distances(self, X):
        if self.distance_metric == 'euclidean':
            return np.vstack([np.linalg.norm(X - centroid, axis=1) for centroid in
    self.centroids])
        elif self.distance_metric == 'city_block':
            return np.vstack([np.sum(np.abs(centroid - X), axis=1) for centroid in
    self.centroids])
        else:
            raise ValueError("Invalid distance metric specified.")

    def _initialize_centroids(self, X):
        # Randomly select the first centroid
        centroids = [X[np.random.randint(X.shape[0])]]

        # Select the remaining centroids using k-means++ initialization
        for _ in range(1, self.k):
            # Calculate squared distances from each data point to the nearest
    centroid
```

```
            distances = np.array([min([np.linalg.norm(x - c)**2 for c in centroids])
    for x in X])
            # Choose new centroid from data points with probability proportional to
    squared distance
            probabilities = distances / distances.sum()
            centroids.append(X[np.random.choice(len(X), p=probabilities)])

        return np.array(centroids)
```

Listing 6.1 – Kmeans class code

Code de la classe `HierarchicalClustering` :

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

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

        # Hierarchical clustering algorithm
        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 6.2 – Hierarchical clustering class code