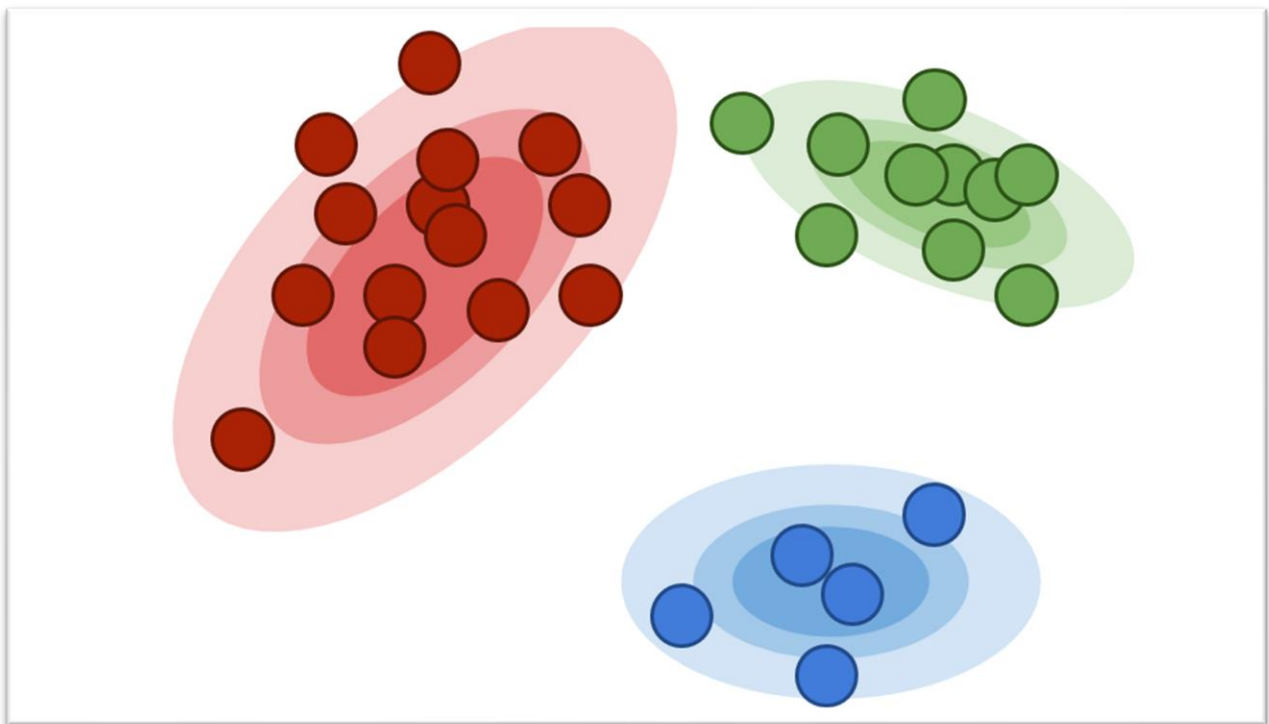


# Analyse de données : TP1

Le clustering



## Table des matières

1 - Initiation à Python et à l'usage des principales librairies.....	3
1.1 Tests élémentaires sous Python.....	3
1.1.1 Création du corpus de test .....	3
1.2 Méthode de K-means .....	4
1.2.1 Test de la Méthode de K-means .....	4
1.2.1 Choix du « bon » nombre de clusters .....	5
1.3 Clustering ascendant hiérarchique.....	7
2 - Application dans un contexte de données réelles .....	11
2.1 - Utilisation de la méthode K-Means pour la réduction de couleur .....	11
2.2 Clustering de données de températures .....	13
2.2.1 Classification hiérarchique ascendante .....	13
2.3 Méthode des K-means.....	15
2.4 Comparaison des classifications.....	16
3 Réalisation algorithmique .....	18

# 1 - Initiation à Python et à l'usage des principales librairies

## 1.1 - Tests élémentaires sous Python

### 1.1.1 - Création du corpus de test

Voici l'affichage de deux classes : classe 1 avec les points bleus et classe 2 avec les croix rouges.

#### Pour la classe 1 :

Simulation de  $n = 128$  données  $X = (X_1, X_2)$  suivant une loi normale bidimensionnelle de moyenne  $(2, 2)$ , de la variance  $\text{Var}(X_1) = 2$ ,  $\text{Var}(X_2) = 2$  et la covariance  $\text{Cov}(X_1, X_2) = 0$

#### Pour la classe 2 :

Simulation de  $n = 128$  données  $X = (X_1, X_2)$  suivant une loi normale bidimensionnelle de moyenne  $(-4, -4)$ , de la variance  $\text{Var}(X_1) = 6$ ,  $\text{Var}(X_2) = 6$  et la covariance  $\text{Cov}(X_1, X_2) = 0$

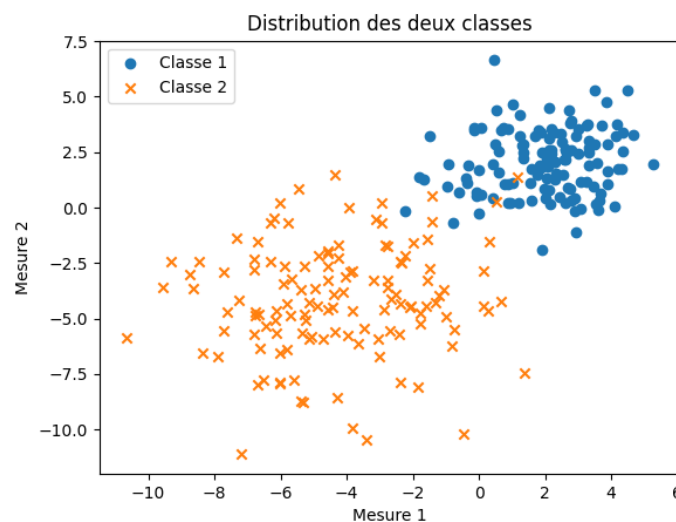


Figure 1 : Affichage de la Classe 1 et de la Classe 2

Code dans le fichier : **1\_Initiation\_à\_Python\_et\_à\_l'usage\_des\_principales\_librairies.py**

- Section : **1.1.1 Création du corpus de test**

On adapte le code pour afficher les deux classes. On met les bons paramètres pour chaque classe et ensuite on les affiche comme l'exemple nous le montre (Résultat : Figure 1).

## 1.2 - Méthode de K-means

### 1.2.1 - Test de la Méthode de K-means

Ici on utilise Kmeans pour avoir les deux centroïdes du graphique précédent. Les points sont ensuite mis dans une classe selon s'ils sont plus proche d'une centroïde ou de l'autre. Bien sûr il y a le même nombre de points dans chaque classe.

Comme on peut le voir, les croix jaune et violette représente deux groupes(cluster).

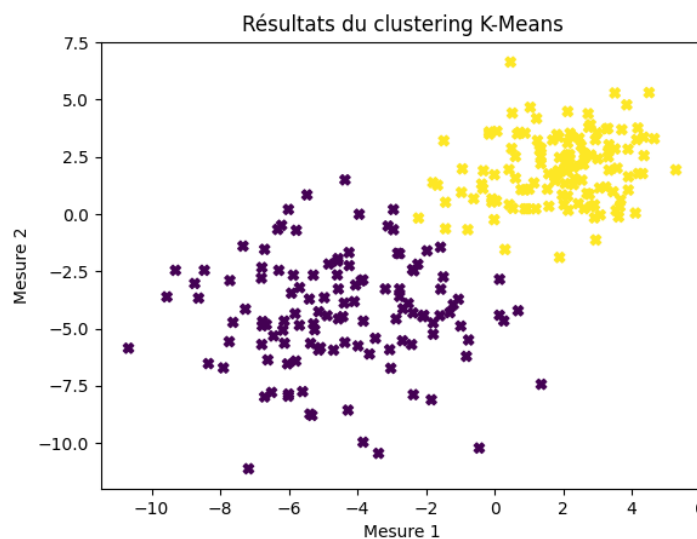


Figure 2 : Clustering de K-Means

Le code suivant permet de générer ces centroïdes et de mettre les points dans les bonnes classes. Aussi nous affichons le tableau avec les points associés avec leur groupe(cluster) par un nombre (ici 0 et 1 comme nous avons deux cluster).

```
k = 2
kmeans = KMeans(n_clusters=k, init='k-means++', n_init=10)
```

Selon la documentation scikit-learn : [sklearn.cluster.KMeans](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html)

- **n\_clusters** est le nombre de clusters et le nombre de centroïdes à générer, ici on en 2.
- **init** est la méthode d'initialisation des centroïdes, ici nous utilisons la méthode k-mean++. k-mean++ utilise une distribution empirique de probabilité pour accélérer la convergence.
- **n\_init** est le nombre de fois où l'on applique la fonction k-mean avec des seeds différentes et on prend le meilleur résultat. La valeur par défaut est 10. Plus le nombre est haut, plus on augmente les chances d'obtenir la meilleure solution en termes d'inertie, tout en minimisant le risque de rester bloqué dans un minimum local. Mais bien sûr plus nous augmentons ce nombre alors plus le temps de calcul sera important.

```
kmeans.fit(X=data)
```

Selon la documentation scikit-learn : [sklearn.cluster.KMeans.fit](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.fit.html)

**kmeans.fit** permet d'avoir les groupes optimaux en ajustant les positions des centroïdes.

- **X** sont les données sur lesquelles nous voulons appliquer l'algorithme K-means pour les regrouper en k groupes.

```
print("Affectations des clusters :\n", kmeans.labels_)
```

Selon la documentation scikit-learn : [sklearn.cluster.KMeans](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html)

- **kmeans.labels\_** est le label pour chaque point du graphe. Si k = 2 alors on aura des labels de 0 à 1.

Le code suivant permet de calculer l'erreur entre la classification obtenue avec les centroïde et la classification « vraie » qui est juste le regroupement des groupes avant l'utilisation de KMeans. Ici comme nous avons les classes obtenu grâce à KMeans plutôt similaire aux classes de base alors l'indice de similarité est haut (environ 0.9) comme nous pouvions nous y attendre.

```
rand_score = adjusted_rand_score(true_labels, labels_pred=kmeans.labels_)
```

Selon la documentation scikit-learn : [sklearn.metrics.adjusted\\_rand\\_score](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html)

- **labels\_true** est la classe dite vraie utiliser comme référence.
- **labels\_pred** est la classe à évaluer.

### 1.2.1 - Choix du « bon » nombre de clusters

Une silhouette permet de savoir la similarité d'un point avec son propre cluster par rapport aux autres clusters. Cette valeur varie entre -1 et 1, plus la valeur est élevée, plus le point est adapté à son cluster et mal adapté aux autres clusters.

Le coefficient de silhouette est la moyenne des silhouettes d'un ensemble de données. Plus un coefficient de silhouette est élevé, plus le clustering est approprié.

L'inertie permet de savoir les clusters sont plus ou moins compacts. Plus l'inertie est faible, plus les clusters sont compacts et mieux définis.

Pour savoir quel est le meilleur K pour le clustering, nous devons prendre le plus grand coefficient de silhouette et la plus petite inertie.

Voyons nos résultats pour le coefficient de silhouette (Figure 3) et l'inertie (Figure 4) :

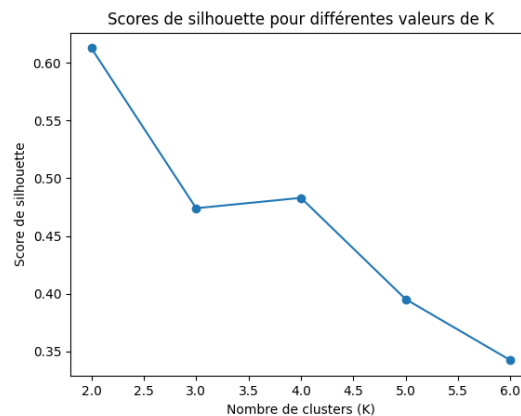


Figure 4 : Coefficient de silhouette

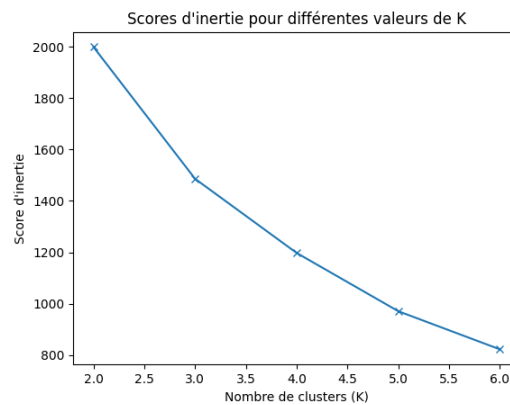


Figure 3 : Inertie

Nous pouvons voir que pour le cluster  $K=4$ , le coefficient de silhouette est supérieur au cluster  $K=3$  et l'inertie de  $K=4$  est inférieure à l'inertie de  $K=3$  donc il semble que le meilleur paramètre soit  $K=4$ .

Nous n'avons pas utilisé la méthode du coude vu qu'ici, notre courbe d'inertie ne nous permet pas d'appliquer cette méthode.

Cette méthode nous aurait permis de voir le bon nombre de cluster à prendre quand l'inertie commence à stagner.

### 1.3 - Clustering ascendant hiérarchique

Le clustering ascendant hiérarchique nous permet de faire le dendrogramme pour afficher comment les points se regroupent en plus gros clusters pour ensuite avoir les trois clusters attendu (pour  $K=3$ ).

On représente la hiérarchie des clusters avec la fonction `linkage` :

```
Z_complete = linkage(data, method='complete', metric='euclidean')
```

Selon la documentation scipy : [scipy.cluster.hierarchy.linkage](https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.linkage.html)

- **data** est notre tableau plat (2D) de données
- **method** est la méthode qui permet de regrouper les clusters. Ici 'complete' mesure la distance maximale entre les deux clusters.
- **metric** permet de spécifier la mesure de distance qui permet de calculer la similarité et la dissimilarité entre les clusters. Ici nous utilisons la mesure 'euclidean' qui est la mesure par défaut qui utilise la distance entre les points.

```
seuil_auto = Z_complete[:, 2][256-k] - 1.e-10
```

Pour le `seuil_auto` nous prenons le tableau de la 3<sup>ème</sup> colonne de 'Z\_complete' qui correspond aux distances entre les clusters fusionnés à chaque étape du processus de CAH et ensuite nous prenons le nombre qui est inférieur à l'avant dernier nombre du tableau (ici pour  $K=3$ ) ce qui nous permet de prendre deux nombres qui sont :

Pour le premier : la distance entre les deux premiers clusters.

Pour le deuxième : la distance entre le regroupement de ces deux clusters et le troisième cluster.

```
dendrogram(Z_complete, color_threshold=seuil_auto)
```

Avec la fonction 'dendrogram' permet de faire le dendrogramme avec seulement nos 3 clusters que nous voulons représenter avec 3 couleurs (pour  $K=3$ ).

Selon la documentation scipy : [scipy.cluster.hierarchy.dendrogram](https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.dendrogram.html)

- **Z\_complete** sont les distances entre les clusters fusionnés
- **color\_threshold** permet de colorier de la même couleur les clusters fusionnés qui se trouvent eux-mêmes dans un cluster supérieur à la valeur de `color_threshold`.

```
plt.axhline(y=seuil_auto, c='grey', lw=1, linestyle='dashed')  
plt.show()
```

Et finalement nous affichons le dendrogramme avec le seuil.

Voici le dendrogramme que nous avons réussie à afficher :

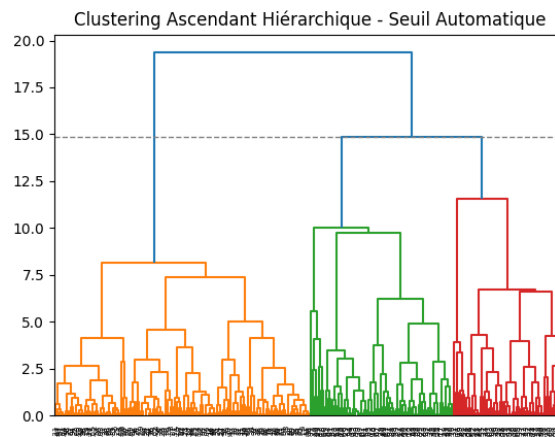


Figure 5 : Dendrogramme avec seuil automatique

Nous avons bien nous trois couleurs pour K=3 ainsi que le seuil au niveau des trois clusters.

Pour avoir le résultat du clustering en 3 groupes (pour k=3), on utilise les résultats du linkage pour faire 3 groupes comme pour le dendrogramme et on fait un graphe avec les points colorier selon leurs appartenances à leur groupe.

Nous utilisons fcluster qui permet de transformer notre 'Z\_complexe' en un cluster plat et on utilise le résultat de cette fonction pour mettre les 3 couleurs pour les clusters sur notre graphique.

```
groupes_cah_auto = fcluster(Z_complexe, t=seuil_auto, criterion='distance')
```

Selon la documentation scipy : [scipy.cluster.hierarchy.fcluster](https://docs.scipy.org/doc/scipy/reference/cluster/hierarchy/fcluster.html)

Fcluster permet former des clusters plats à partir d'un clustering hiérarchique.

- **Z\_complexe** est le résultat de notre clustering hiérarchique.
- **t** est la valeur à partir de laquelle les clusters sont formés.
- **criterion** est spécification du critère à utiliser pour la formation des clusters et détermine comment la distance entre les clusters est évaluée. Ici nous utilisons 'distance' qui utilise la distance directe entre les clusters.



```
plt.scatter(data[:, 0], data[:, 1], c=groupe_cah_auto, cmap='viridis')
plt.title("Clustering Ascendant Hiérarchique - Seuil Automatique")
plt.show()
```

Selon la documentation matplotlib : [matplotlib.pyplot.scatter](https://matplotlib.org/3.1.1/pyplot-api/#matplotlib.pyplot.scatter)

Nous prenons les coordonnées x et y du résultat de notre clustering hiérarchique, nous prenons le tableau des points avec leur affectation à leur cluster (donc 0, 1 ou 2).

- **X** représente coordonnées en x.
- **Y** représente coordonnées en y.
- **c** représente les points associés à leurs clusters.
- **cmap** est la coloration des points.

Et donc voici nos résultats pour l'affichage du graphique :

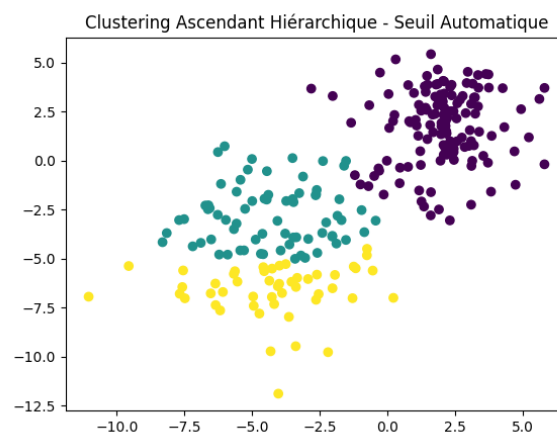


Figure 6 : Graphique à l'aide de groupes\_cah

Voici les résultats que nous obtenons en utilisant les quatre méthodes de clustering 'single', 'complete', 'average' et 'ward' :

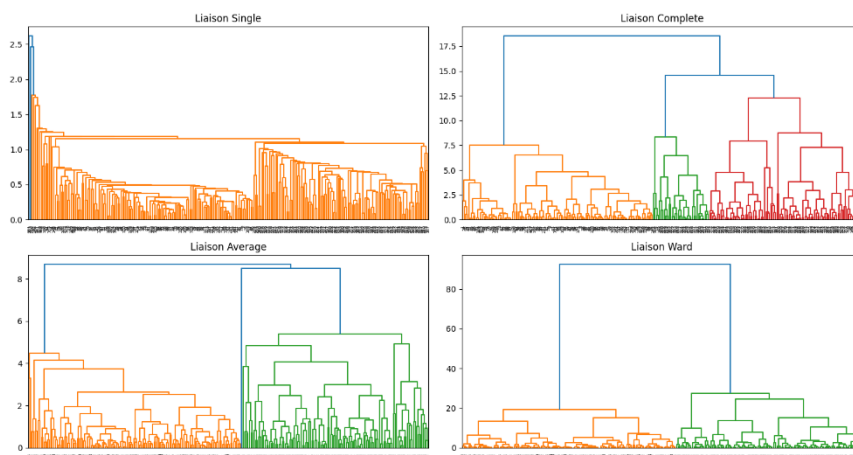


Figure 7 Dendrogramme avec 4 méthodes de clustering

Voici les résultats que nous obtenons pour les scores de rand ajusté :

- Méthode '**single**' : 0.00027
- Méthode '**complete**' : 0.78
- Méthode '**average**' : 0.65
- Méthode '**ward**' : 0.64

Le meilleur score que nous pouvons avoir est 1. Donc la meilleure méthode possible est celle qui s'en rapproche le plus et donc ici la meilleure méthode est '**complete**'. On peut notamment le voir sur les différents graphiques, la méthode '**single**' est anarchique, les méthodes '**average**' et '**ward**' sont en très grande partie composées de 2 clusters et la méthode '**complete**' a bien les 3 clusters que nous attendions.

Bien sûr, d'une itération à l'autre les résultats ne sont pas forcément les mêmes mais il semble que '**complete**' ait toujours le plus gros score avec en second '**average**' et '**ward**' et '**single**' est toujours loin derrière.

En comparaison à l'algorithme K-Means, ces méthodes ne sont pas aussi efficaces, K-Means a des résultats entre 0.8 et 0.9 ce qui se rapproche beaucoup plus de 1 que les autres méthodes. Aussi le paramètre `n_init` nous permet de faire plusieurs fois le calcul et de prendre le meilleur résultat donc cela nous permet d'avoir des scores plus hauts pour K-Means.

## 2 - Application dans un contexte de données réelles

### 2.1 - Utilisation de la méthode K-Means pour la réduction de couleur

Pour l'utilisation de K-Means pour la réduction des couleurs d'une image, nous avons fait les tests pour 5 K différents. Nos K sont : 2, 5, 15, 20, 50.

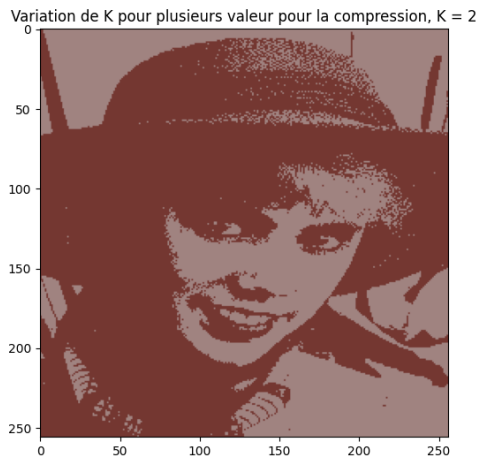


Figure 11 : Réduction d'image pour K = 2

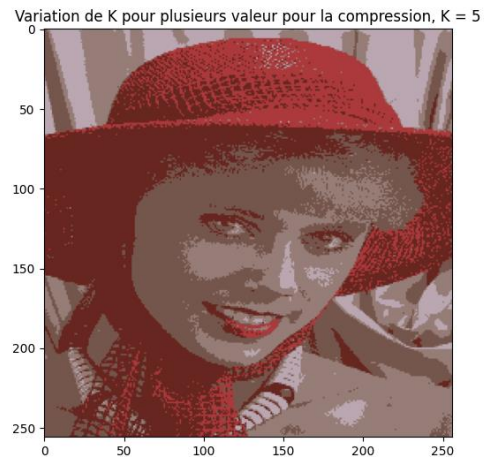


Figure 10 : Réduction d'image pour K = 5

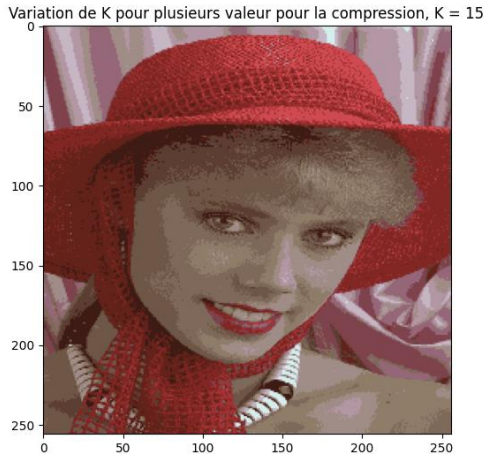


Figure 9 : Réduction d'image pour K = 15

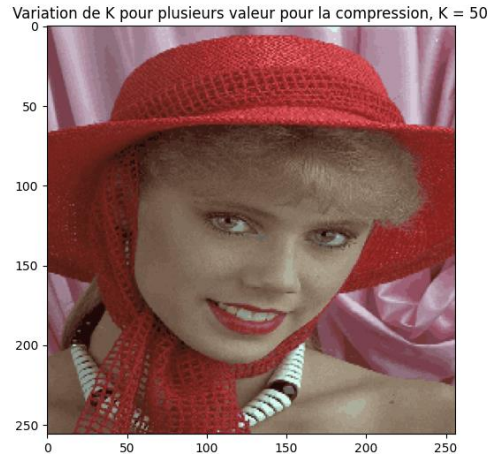


Figure 8 : Réduction d'image pour K = 50

Comme nous pouvons le voir sur les images, la valeur de K représente le nombre de couleurs qu'il y aura dans l'image donc plus le K est grand plus l'image est fidèle à l'original.

Nous pouvons voir que pour K = 50, l'image est casi identique à celle de base.

Explication : Pour faire cette réduction nous avons donc utilisé l'algorithme K-Means. Nous avons transformé les pixels de l'image en points avec une couleur associé, ensuite suivant le K (nombre de clusters), nous associons les points proches entre eux à un cluster pour ensuite faire la moyenne des couleurs qui sont dans ce clusters, ce qui nous permet

finalement de donner une même couleur à l'ensemble des points du cluster. Et comme les points les plus proches entre eux ont une couleur plutôt similaire (quand nous avons un K au-dessus environ de 50), la réduction n'affecte pas la qualité de l'image à l'œil nu.



*Figure 12 : Image d'origine*

## 2.2 - Clustering de données de températures

### 2.2.1 - Classification hiérarchique ascendante

```
data_scaled = scale(data)
```

Selon la documentation sklearn : [sklearn.preprocessing.scale](https://scikit-learn.org/stable/modules/preprocessing.html)

**Scale** permet de normaliser un ensemble de données en soustrayant la moyenne et en divisant par l'écart type.

- **data** est la matrice (ou le tableau) des données à normaliser.

Pour le dendrogramme nous avons choisi de faire ressortir 3 classes et d'utiliser la méthode 'complete'. Une pour les pays froids, une autre pour les pays tempérés et la dernière pour les pays chauds.

Voici le dendrogramme que nous obtenons pour les températures dans les différentes villes :

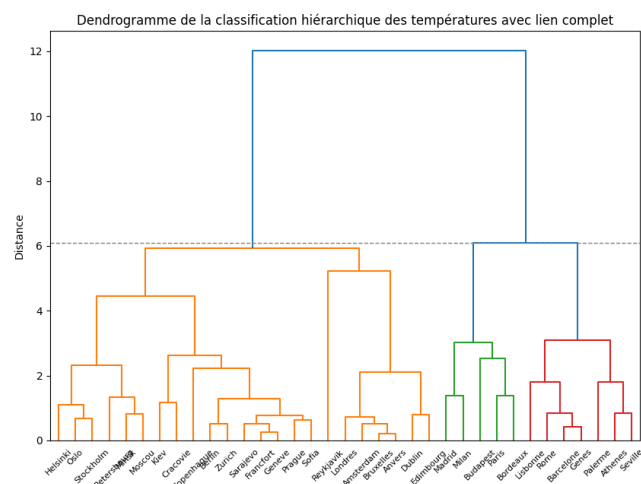


Figure 13 : Dendrogramme sur les températures des villes en 3 classes avec la méthode 'complete'

Notre dendrogramme est cohérent, nous avons toutes les villes 'froides' ensemble (au nombre de 23), toutes les villes tempérées ensemble (au nombre de 5) et toutes les villes chaudes ensemble (au nombre de 7). Les résultats correspondent avec la situation géographique de chaque ville. Les villes froides sont bien le plus au nord, les villes tempérées sont au centre et les villes chaudes sont au sud de la carte.

Pour une autre mesure de la dissimilarité entre classe, nous avons choisi la méthode 'ward'.

Voici nos résultats pour le dendrogramme :

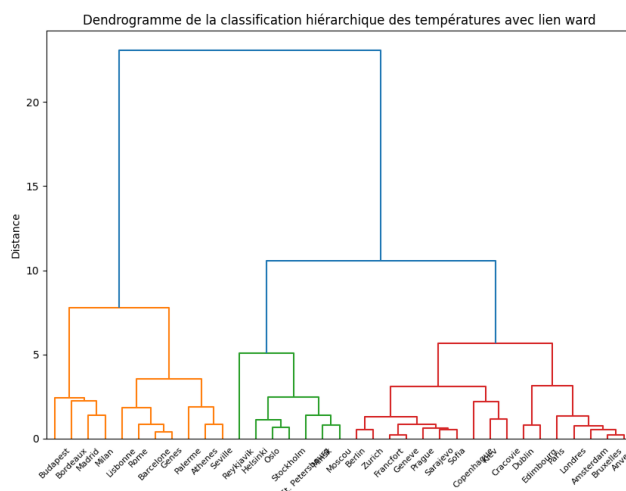


Figure 14 : Dendrogramme sur les températures des villes en 3 classes avec la méthode 'ward'

Et voici nos résultats pour la représentation graphique des villes selon leurs coordonnées géographiques avec en bleu les villes 'froides', en jaune les villes 'tempérées' et en violet les villes 'chaudes' :

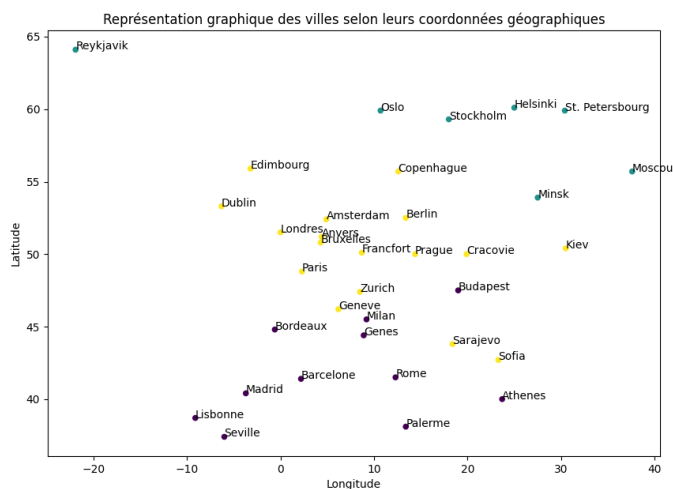


Figure 15 : Représentation graphique des villes selon leurs coordonnées géographiques

Cette fois-ci nous pouvons voir que la répartition entre les clusters est différente. Nous avons les villes chaudes en bas de la carte comme attendu (au nombre de 11), les villes tempérées sont au centre de la carte (au nombre de 17) et les villes froides sont tout en haut de la carte (au nombre de 7).

## 2.3 - Méthode des K-means

Avec la méthode K-Means nous obtenons le graphique suivant :

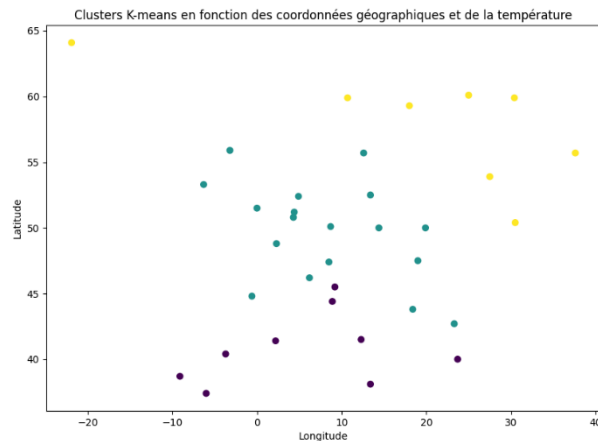


Figure 16 : Graphique avec l'utilisation de K-Means

Nous pouvons voir qu'il y a des petites différences entre le graphique fait avec K-Means et le graphique fait avec la hiérarchie ascendante mais globalement nous avons les pays chauds en bas et les pays froids en haut.

Voici notre courbe d'inertie en fonction du nombre de clusters :

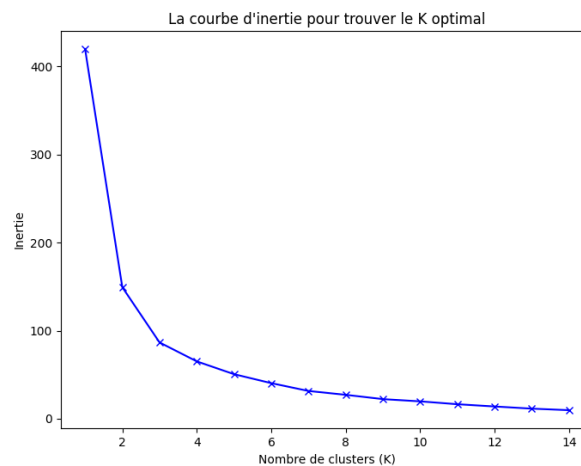


Figure 17: Courbe d'inertie pour la méthode K-Means

Avec la courbe nous pouvons voir que l'inertie commence à se stabiliser après  $K = 3$ , donc en utilisant la méthode du coude, le bon nombre de cluster semble être 3.

Pour la méthode K-Means, nous avons une répartition comme pour la hiérarchie ascendante en 3 groupes : froids, tempérés et chauds. Cette fois-ci le nombre de villes dans le cluster des villes froides est de 8, pour les villes tempérées, il y en a 18 et pour les pays chauds il y en a 9.

## 2.4 - Comparaison des classifications

Comme nous pouvons le voir la méthode K-Means et la classification hiérarchie ascendante avec l'utilisation de la méthode 'ward' on des résultats plutôt similaires avec la moitié des villes en climat tempéré et une répartition des villes froides et chaudes plutôt équitable. C'est ce que l'on pouvait attendre comme résultat avec l'hypothèse de base qui était une répartition en 3 climats (froid, tempéré et chaud).

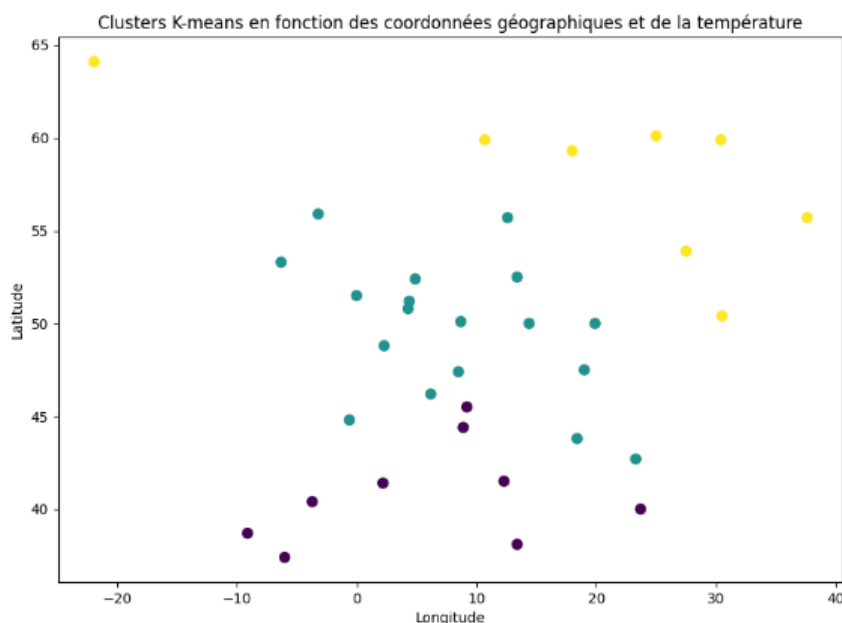
En utilisant la classification hiérarchie ascendante avec l'utilisation de la méthode 'complete' nous avons un résultat différent, nous avons plus de la moitié des villes qui sont dans le groupe des villes froides et nous avons peu de villes avec le climat tempéré mais il semble que le nombre de villes chaudes est semblable au résultat des autres méthodes.

Nous pouvons en conclure que pour notre l'hypothèse avec 3 climats, l'utilisation de K-Means et la méthode 'ward' nous donne les meilleurs résultats et que la méthode 'complete' ne nous permet pas de séparer correctement les points pour cette l'hypothèse.

Nos résultats obtenus :




### K-Means :

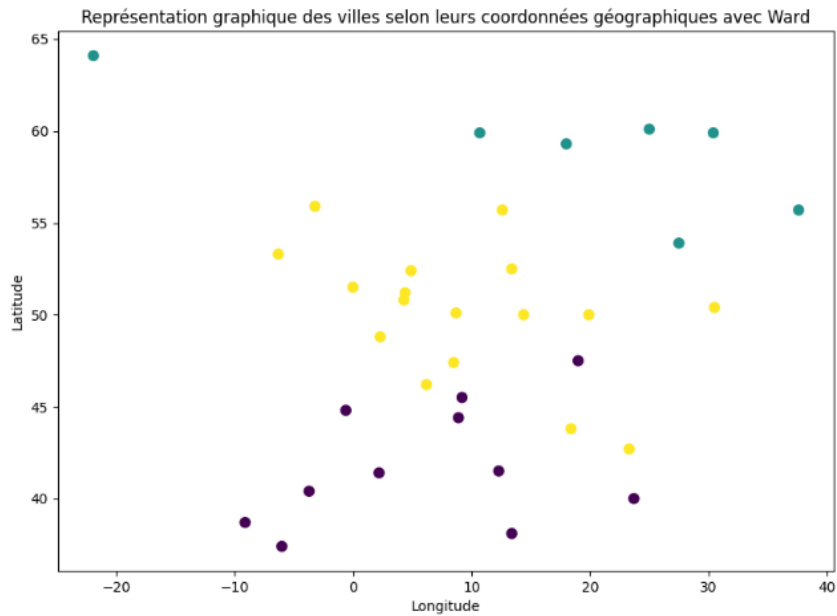
<i>Villes froides</i>	<i>Villes tempérées</i>	<i>Villes chaudes</i>
8 ●	18 ●	9 ●








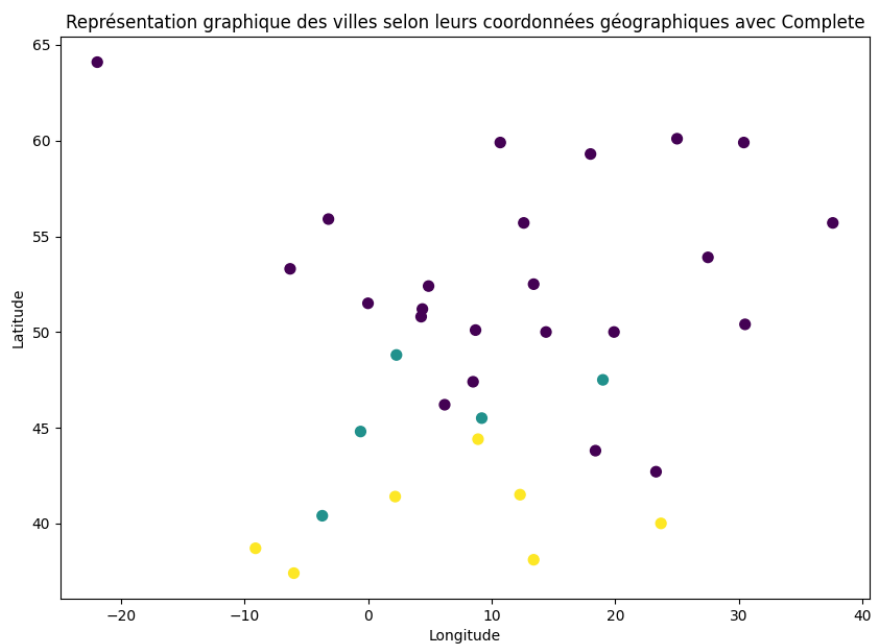
### Méthode Ward :

<i>Villes froides</i>	<i>Villes tempérées</i>	<i>Villes chaudes</i>
11 	17 	7 



### Méthode Complete :

<i>Villes froides</i>	<i>Villes tempérées</i>	<i>Villes chaudes</i>
23 	5 	7 



### 3 - Réalisation algorithmique

Voici notre fonction 'perform\_coalescence', pour faire l'algorithme de coalescence :

```
cluster_labels, updated_centroids = perform_coalescence(data_points, num_clusters, initial_centroids)
```

Les arguments :

- `data_points` : est l'ensemble des points à regrouper.
- `num_clusters` : est le nombre de cluster à calculer.
- `initial_centroids` : est le tableau des centroïdes de base.

Les résultats (sous forme de tuple) :

- `cluster_labels` : est le tableau des labels des points associées à leur groupes une fois le calcul fini.
- `updated_centroids` : est le tableau des centroïdes une fois le calcul fini.

#### Explication de l'algorithme :

L'algorithme de coalescence est une méthode de clustering qui vise à regrouper des points en clusters en minimisant la distance entre chaque point et le centroïde de son cluster. L'algorithme commence par une initialisation aléatoire des centroïdes (ici nous donnons à notre fonction de calcul les centroïdes aléatoires, donc pas besoin de faire le calcul), puis attribue chaque point au cluster dont le centroïde est le plus proche. Ensuite, les centroïdes sont mis à jour en fonction des points qui leur sont attribués. Ce processus est répété jusqu'à ce que les centroïdes ne changent plus (ou que le nombre maximum d'itérations soit atteint, ce qui n'est pas le cas de notre fonction).

#### Nos résultats :

En partant de notre nuage de points initial et avec nos deux centroïdes initiaux, nous pouvons voir qu'après l'utilisation de l'algorithme de coalescence, nous obtenons bien deux clusters avant en leurs centre les nouvelles centroïdes que nous venons de calculer avec l'algorithme. Les résultats semblent cohérents par rapport aux clusters et au centroïdes.

Pour nous assurer que notre algorithme fonctionner correctement, nous avons appliqué l'algorithme de K-Means en prenant les mêmes points initiaux et les même centroïdes initiaux également. En comparant nos deux résultats, pour l'algorithme de coalescence et celui de K-Means, nous pouvons voir que les résultats sont similaires, nous avons les centroïdes au même endroit et pour les points dans les clusters, nous avons quelques petites différences.

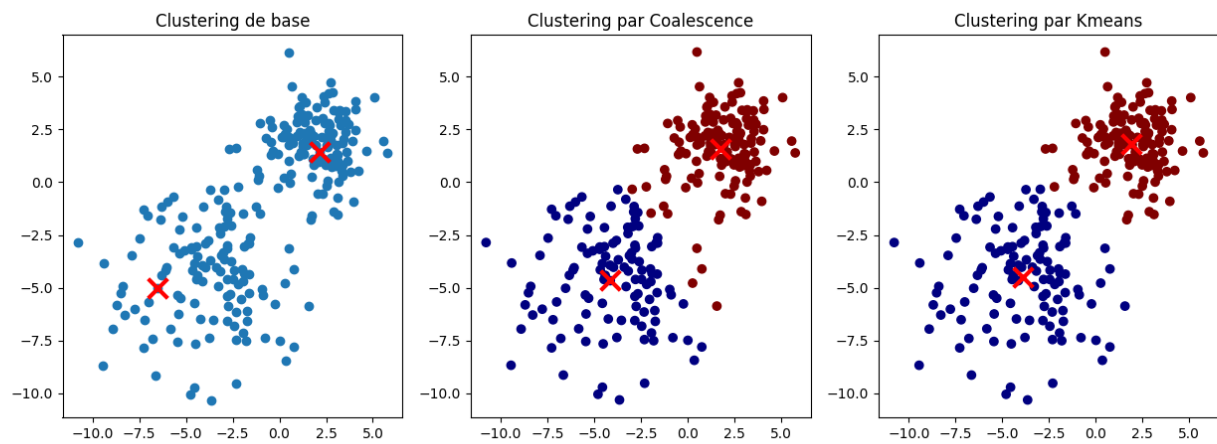


Figure 18 : Graphiques pour le clustering coalescence et K-Means

Nous aurions dû mesurer la convergence de notre algorithme en regardant à quel moment l'algorithme arrêterait de changer les positions des centroïdes. Pour cela, nous aurions dû rajouter un nombre d'itérations maximum dans notre fonction afin de pouvoir comparer les résultats pour des nombres d'itérations différents.