Narjisse LASRI Estelle DUHEM

Compte rendu du Projet ODATA

Analyse de données pour guider les choix d'affectation de l'aide humanitaire internationale

1 - Objectifs du projet et livrables attendus

Ce projet a pour objectif d'effectuer une classification non supervisée des principaux pays du monde en matière de développement à partir de données socioéconomiques et de santé sur chaque pays, afin de guider les choix d'affectation de l'aide humanitaire en identifiant les pays prioritairement dans le besoin.

Nous allons tester 4 méthodes de clustering différentes : K-means, la Classification hiérarchique ascendante, le Modèle de mélange de Gaussiennes et DBSCAN.

2 - Etude préalable : Comparaison des méthodes de clustering sur des données simulées

2.1 Etude de la méthode DBSCAN

La méthode DBSCAN est une méthode de clustering par densité. Elle permet de trouver des clusters qui ont une forme géométrique autre que celles qui peuvent être identifiées par la méthode K-means et la classification hiérarchique ascendante, comme par exemple des cercles imbriqués. Cette méthode se base sur le fait que 2 points appartiennent au même cluster s'il est possible de créer un chemin pour passer de l'un à l'autre de proche en proche en restant à l'intérieur du même cluster.
Cette méthode réalise une itération sur les points du jeu de données. Pour chacun des points analysés, DBSCAN construit l'ensemble des points atteignables par densité depuis ce point. Il calcule l'epsilon-voisinage de ce point, epsilon étant la distance maximale entre deux points pour que l'un soit considéré comme le voisinage de l'autre. Si ce voisinage contient plus de n_min points, il calcule ensuite les epsilon-voisinages de chacun de ces points, et ainsi de suite jusqu'à ce que le cluster ne s'agrandisse plus. Si le point analysé a un voisinage contenant moins de n_min points, il est considéré comme du bruit car il n'a pas suffisamment de voisins. L'un des avantages de DBSCAN est donc d'être robuste aux données aberrantes car ce mécanisme permet de les isoler.
L'avantage majeur de cette méthode est qu'elle permet d'identifier des clusters de forme arbitraire, impossible à identifier avec d'autres méthodes de clustering. Elle permet également d'identifier des clusters de formes et de tailles différentes dans un même jeu de données. De plus, cette méthode ne nécessite pas de préciser le nombre de clusters désirés car elle les identifie automatiquement.

2.2 Etude des classes et modules relatifs aux différentes méthodes

- ➔ Méthode K-means : sklearn.cluster.KMeans
  - ◆ Paramètres d'appel
    - n_clusters : Le nombre de clusters à former
    - init : Méthode d'initialisation, qui peut être :
      - 'k-means++' : choisit les centres des clusters d'une façon intelligente pour accélérer la convergence → valeur par défaut
      - 'random' : choisit n_clusters observations (lignes) aléatoirement dans les données comme centres des clusters
      - un objet de type ndarray, de taille n_clusters x nombre de variables, qui donne les centres initiaux
      - un callable, prenant comme arguments X, n_clusters et un état aléatoire, et retournant une initialisation
    - n_init : Le nombre de fois où l'algorithme k-means sera exécuté avec différents centres initiaux. Le résultat final sera le meilleur résultat des n_init exécutions consécutives en termes d'inertie.
    - max_iter : Nombre d'itérations maximal de l'algorithme k-means pour une seule exécution
    - tol : Tolérance relative par rapport à la norme de Frobenius entre les centres de clusters de deux itérations consécutives pour déclarer la convergence.
    - Et d'autres paramètres plus poussés que nous n'utiliserons pas
  - ◆ Attributs
    - cluster_centers_ : Coordonnées des centres de clusters. Si l'algorithme s'arrête avant la convergence complète (voir tol et max_iter), ces coordonnées ne seront pas cohérentes avec les labels_
    - labels_ : Étiquettes de chaque point
    - inertia_ : Somme des carrés des distances entre les échantillons et leur centre de cluster le plus proche (inertie intra-classe)
    - n_iter : nombre d'itérations
  - ◆ Méthodes
    - fit(X[, y, sample_weight]) : Effectue le clustering par k-means
    - fit_predict(X[, y, sample_weight]) : Calcule les centres de clusters et prédit l'indice de cluster pour chaque échantillon
    - fit_transform(X[, y, sample_weight]) : Effectue le clustering et transforme X en un espace de clusters basés sur les distances
    - get_params([deep]) : Donne les paramètres pour cet estimateur
    - predict(X[, sample_weight]) : Prédit le cluster le plus proche auquel appartient chaque échantillon dans X
    - score(X[, y, sample_weight]) : Opposé à la valeur de X sur l'objectif de K-means (inertie inter-classe)
    - set_params(**params) : Définit les paramètres de cet estimateur
    - transform(X) : Transforme X en un espace de clusters basés sur les distances

- ➔ Classification hiérarchique ascendante : scipy.cluster.hierarchy
  - ◆ Fonctions
    - fcluster(Z, t[, criterion, depth, R, monocrit]) : Forme des clusters plats à partir du clustering hiérarchique défini par la matrice de liaison donnée
    - fclusterdata(X, t[, criterion, metric, …]) : Données d'observation des clusters en utilisant une métrique donnée
    - leaders(Z, T) : Retourne les noeuds racine dans un clustering hiérarchique
    - linkage(y[, method, metric, optimal_ordering]) : Effectue le clustering hiérarchique/agglomératif
    - Plusieurs fonctions effectuent une liaison selon différentes méthodes sur la matrice de distance condensée y : single(y), complete(y), average(y), etc…
    - Plusieurs fonctions calculent des statistiques sur les hiérarchies : cophenet(Z[, Y]), from_mlab_linkage(Z), inconsistent(Z[, d]), etc...
    - dendrogram(Z[, p, truncate_mode, …]) : Trace le clustering hiérarchique comme un dendrogramme
    - set_link_color_palette(palette) : Établit la liste des codes de couleur de matplotlib à utiliser pour le dendrogramme.
    - Il y a également des structures de données et des fonctions permettant de représenter les hiérarchies sous forme d'objets arborescents : ClusterNode(id[, left, right, dist, count]), to_tree(Z[, rd]), etc…
    - Ces indicateurs permettent de vérifier la validité des matrices de liens et d'incohérences ainsi que l'isomorphisme de deux affectations à des clusters plats : is_valid_im(R[, warning, throw, name]), is_isomorphic(T1, T2), correspond(Z, Y)

- ➔ Modèle de mélange de Gaussiennes : sklearn.mixture.GaussianMixture
  - ◆ Paramètres d'appel
    - n_components : Le nombre de composants du mélange
    - covariance_type : type de paramètres de covariance à utiliser
      - ○ 'full' : chaque composant a sa propre matrice de covariance générale, les composants peuvent adopter indépendamment toute position et forme
      - ○ 'tied' : tous les composants partagent la même matrice de covariance générale, les composants ont la même forme
      - ○ 'diag' : chaque composant a sa propre matrice de covariance diagonale, les axes de contour sont orientés le long des axes de coordonnées, situation "diagonale" avec des contours circulaires
      - ○ 'spherical' : chaque composant a sa propre variance unique
    - tol : seuil de convergence
    - reg_covar : Permet de s'assurer que les matrices de covariance sont toutes positives
    - max_iter : Le nombre d'itérations EM à effectuer (EM est l'algorithme espérance-maximisation qui permet de trouver les paramètres du maximum de vraisemblance d'un modèle probabiliste)
    - n_init : Le nombre d'initialisations à effectuer. Les meilleurs résultats sont conservés.
    - init_params{'kmeans', 'random'}, defaults to 'kmeans' :
    - reg_covar : La méthode utilisée pour initialiser les poids, les moyennes et les précisions
    - weights_init : Les poids initiaux fournis par l'utilisateur
    - means_init : Les moyennes initiales fournis par l'utilisateur

- precisions_init : Les précisions initiales fournies par l'utilisateur (inverses des matrices de covariance)
- random_state {int, RandomState instance or None}, optional (default=None): Contrôle les données aléatoires de la méthode choisie pour initialiser les paramètres
- warm_start : Si 'warm_start' vaut True, la solution du dernier ajustement est utilisée comme initialisation pour le prochain appel de fit (). Cela peut accélérer la convergence lorsque l'ajustement est appelé plusieurs fois sur des problèmes similaires.
- verbose : Si 1, imprime l'initialisation en cours et chaque étape d'itération. Si elle est supérieure à 1, affiche également la probabilité logarithmique et le temps nécessaire pour chaque étape.
- verbose_interval : Nombre d'itérations effectuées avant la prochaine impression

◆ Attributs
- weights_ : Les poids de chaque composant du mélange
- means_ : La moyenne de chaque composant du mélange
- covariances_ : La covariance de chaque composant du mélange
- precisions_ : Les matrices de précision pour chaque composant du mélange
- precisions_cholesky_ : La décomposition cholesky des matrices de précision de chaque composant du mélange
- converged_ : True lorsque la convergence a été atteinte dans fit (), False sinon
- n_iter_: Nombre de pas utilisé par le meilleur ajustement de EM pour atteindre la convergence
- lower_bound_ : Valeur limite inférieure de la log-vraisemblance du meilleur ajustement de EM.

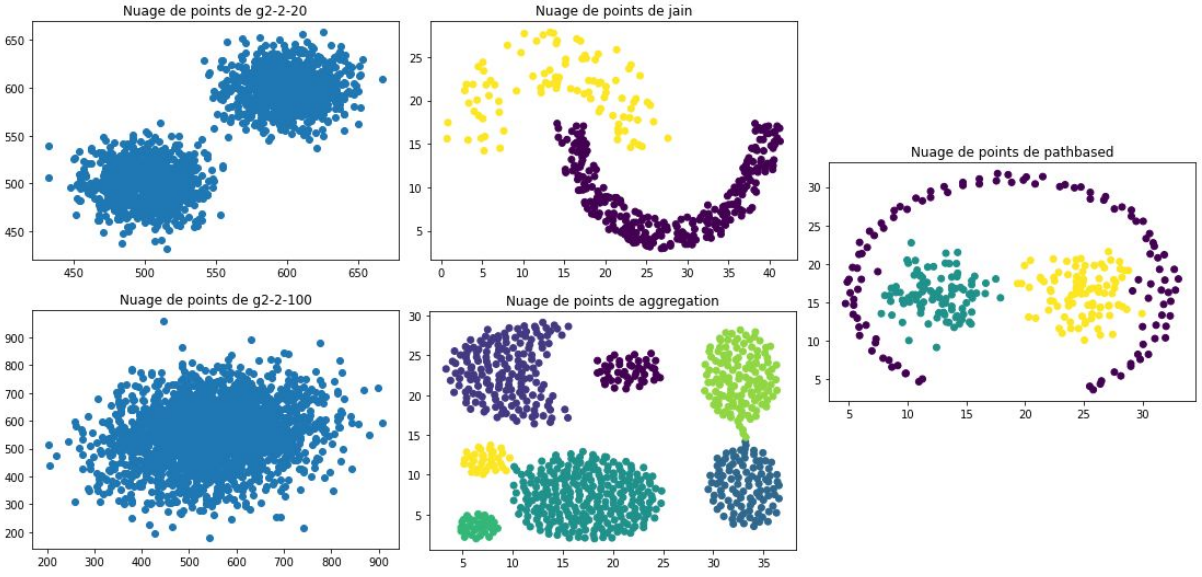
◆ Méthodes
- aic(X) : donne le critère d'information Akaike pour le modèle actuel sur l'entrée X
- bic(X) : donne le critère d'information bayésien pour le modèle actuel sur l'entrée X
- fit(X [, y]) : estime les paramètres du modèle avec l'algorithme EM
- fit_predict(X [, y]) : estime les paramètres du modèle et prédit les étiquettes pour X
- get_params([deep]) : obtient les paramètres de l’estimateur
- predict(X) : prédit la probabilité a posteriori de chaque composant compte tenu des données
- predict_proba(X)
- sample([n_samples]) : génère des échantillons aléatoires à partir de la distribution gaussienne ajustée
- score(X[, y]) : calcule la log-vraisemblance moyenne par échantillon des données X
- score_samples(X) : calcule les probabilités log pondérées pour chaque échantillon
- set_params(**params) : définit les paramètres de l’estimateur

➔ DBSCAN : sklearn.cluster.DBSCAN

- ◆ Paramètres d'appel
  - eps : La distance maximale entre deux points pour que l'un soit considéré comme le voisinage de l'autre
  - min_samples : Le nombre minimal de points dans le voisinage d'un point central.
  - metric (string, or callable, default='euclidean'): La métrique à utiliser lors du calcul de la distance entre les points.
  - metric_params : Arguments de mot clé supplémentaires pour la fonction métrique.
  - algorithm {'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto' : L'algorithme à utiliser pour calculer les distances ponctuelles et trouver les voisins les plus proches
  - leaf_size : nombre de points regroupés dans chaque nœud feuille dans le KDTree
- ◆ Attributs
  - core_sample_indices_ : Indices des clusters
  - components_ : Copie de chaque cluster trouvé par l'entraînement des données
  - labels_ : Étiquettes des clusters pour chaque point donné à fit ()
- ◆ Méthodes
  - fit(X[, y, sample_weight]) : permet d'effectuer une mise en cluster DBSCAN à partir de points ou d'une matrice de distance.
  - fit_predict(X[, y, sample_weight]) : permet d'effectuer une mise en cluster DBSCAN sur X et de renvoyer les étiquettes des clusters.
  - get_params([deep]) : permet d'obtenir les paramètres de l'estimateur.
  - set_params(**params) : permet de définir les paramètres de l'estimateur.

2.3 Expérimentations

❖ Visualisation des nuages de points pour chaque fichier de données :



- ➢ Données de g2-2-20.txt : Les différents clusters ne sont pas indiqués dans une troisième colonne de nos données, cependant nous pouvons clairement identifier 2 clusters de forme elliptique.
- ➢ Données de g2-2-100.txt : Il nous est impossible d'identifier des clusters dans ces données, on ne discerne qu'un seul et unique cluster avec éventuellement quelques données éloignées qui pourraient être jugées aberrantes.
- ➢ Données de jain.txt : La troisième colonne de notre fichier de données nous permet de colorer les points en fonction de leur cluster, nous identifions donc ici clairement deux clusters de forme arbitraire.
- ➢ Données de Aggregation.txt : Grâce à la troisième colonne de notre fichier de données, nous identifions 7 clusters, tous plus ou moins de forme elliptique, mais qui ne nous semblent pas si faciles à identifier du fait du manque de séparation nette entre certains clusters.
- ➢ Données de pathbased.txt : Nous identifions ici 3 clusters, dont un qui "encercle" les deux autres clusters, cette disposition particulière rendra certainement l'identification de ce cluster plus difficile.

❖ Choix des paramètres pour chaque méthode et chaque fichiers de données :

• Méthode K-means :

Pour tous les fichiers, nous laissons la valeur par défaut pour le paramètre 'init' qui est 'k-means++', car nous estimons que c'est la méthode d'initialisation la plus performante.

- ➢ sur g2-2-20.txt : Suite à la visualisation du nuage de points de ces données, nous avons identifié 2 clusters qui semblent aisément séparables par la méthode k-means. Lors de l'appel du constructeur KMeans de sklearn.cluster, nous définissons donc le paramètre 'n_clusters' à 2, afin que 2 clusters soient formés par cette méthode. Nous définissons le paramètre 'n_init' à 2 car nous pensons qu'il ne sera pas nécessaire de réaliser beaucoup d'initialisations différentes afin d'obtenir un bon clustering.
- ➢ sur g2-2-100.txt : Pour ces données, nous pensons qu'il n'y qu'un seul cluster à identifier, or il est inutile de lancer la méthode K-means pour former un unique cluster. C'est pourquoi nous décidons d'utiliser ces données pour observer le comportement des différentes méthodes sur des données que l'on ne peut pas partitionner. Pour les 4 méthodes à évaluer, nous chercherons donc à faire 4 clusters sur les données du fichier g2-2-100.txt, afin d'observer les formes des clusters que ces différentes méthodes produisent automatiquement.
- ➢ sur jain.txt : Suite à la visualisation du nuage de points de ces données, nous avons identifié 2 clusters mais ceux-ci ne semblent pas aisément identifiables par la méthode k-means du fait de leurs formes arbitraires. C'est pourquoi nous décidons de mettre 'n_clusters' à 2 et n_init à 10, nous espérons qu'avec 10 initialisations différentes nous obtiendrons un résultat meilleur sur ces données plus difficilement séparables.

- ➢ sur Aggregation.txt : De même, nous avions identifié 7 clusters pour ces données, nous définissons donc 'n_clusters' à 7. Concernant n_init, du fait du grand nombre de clusters et du manque de frontières bien définies avec des trous importants entre ceux-ci, nous pensons qu'il faut un nombre d'initialisations plus grand pour obtenir un résultat plus correct. Nous passons donc n_init à 20 pour commencer, nous avons ensuite tenté d'augmenter cette valeur mais sans constater d'amélioration notable.
- ➢ sur pathbased.txt : De même, nous définissons 'n_clusters' à 3 suite à la visualisation du nuage de points. Pour le nombre d'initialisations 'n_init', nous commençons par le mettre à 5 car nous ne pensons pas que la méthode k-means sera capable d'identifier le troisième cluster qui "entoure" les deux autres. En augmentant 'n_init', nous ne constatons aucune amélioration des performances de la méthode k-means.

- **Classification hiérarchique ascendante (CHA) :**

Pour tous les fichiers, dans la fonction *scipy.cluster.hierarchy.linkage(y, method='single', metric='euclidean', optimal_ordering=False)* nous laissons le paramètre 'metric' sur sa valeur par défaut 'euclidean' de façon à ce que ça soit la distance euclidienne qui soit utilisée pour la distance entre points. De même, pour tous les fichiers, nous définissons le paramètre 'criterion' sur 'maxclust' dans la fonction *scipy.cluster.hierarchy.fcluster(Z, t, criterion='inconsistent', depth=2, R=None, monocrit=None)* de façon à pouvoir directement choisir le nombre maximum de clusters souhaités avec le paramètre 't'. Seuls les paramètres 'method' (pour la distance entre clusters) et 't' (nombre de clusters souhaités) changeront d'un fichier à un autre.

- ➢ sur g2-2-20.txt : On ne peut pas définir le paramètre 'method' sur la méthode 'single' car elle permet de séparer des clusters de formes non elliptiques à condition que le "trou" entre ces 2 clusters soit suffisamment grand, ce qui n'est pas le cas d'après notre visualisation du nuage de points. Nous avons un peu de bruit entre nos 2 clusters qui fausse les résultats. Nous définissons donc le paramètre 'method' sur méthode 'complete' car celle-ci est performante pour séparer les clusters même s'il y a du bruit entre ceux-ci.
- ➢ sur jain.txt : Comme dit précédemment, la méthode 'single' permet de séparer des clusters de formes non elliptiques à condition que le "trou" entre ces deux clusters soit suffisamment grand. Or d'après notre visualisation du nuage de points du fichier jain.txt, les conditions sont remplies donc la méthode 'single' devrait avoir de bonnes performances sur ces données. C'est pourquoi nous décidons de définir le paramètre 'method' sur 'single'.
- ➢ sur Aggregation.txt : Nous avons du bruit entre nos différents clusters d'après la visualisation du nuage de points, c'est pourquoi nous définissons ici 'method' sur 'complete'.
- ➢ sur pathbased.txt : La méthode 'complete' a tendance à "casser" les clusters larges, or d'après la visualisation du nuage de points, les 2 clusters centraux sont plutôt larges. Du fait du bruit entre ces 2 clusters, nous ne pouvons pas non plus utiliser la méthode 'single'. Nous avons le choix entre les méthodes 'average', 'centroid' et 'ward', nous utilisons la méthode 'ward' car c'est la plus populaire. Nous avons ensuite calculé l'indice de Rand ajusté pour les 3 méthodes et c'est bien la méthode 'ward' qui a les meilleures performances sur ce fichier.

- ➢ sur g2-2-100.txt : Nous décidons d'appliquer la méthode 'complete' à ces données, car c'est celle que nous avons utilisée le plus, de ce fait nous allons pouvoir observer son comportement sur des données que l'on ne peut pas partitionner.

- **Modèle de mélange de Gaussiennes :**

Pour tous les fichiers, nous laissons les paramètres par défaut notamment le type de covariance ('full') : chaque composant a sa propre matrice de covariance général ce qui permet aux clusters de s'adapter à toutes les formes; et la méthode pour initialiser les poids, les moyennes et la précision : K-means qui permet d'identifier en peu d'itérations les clusters.
Seul le paramètre n_components changera en fonction des fichiers et le paramètre n_init qui sera fixé à 3 pour le fichier aggregation.

- ➢ sur g2-2-20.txt : Lors de l'analyse visuelle du nuage de points, nous avons identifié 2 clusters. Nous fixons donc n_components à 2.
- ➢ sur g2-2-100.txt : Comme pour la méthode K-means, nous décidons d'utiliser ces données pour observer le comportement des différentes méthodes sur des données que l'on ne peut pas partitionner.
- ➢ sur jain.txt : Nous définissons n_components à 2 pour identifier les 2 clusters du nuage de points.
- ➢ sur Aggregation.txt : Nous remarquons que les clusters du fichier Aggregation varient en fonction des initialisations et n'est pas stable. Nous décidons donc de fixer le paramètre n_init à 3 pour tester plusieurs initialisations et prendre la meilleure. Le modèle est ainsi plus précis et plus stable. Nous fixons également n_components à 7 pour trouver 7 clusters.
- ➢ sur pathbased.txt : 3 clusters sont apparents sur le nuage de points, n_components est donc égal à 3.

- **DBSCAN :**

Nous choisissons de paramétrer seulement eps et min_samples dans la fonction *sklearn.cluster.DBSCAN*. La distance euclidienne est donc choisie pour calculer la distance entre deux points et l'algorithme pour trouver le plus proche voisin sera trouvé automatiquement.
Pour le paramètre eps, nous créons une fonction permettant de trouver le epsilon optimal. Cette fonction calcule pour chaque point la distance à ses deux plus proches voisins, puis trie et trace les résultats. La valeur optimale pour epsilon sera trouvée au point de courbure maximale. Nous ajustons les espilons trouvés en faisant des tests visuels sur les fichiers.
Pour le paramètre min_samples, nous partons de la valeur par défaut 5 et nous l'ajustons grâce à des tests visuels sur les fichiers.

- ➢ sur g2-2-20.txt : Nous déterminons epsilon à 5 en utilisant la fonction `epsilon()` puis, après quelques tests, nous le changeons à 10. Nous gardons min_samples à 5.
- ➢ sur g2-2-100.txt : Selon le graphique des résultats, le epsilon optimal se trouve à 20. Après quelques tests, nous le passons à 30. Nous déterminons également de même min_sample et le fixons à 10.

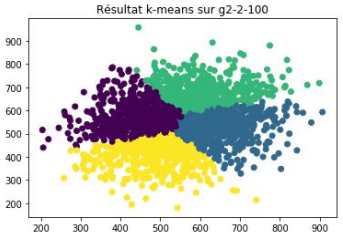
- ➢ sur jain.txt : Nous déterminons epsilon à 1 et nous le changeons ensuite à 2.7 après des tests. Nous choisissons min_sample égal à 3.
- ➢ sur Aggregation.txt : Nous trouvons le epsilon optimal, selon le graphique, égal à 0.9, nous l'augmentons à 1.5 et nous gardons n_samples à 5.
- ➢ sur pathbased.txt : Nous trouvons grâce à la fonction un epsilon égal à 1.5. Après quelques tests sur le fichier pathbased, nous choisissons epsilon égale à 2. Nous faisons également des tests pour définir le min_samples et nous obtenons 10 comme valeur optimale.

❖ Evaluation des performances des 4 méthodes :

• Méthode K-means :

Lorsque nous évaluons qualitativement / visuellement la méthode k-means (avec la représentation du nuage de points colorés selon les différents clusters produits) :

- Les performances de cette méthode sont très bonnes sur les données du fichier g2-2-20.txt, même avec peu d'initialisations différentes, car les clusters sont aisément identifiables. En effet, nous pouvons voir sur la représentation du nuage de points que les 2 clusters sont de forme elliptique, avec un large trou entre ceux-ci.
- Sur les clusters de formes arbitraires qui semblent constituer un tracé continu, comme dans les données des fichiers jain.txt et pathbased.txt, les performances de la méthode k-means sont limitées. Il est difficile pour la méthode de trouver des centres de clusters du fait de leurs formes particulières.
- Sur les données du fichier Aggregation.txt, malgré les formes elliptiques des 7 clusters, les clusters identifiés ne sont pas tous corrects. Nous observons ici que la méthode k-means voit ses performances limitées lorsque les clusters sont trop reliés les uns aux autres.
- Notons également que sur les données du fichier g2-2-100.txt, la méthode k-means a divisé le nuage des points en 4 clusters semblables à des quarts de diagramme de taille égale. La division se fait selon des axes diagonaux par rapport aux axes de coordonnées.



Lorsque nous évaluons quantitativement la méthode k-means (avec l'indice de Rand ajusté calculé par la fonction *adjusted_rand_score()* du module sklearn.metrics) :

- Nous ne pouvons faire cette évaluation quantitative que sur les fichiers de données jain.txt, Aggregation.txt et pathbased.txt, car ce sont les seuls pour lesquels nous avons le vrai partitionnement.
- Voici les ARI (Adjusted Rand Index) obtenus sur les différents fichiers :

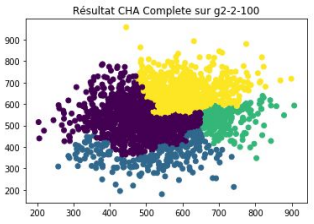
	jain	aggregation	pathbased
KMeans	0.324108	0.774173	0.461822

Nous constatons que les performances de la méthode k-means restent correctes sur le fichier Aggregation.txt, en effet visuellement il y avait tout de même 4 clusters sur 7 qui étaient plutôt bien identifiés. Cependant, pour les fichiers jain.txt et pathbased.txt, nous pouvons confirmer que les performances de la méthode sont mauvaises.

- **Méthode CHA :**

Évaluation qualitative / visuelle :

- Les performances de la méthode semblent très bonnes sur les fichiers g2-2-20.txt, jain.txt et aggregation.txt. Les clusters identifiés correspondent à ceux que nous avions identifiés lors de la visualisation des nuages de points.
- Sur le fichier pathbased.txt, les 3 clusters ne sont pas correctement identifiés, les 2 clusters centraux sont identifiés mais ils “débordent” sur le troisième cluster qui les entoure. La méthode du Clustering Hiérarchique Ascendant semble incapable d’identifier des clusters avec des formes trop particulières et surtout en quelque sorte “imbriqués” les uns avec les autres.
- Notons que sur les données du fichier g2-2-100.txt, la méthode CHA a divisé le nuage des points en 4 clusters de formes et de tailles différentes, on ne peut pas remarquer de tendance particulière.



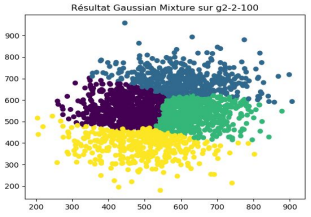
Évaluation quantitative avec les ARI :

```
Indices de Rand Ajustés :
                      jain  aggregation  pathbased
KMeans            0.324108     0.774137   0.461822
CHA               1.000000     1.000000   0.676576
```

Les indices de Rand ajustés confirment que les performances de la méthode CHA sont très bonnes sur les fichiers jain.txt et aggregation.txt.
Ces très bonnes performances indiquent que nous avons choisi les bonnes méthodes de calcul de distance entre clusters (paramètre ‘method’) pour chaque fichier en fonction des observations que nous avons pu faire sur les nuages de points.

- **Modèle de mélange de Gaussiennes :**

Lorsque nous évaluons qualitativement / visuellement la méthode de mélange de Gaussienne (avec la représentation du nuage de points colorés selon les différents clusters produits) :



- Les performances sont très bonnes sur le fichier g2-2-20.txt, on observe 2 clusters séparés de forme elliptique.
- Nous observons une séparation du fichier g2-2-100.txt presque semblable à la méthode K-means. Ce résultat peut être expliqué par le fait que K-means soit la méthode utilisée pour initialiser les paramètres. La seule différence avec la méthode K-means est la forme rectangulaire des clusters.
- Le mélange de Gaussienne identifie parfaitement les clusters de forme arbitraire et de tracé continu comme dans les fichiers jain.txt et pathbased.txt.
- Le fichier aggregation.txt montre les limites du modèle mélange de Gaussienne. Les clusters trop proches, sans séparations nettes, ne sont pas bien identifiés par le modèle et entraînent beaucoup d’instabilité sur les résultats.

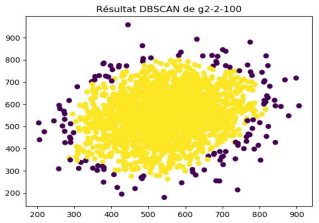
Évaluation quantitative avec les ARI :
L'évaluation des indices de Rand ajustés montre que le mélange de Gaussienne est une méthode efficace. Elle est cependant un peu moins précise sur le fichier aggregation.txt dû à ses clusters très proches.

```
Indices de Rand Ajustés :
                      jain  aggregation  pathbased
KMeans            0.324108     0.774173   0.461822
CHA               1.000000     1.000000   0.676576
Gaussian Mixture  1.000000     0.917654   1.000000
```

- **DBSCAN :**

Lorsque nous évaluons qualitativement / visuellement la méthode de mélange de Gaussienne (avec la représentation du nuage de points colorés selon les différents clusters produits) :

- Sur le fichier g2-20.txt, on observe 2 clusters séparés avec autour des valeurs aberrantes. Cette méthode permet donc de ne prendre en compte, dans les clusters, que les valeurs pertinentes et de trouver automatiquement le bon nombre de clusters.
- Sur le fichier g2-100.txt, un seul cluster se forme avec des valeurs aberrantes autour de celui-ci.
- La méthode DBSCAN permet de trouver parfaitement les clusters de forme arbitraire et plus ou moins séparés. Aucun des clusters des fichiers jain.txt, aggregation.txt et pathbased.txt n'a été mal identifié. Cette méthode n'a donc pas de type de cluster à éviter.



Évaluation quantitative avec les ARI :
La méthode DBSCAN a de très bons indices de Rand ajustés sur les fichier jain.txt, pathbased.txt et aggregation.txt. Cette méthode est donc idéale pour ce type de fichier sous conditions que les paramètres soient bien adaptés.

```
Indices de Rand Ajustés :
                      jain  aggregation  pathbased
KMeans            0.324108     0.774173   0.461822
CHA               1.000000     1.000000   0.676576
Gaussian Mixture  1.000000     0.917654   1.000000
DBSCAN            1.000000     0.998392   0.980955
```

3 - Classement des principaux pays du monde en fonction de leur développement

Nous allons maintenant travailler sur le fichier data.csv, qui stocke différentes données socio-économiques et de santé pour 167 pays différents. L'objectif est maintenant d'utiliser ces 4 méthodes pour classer automatiquement les principaux pays du monde en fonction de leur développement. Cette classification permettra de guider les choix d'affectation de l'aide humanitaire internationale.

3.1 Examen des données

Nous examinons en détail les données du fichier data.csv après les avoir importées.

- Taille du jeu de données : 167 lignes et 10 colonnes. Les lignes correspondent aux pays, les colonnes correspondent à différents attributs tels que le taux de mortalité infantile, le PIB, ou encore l'espérance de vie.
- Type des données : Toutes les données sont numériques à part celles de la colonne 'country' puisqu'il s'agit du nom du pays. Toutes les données numériques sont des float à part 'income' qui est un entier.
- La qualité des données : Nous recherchons le nombre de données manquantes, grâce à la méthode isna() de la classe DataFrame nous trouvons qu'il y a 4 valeurs manquantes. Nous pouvons donc en déduire que nous avons une bonne qualité des données car la proportion de données manquantes est faible.
- La distribution des données :
    - La méthode describe() de la classe DataFrame nous donne énormément d'informations sur la distribution des données, notamment la moyenne, la variance, les minimums et maximums pour chaque colonne, mais aussi le premier et le troisième quartile ainsi que la médiane. L'information count nous indique d'ailleurs que les données manquantes sont dans les colonnes 'GDP' et 'total_fertility' (165 données contre 167 pour les autres colonnes).
    - Nous construisons ensuite une visualisation de type histogramme pour chaque variable numérique avec la méthode hist() de la classe DataFrame. Cela nous permet d'identifier des distributions remarquables telle que la distribution Gaussienne, mais également des valeurs trop éloignées des autres qui pourraient donc être aberrantes.

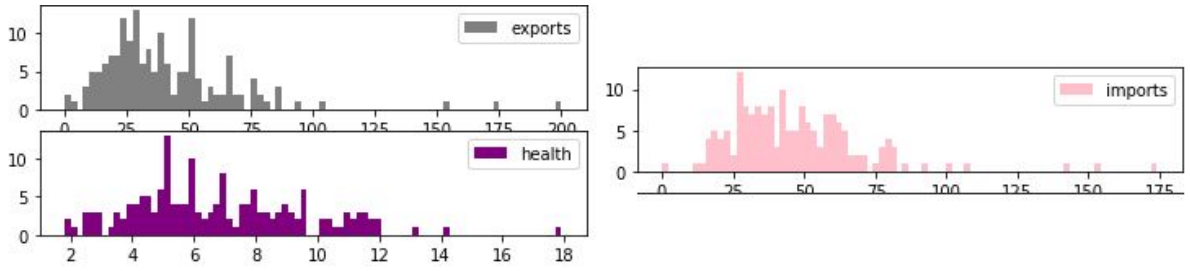
3.2 Préparation des données

De façon à améliorer la qualité des données numériques, nous allons rechercher les données aberrantes et les gérer en suivant plusieurs méthodes.

- 1ère méthode : Nous analysons au cas par cas les valeurs "extrêmes" / à l'écart d'après les histogrammes de chaque attribut, en recherchant les "vraies" valeurs sur internet, pour ensuite décider de les corriger ou non.
    - La valeur maximale qui semble aberrante pour le GDP est celle du pays : Australia avec la valeur : 1000000.0 → Après recherche cette valeur n'est pas aberrante, elle est correcte.
    - La valeur maximale qui semble aberrante pour child_mortality est celle du pays : Haïti avec la valeur : 208.0 → Après recherche cette valeur n'est pas aberrante, elle est correcte.
    - La valeur minimale qui semble aberrante pour life_expectation est celle du pays : Bangladesh avec la valeur : 0.0 → Après recherche la valeur correcte pour l'espérance de vie du Bangladesh est 72.05, nous corrigeons.
    - Nous détectons que l'espérance de vie de Haïti est de 31.1 en observant nos données, ce qui est aberrant, nous corrigeons par la vraie valeur qui est 63.29.

- 2ème méthode :

Afin de détecter les données aberrantes, nous décidons d'appliquer la règle des trois sigmas ou règle empirique, qui indique que pour une loi normale, presque toutes les valeurs se situent dans un intervalle centré autour de la moyenne et dont les bornes se situent à 3 écarts-types de part et d'autre. Nous allons appliquer cette règle uniquement sur les attributs qui ont une distribution qui semble être Gaussienne car pour les autres attributs la règle des trois sigmas ne peut pas s'appliquer.
L'observation des distributions montre que les attributs suivants semblent suivre une loi normale : exports, health et imports. Voici les histogrammes correspondants :



De ce fait, parmi ces attributs, si certaines données sont éloignées de la moyenne à plus de 3 écarts-types, nous décidons qu'elles sont aberrantes, car il y a peu de chance qu'elles soient réelles.

Nous détectons 7 valeurs aberrantes :

- ➔ La valeur 175.0 de la colonne "exports" et du pays Luxembourg
- ➔ La valeur 153.0 de la colonne "exports" et du pays Malta
- ➔ La valeur 200.0 de la colonne : exports et du pays Singapore
- ➔ La valeur 17.9 de la colonne "health" et du pays United States
- ➔ La valeur 142.0 de la colonne "imports" et du pays Luxembourg
- ➔ La valeur 154.0 de la colonne "imports" et du pays Malta
- ➔ La valeur 174.0 de la colonne "imports" et du pays Singapore

Après avoir identifié ces valeurs aberrantes, nous les remplaçons par NaN pour ensuite pouvoir les traiter en même temps que les valeurs manquantes.Nous remplaçons ensuite toutes les valeurs manquantes (comprenant donc les valeurs aberrantes), 11 au total, par la moyenne de l'attribut correspondant. Nous aurions également pu les remplacer par les médianes mais nous faisons ce choix afin de ne pas trop modifier les distributions.

Il est aussi nécessaire de recalibrer les données car les plages de valeurs sont très différentes d'un attribut à l'autre, nous l'observons dans le résultat de la méthode describe(). Nous décidons donc de centrer et réduire les données en utilisant la classe StandardScaler de scikit-learn. Pour utiliser cette méthode, il faut que toutes nos données soient numériques, c'est pourquoi nous devons nous débarrasser de la première colonne 'country'. Notre fonction remplacer_val_aberrantes() a d'ailleurs également nécessité de supprimer cette première colonne, afin de faire la méthode des trois sigmas uniquement sur des valeurs numériques. Cette fonction nous renvoie donc nos données sans la première colonne, ainsi que la première colonne séparément afin de pouvoir continuer à travailler sur celle-ci pour identifier les différents pays par la suite.

Nous appliquons la méthode StandardScaler, de ce fait nous vérifions que la moyenne est à 0 et la variance à 1 pour Z, la matrice de données centrées réduites que nous obtenons.

3.3 Recherche de corrélations

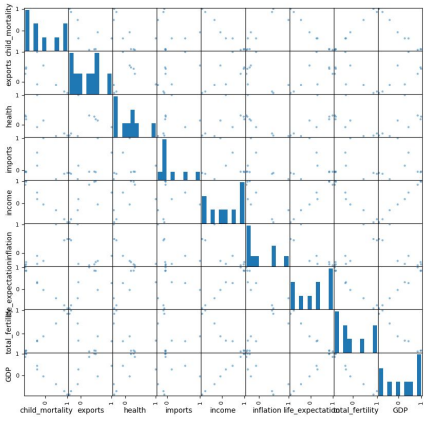
Nous nous intéressons désormais aux relations qui existent entre les variables, en calculant la matrice de corrélation de nos données.

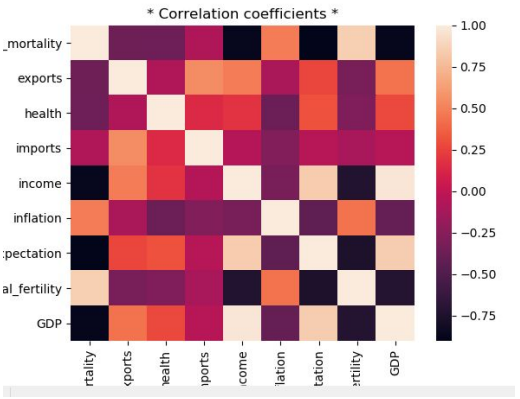
Nous créons d'abord une fonction qui permet de trouver les attributs corrélés à partir d'un certain seuil. Nous fixons ce seuil à 0,8 et nous obtenons les attributs corrélés suivants : 'income' et 'child_mortality', 'life_expectation' et 'child_mortality', 'life_expectation' et 'income', 'total_fertility' et 'child_mortality', 'GDP' et 'child_mortality', 'GDP' et 'income', 'GDP et life_expectation'.

Nous créons une deuxième fonction permettant de trouver les attributs :

- les plus corrélés : 'GDP' et 'income' avec comme corrélation 0.9772 environ
- les plus corrélés positivement : 'GDP' et 'income' avec comme corrélation 0.9772 environ
- les plus corrélés négativement : 'life_expectation' et 'child_mortality' avec comme corrélation -0.9034 environ
- les moins corrélés : 'GDP' et 'imports' avec comme corrélation -0.0427714685098816

Nous visualisons ensuite la matrice de corrélation de deux méthodes différentes :





- avec la fonction **`scatter_matrix()`** de la librairie pandas : (à gauche)

Ici nous visualisons bien les corrélations positives lorsque les points forment des droites croissantes, et négatives lorsque les points forment des droites décroissantes.

- avec la fonction **`heatmap()`** de la librairie seaborn : (à droite)

Ici ce sont les coefficients de corrélations qui sont représentés par des couleurs selon une échelle de façon à mieux identifier les fortes corrélations, c'est plus visuellement frappant que lorsqu'on regarde la matrice de corrélation.

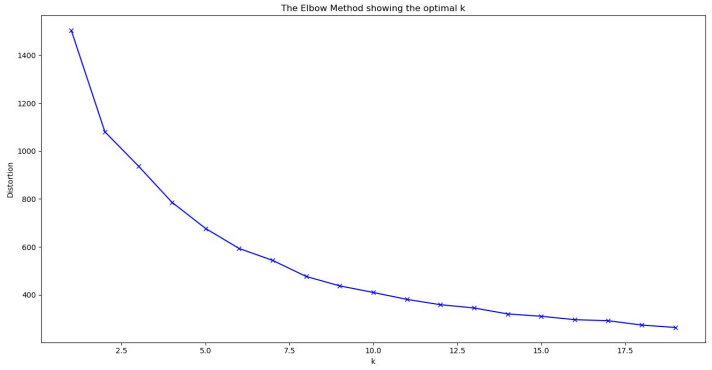
Nous pourrions supprimer les attributs avec une corrélation supérieure à 0.8 : income, life_expectation, life_expectation et income, total_fertility, GDP. Ceci permettrait d'avoir des données moins complexes pour rendre un traitement plus facile. Nous ne supprimons aucune colonne car le sujet ne le demande pas.

3.4 Clustering des données

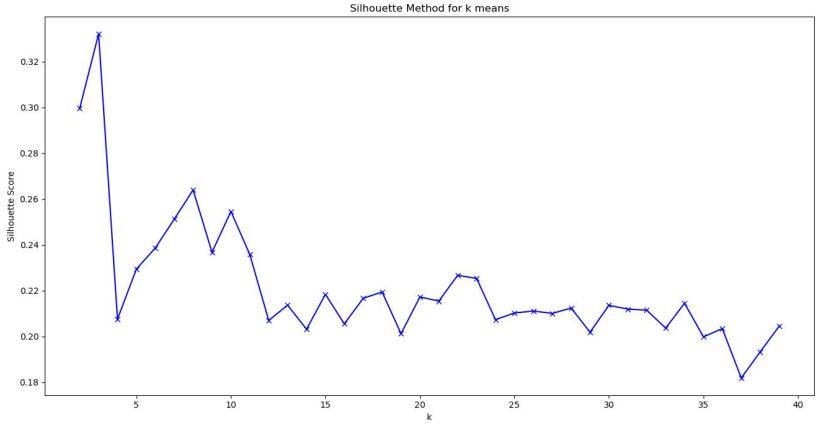
Nous effectuons maintenant le clustering des données. Pour cela, nous devons d'abord trouver le bon nombre de clusters devant séparer les données.

Nous créons deux fonctions permettant de trouver le bon nombre de clusters :

- **`elbow_method_kmeans()`** qui trace un graphique de l'inertie des données avec la méthode K-means en fonction du nombre de clusters. Cette méthode ne donne pas de résultat net car il n'y a pas de "coude" sur le graphique.



- **`silhouette_score()`** qui trace un graphique du score silhouette avec la méthode choisie (ici K-means) en fonction du nombre de clusters. Nous obtenons comme plus grand maximum local 8 clusters. Nous ne prenons pas le maximum 3 car il donne des clusters avec un nombre trop élevé de données.



- **Méthode K-means :**

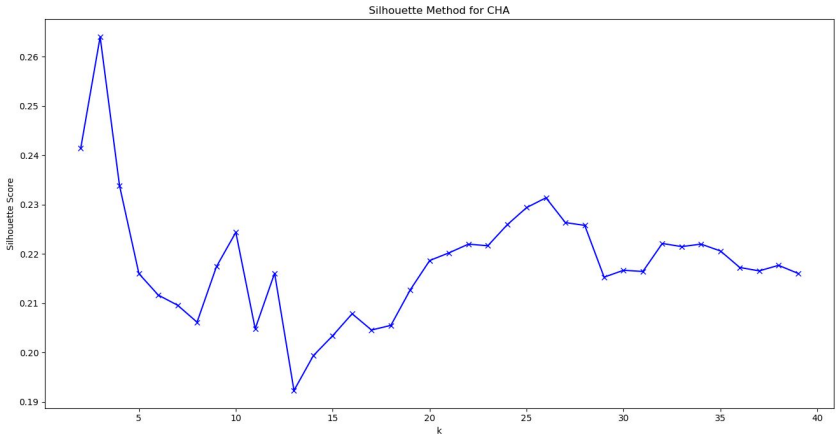
Nous appliquons la méthode K-means avec 8 clusters, nous définissons donc le paramètre n_clusters à 8, le paramètre n_init à 20 (afin de réaliser un nombre suffisant d'initialisations différentes et de sélectionner le meilleur résultat), et nous laissons la valeur par défaut pour le paramètre 'init' qui est 'k-means++', car nous estimons que c'est la méthode d'initialisation la plus performante.
Nous pouvons remarquer que deux listes comportent les pays qui ont le plus besoin de l'aide humanitaire internationale.

```
list 6:
 ['Afghanistan', 'Angola', 'Benin', 'Burkina Faso', 'Cameroon',
'Central African Republic', 'Chad', 'Comoros', 'Congo Dem. Rep.',
'Congo Rep.', "Cote d'Ivoire", 'Equatorial Guinea', 'Eritrea', 'Gabon',
'Gambia', 'Ghana', 'Guinea', 'Kenya', 'Lao', 'Madagascar', 'Malawi',
'Mali', 'Mauritania', 'Mozambique', 'Niger', 'Pakistan', 'Senegal',
'Sudan', 'Tanzania', 'Timor-Leste', 'Uganda', 'Yemen', 'Zambia']
```

```
list 2:
 ['Botswana', 'Burundi', 'Guinea-Bissau', 'Haiti', 'Kiribati',
'Lesotho', 'Liberia', 'Micronesia Fed. Sts.', 'Namibia', 'Rwanda',
'Sierra Leone', 'Solomon Islands', 'South Africa', 'Togo']
```

- **CHA :**

Nous utilisons la fonction **`silhouette_score()`** pour tracer un graphique du score silhouette avec la méthode CHA en fonction du nombre de clusters. Le maximum local le plus élevé est à 26 clusters.



Nous avons décidé de définir le paramètre ‘method’ de la fonction linkage à ‘ward’ car nous estimons que c’est la distance qui nous donnent les meilleurs résultats, d’après notre analyse sur les données simulées dans la partie 2. Nous appliquons ensuite la méthode du Clustering Hiérarchique Ascendant grâce à la fonction fcluster en définissant le paramètre ‘t’ à 26 et le paramètre ‘criterion’=’maxclust’ pour avoir 26 clusters formés.

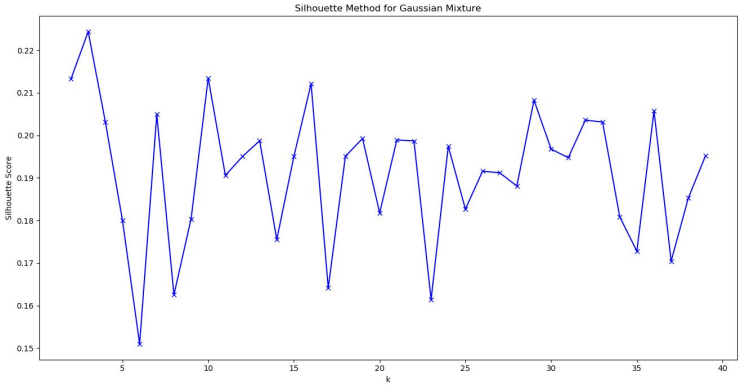
Nous obtenons alors 26 listes de pays. Nous pouvons remarquer que deux listes comportent les pays qui ont le plus besoin de l’aide humanitaire internationale.

```
list 6:
 ['Afghanistan', 'Burkina Faso', 'Burundi', 'Guinea-Bissau', 'Rwanda',
'Sierra Leone', 'Uganda']

list 7:
 ['Benin', 'Cameroon', 'Central African Republic', 'Chad', 'Congo Dem.
Rep.', "Cote d'Ivoire", 'Guinea', 'Malawi', 'Mali', 'Mozambique',
'Tanzania', 'Zambia']
```

- **Mélange de Gaussiennes :**

Nous utilisons la fonction **`silhouette_score()`** pour tracer un graphique du score silhouette avec la méthode Mélange de Gaussiennes en fonction du nombre de clusters. Nous obtenons un maximum local pour 29 clusters.



Nous effectuons la méthode Mélange de Gaussiennes sur les données en prenant 29 clusters et 50 initialisations pour avoir un résultat stable. Nous laissons les autres paramètres par défaut, notamment le type de covariance (‘full’) et la méthode pour initialiser les poids, les moyennes et la précision (‘K-means’).
Nous obtenons alors 29 listes de pays. Nous pouvons remarquer que deux listes comportent les pays qui ont le plus besoin de l’aide humanitaire internationale.

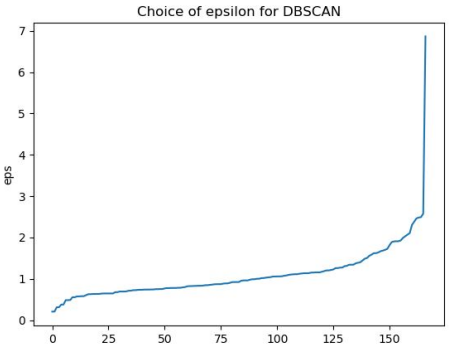
```
list 0:
 ['Chad', 'Congo Dem. Rep.', "Cote d'Ivoire", 'Guinea', 'Malawi',
'Mozambique', 'Togo', 'Zambia']
```

```
list 9:
 ['Afghanistan', 'Burundi', 'Guinea-Bissau', 'Rwanda', 'Sierra Leone',
'Uganda']
```

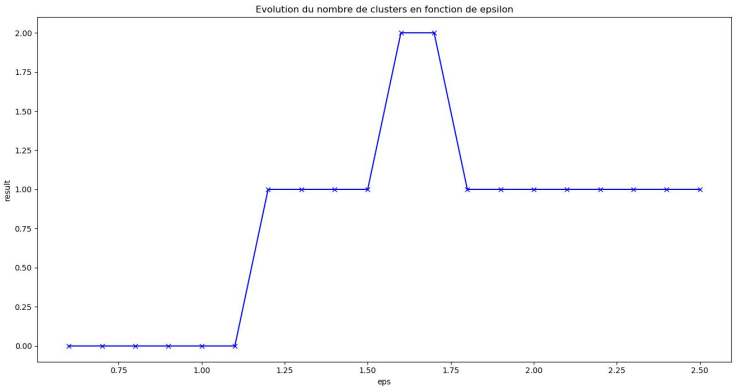
- **DBSCAN :**

Pour utiliser la méthode DBSCAN, nous devons d’abord déterminer ses paramètres. Pour cela nous utilisons deux fonctions :

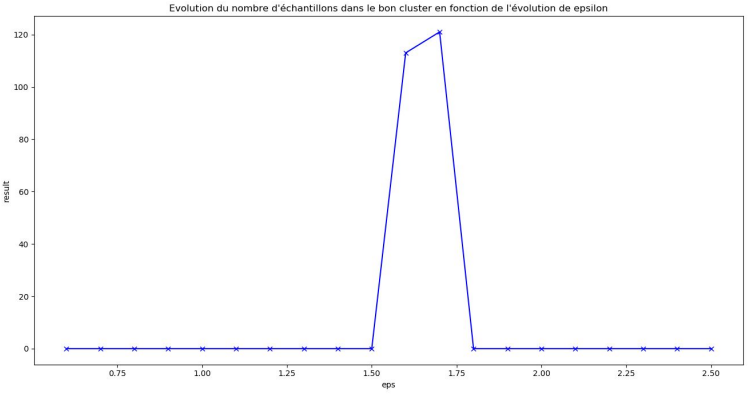
- **`epsilon()`** qui calcule pour chaque point la distance à ses deux plus proches voisins, puis trie et trace les résultats. Le graphique obtenu nous indique que la valeur de epsilon doit se trouver entre 1.5 et 2.6.



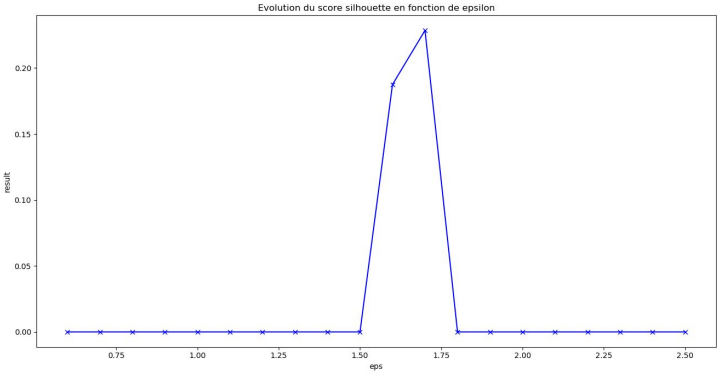
- **`eps_DBSCAN()`** qui trace 3 graphiques au choix :
  - l'évolution du nombre de clusters en fonction de epsilon : nous répétons cette fonction en faisant varier n_samples pour obtenir un epsilon compris entre 1.5 et 2.6 donnant un nombre de clusters supérieur à 2. Nous obtenons un résultat pour n_sample égal à 15 et trouvons epsilon compris entre 1.6 et 1.7.



  - l'évolution du nombre d'échantillons dans le bon cluster en fonction de l'évolution de epsilon (utilisation du score silhouette) : cette fonction comporte un maximum. Nous choisissons donc le epsilon le plus élevé possible soit 1.7 avec 120 échantillons bien classés.



  - l'évolution du score silhouette en fonction de epsilon : nous observons également un maximum et le choix de epsilon à 1.7 est confirmé. Son score silhouette est de 0.23.



Nous fixons donc epsilon à 1.7 et n_sample à 15 dans les paramètres de la méthode DBSCAN. Nous laissons les autres paramètres par défaut. Nous obtenons 2 clusters représentant les pays qui ont ou pas besoin de l'aide humanitaire, nous obtenons également des valeurs aberrantes. Nous pouvons noter que les pays suivants ont le plus besoin d'aide humanitaire :

```
list 1  :
 ['Afghanistan', 'Benin', 'Burkina Faso', 'Cameroon', 'Central African Republic', 'Chad', 'Congo Dem.
Rep.', "Cote d'Ivoire", 'Eritrea', 'Gambia', 'Ghana', 'Guinea', 'Guinea-Bissau', 'Madagascar',
'Malawi', 'Mali', 'Mozambique', 'Senegal', 'Sudan', 'Tanzania', 'Togo', 'Uganda', 'Zambia']
```

Après comparaison des pays ayant le plus besoin de l'aide humanitaire internationale avec chaque méthode, nous remarquons que 9 pays apparaissent dans toutes les méthodes. Ces pays sont donc ceux qui ont le plus besoin de l'aide humanitaire internationale :
Chad, Cote d'Ivoire, Malawi, Guinea, Mozambique, Zambia, Afghanistan, Uganda, Congo Dem Rep

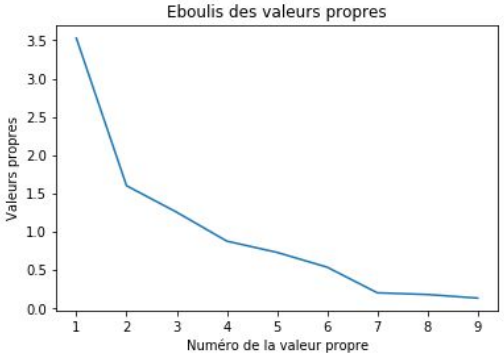
3.5 Clustering des données après réduction de dimension

Nous allons effectuer une ACP (Analyse en Composantes Principales) sur nos données afin d'en réduire la dimension.

Afin de déterminer le nombre d'axes à conserver nous utilisons deux critères fréquemment utilisés :

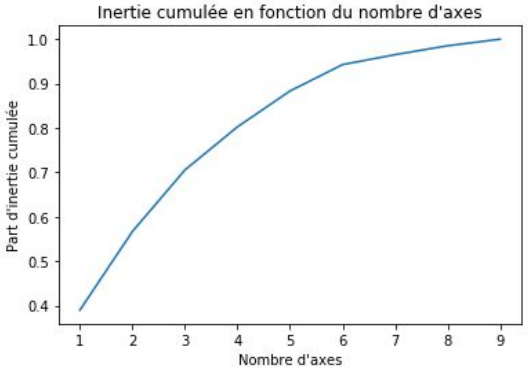
- l'éboulis des valeurs propres : on trace un graphe représentant la décroissance des valeurs propres, on recherche un "coude" dans le graphe, on retient ensuite les axes associés aux valeurs propres situées avant le "coude".
- la part d'inertie : on choisit le nombre d'axes de façon à conserver une certaine part de l'inertie totale.

Voici le graphe obtenu pour l'éboulis des valeurs propres :



Nous y voyons 2 "coudes" possibles, car pas très marqués : un premier pour la valeur propre numéro 2, un deuxième pour la valeur propre numéro 4.
Nous pouvons donc choisir de conserver 2 ou 4 axes. Le deuxième critère va sûrement nous aider à faire ce choix.

Voici le graphe obtenu pour l'évolution de l'inertie cumulée en fonction du nombre d'axes :

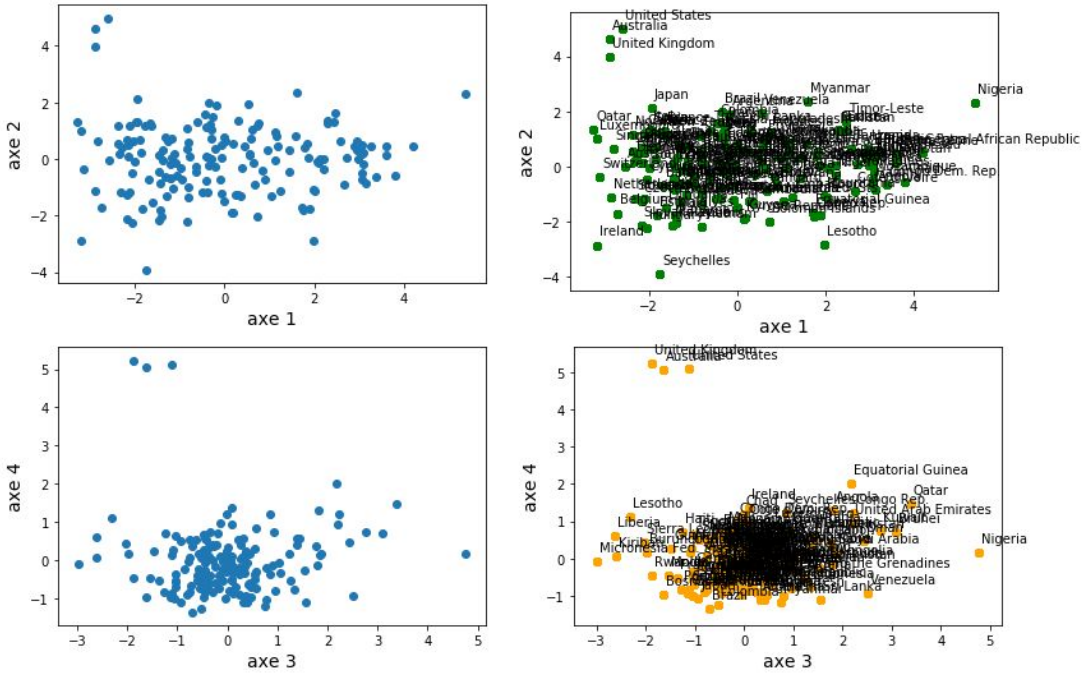


Nous pouvons voir ici que si nous ne conservons que les 2 premiers axes la part d'inertie cumulée n'atteindra que 56% environ, ce qui est faible. En revanche, pour 4 axes, nous atteignons plus de 80% de l'inertie, ce qui est un résultat très correct.

Nous décidons donc de conserver les 4 premiers axes.

Nous évaluons la qualité globale de la représentation du nuage avec les 4 premiers vecteurs principaux en calculant la part d'inertie expliquée par le sous-espace considéré. Nous obtenons une qualité globale de 0.8025458 environ. Cette valeur étant proche de 1, le nuage d'origine a donc une projection "fidèle" en conservant les 4 premiers axes.

Lorsque nous représentons le nuage des individus projeté dans le premier plan principal défini par les axes 1 et 2, puis dans le deuxième plan principal défini par les axes 3 et 4, voici ce que nous obtenons :



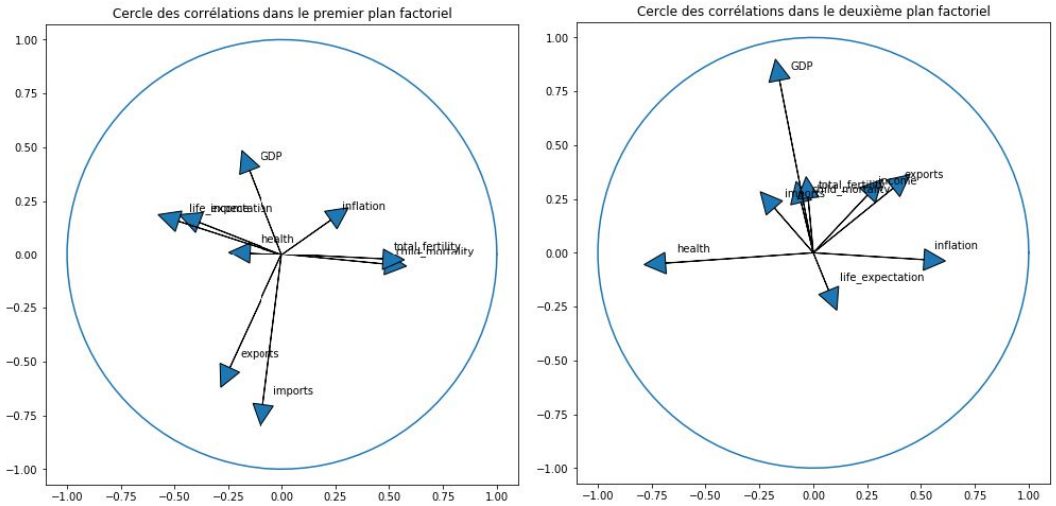
Afin de pouvoir donner une interprétation de ces axes, nous allons dans un premier temps analyser les contributions des individus aux 4 axes principaux, puis dans un second temps les contributions des variables aux 4 axes principaux.

Lorsque nous calculons les contributions des individus pour chacun des axes, nous trouvons ensuite que :

- L'individu qui contribue le plus à l'axe 1 est : Nigeria
- L'individu qui contribue le plus à l'axe 2 est : United States
- L'individu qui contribue le plus à l'axe 3 est : Nigeria
- L'individu qui contribue le plus à l'axe 4 est : United Kingdom

Malheureusement, les contributions des individus aux axes ne sont pas exploitables car nous n'avons pas les connaissances sur les pays, elles ne nous permettent donc pas d'interpréter les différents axes.

Nous nous penchons alors sur les contributions des variables aux axes, pour ce faire nous représentons les cercles des corrélations pour les 2 premiers plans factoriels :



- Interprétation du cercle des corrélations dans le premier plan factoriel :

	Axe 1	Axe 2
Variables qui contribuent le plus, positivement	total_fertility child_mortality	GDP
Variables qui contribuent le plus, négativement	life_expectation incomes	exports imports

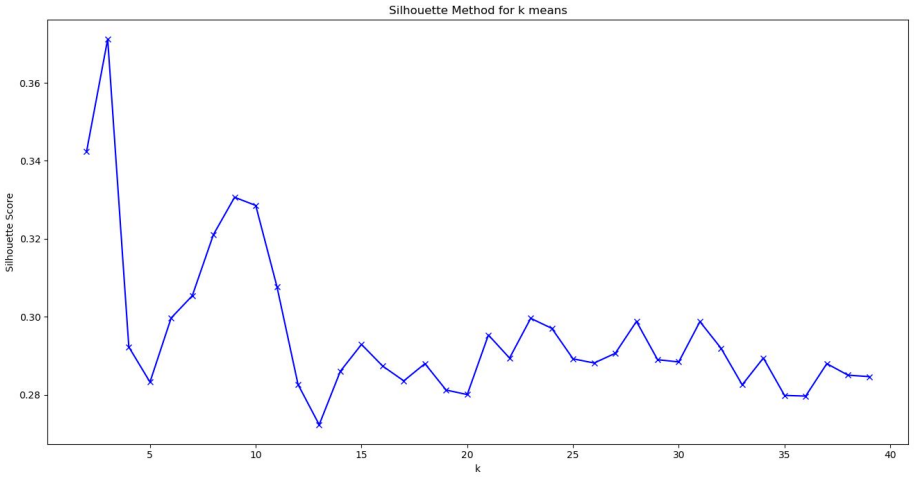
- Interprétation du cercle des corrélations dans le premier plan factoriel :

	Axe 3	Axe 4
Variables qui contribuent le plus, positivement	inflation	GDP
Variables qui contribuent le plus, négativement	health	life_expectation

Après avoir effectué l'ACP, nous appliquons les méthodes de clustering sur les nouvelles données.

- **Méthode K-means :**

Nous utilisons la fonction **`silhouette_score()`** pour tracer un graphique du score silhouette avec la méthode K-means en fonction du nombre de clusters. Le maximum local le plus élevé est à 9 clusters.



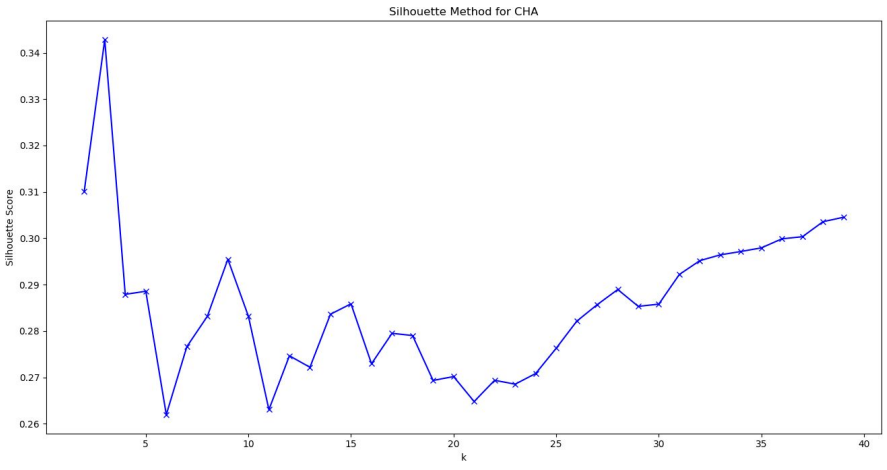
Nous appliquons donc la méthode K-means avec 9 clusters : le paramètre n_clusters est fixé à 9, le paramètre n_init à 20 (afin de réaliser un nombre suffisant d'initialisations différentes et de sélectionner le meilleur résultat), et nous laissons la valeur par défaut pour le paramètre 'init' qui est 'k-means++', car nous estimons que c'est la méthode d'initialisation la plus performante.
Nous obtenons alors 9 listes de pays. Nous pouvons remarquer que deux listes comportent les pays qui ont le plus besoin de l'aide humanitaire internationale.

```
list 1:
 ['Angola', 'Congo Rep.', 'Equatorial Guinea', 'Mauritania', 'Nigeria']
```

```
list 7:
 ['Afghanistan', 'Benin', 'Burkina Faso', 'Burundi', 'Cameroon',
'Central African Republic', 'Chad', 'Comoros', 'Congo Dem. Rep.', "Cote
d'Ivoire", 'Eritrea', 'Gambia', 'Ghana', 'Guinea', 'Guinea-Bissau',
'Haiti', 'Kenya', 'Lao', 'Madagascar', 'Malawi', 'Mali', 'Mozambique',
'Niger', 'Pakistan', 'Rwanda', 'Senegal', 'Sierra Leone', 'South
Africa', 'Sudan', 'Tanzania', 'Timor-Leste', 'Uganda', 'Zambia']
```

- **CHA :**

Nous utilisons la fonction **`silhouette_score()`** pour tracer un graphique du score silhouette avec la méthode CHA en fonction du nombre de clusters. Le maximum local le plus élevé est à 28 clusters.

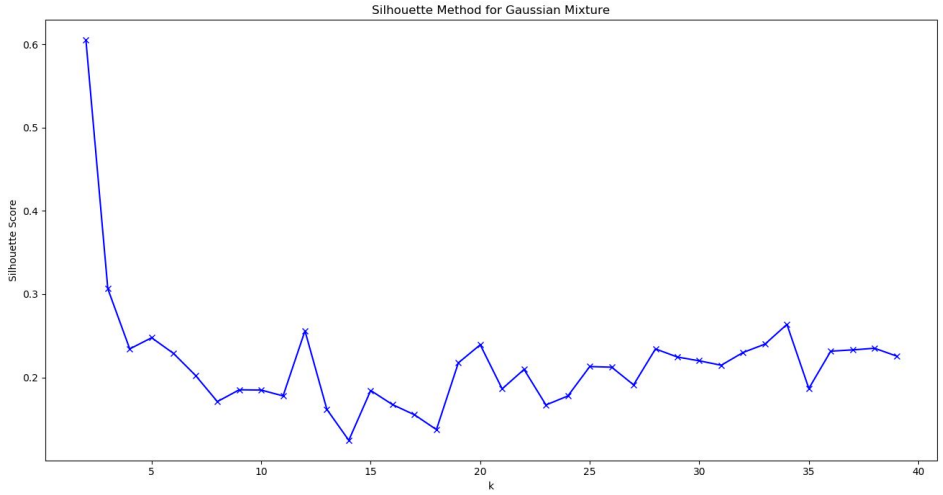


Nous fixons le paramètre ‘method’ de la fonction linkage à ‘ward’ car nous estimons que c’est la distance qui nous donnent les meilleurs résultats, d’après notre analyse sur les données simulées dans la partie 2. Nous utilisons la méthode du Clustering Hiérarchique Ascendant grâce à la fonction fcluster en définissant le paramètre ‘t’ à 28 et le paramètre ‘criterion’=’maxclust’ pour avoir 28 clusters formés.
Nous obtenons alors 28 listes de pays. Nous pouvons remarquer qu’une liste comporte les pays qui ont le plus besoin de l’aide humanitaire internationale.

```
list 1:
 ['Chad', 'Congo Dem. Rep.', "Cote d'Ivoire", 'Guinea', 'Haiti',
'Mozambique', 'Togo', 'Zambia']
```

- **Mélange de Gaussiennes :**

Nous utilisons la fonction **`silhouette_score()`** pour tracer un graphique du score silhouette avec la méthode Mélange de Gaussiennes en fonction du nombre de clusters. Nous obtenons un maximum pour 20 clusters.



Nous effectuons la méthode Mélange de Gaussiennes sur les données en prenant 20 clusters et 50 initialisations pour avoir un résultat stable. Nous laissons les autres paramètres par défaut, notamment le type de covariance (‘full’) et la méthode pour initialiser les poids, les moyennes et la précision (‘K-means’).
Nous obtenons alors 20 listes de pays. Nous pouvons remarquer que deux listes comportent les pays qui ont le plus besoin de l’aide humanitaire internationale.

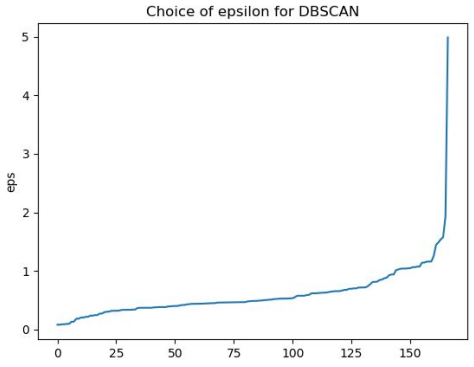
```
list 17:
 ['Benin', 'Cameroon', 'Chad', 'Congo Dem. Rep.', "Cote d'Ivoire",
'Guinea', 'Mali', 'Mozambique', 'Tanzania', 'Zambia']
```

```
list 8:
 ['Afghanistan', 'Burundi', 'Central African Republic', 'Guinea-
Bissau', 'Haiti', 'Malawi', 'Sierra Leone', 'Uganda']
```

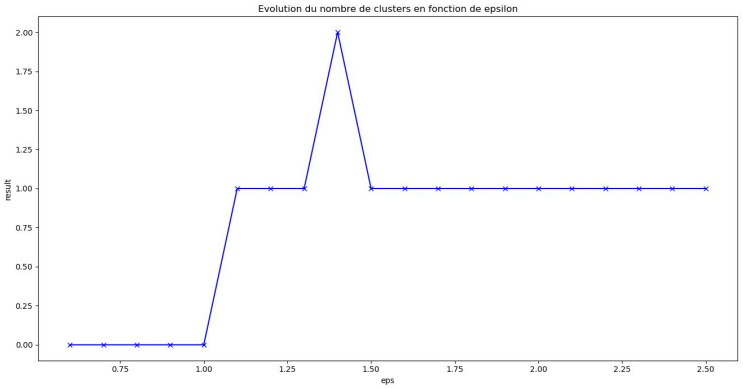
- **DBSCAN :**

Pour utiliser la méthode DBSCAN, nous devons d'abord déterminer ses paramètres. Pour cela nous utilisons deux fonctions :

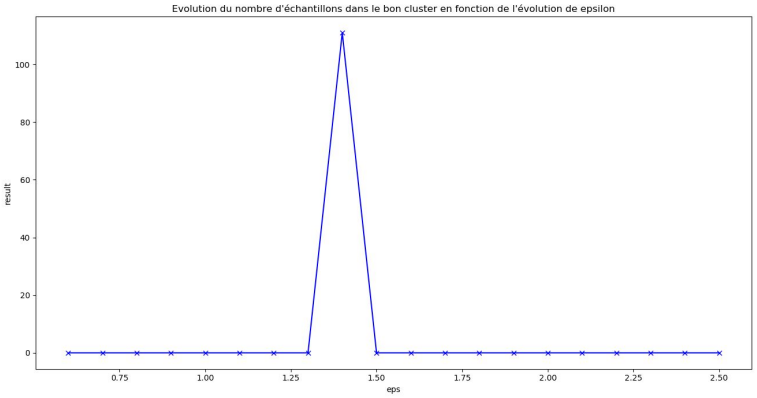
- **`epsilon()`** qui calcule pour chaque point la distance à ses deux plus proches voisins, puis trie et trace les résultats. Le graphique obtenu nous indique que la valeur de epsilon doit se trouver entre 0.6 et 1.6.



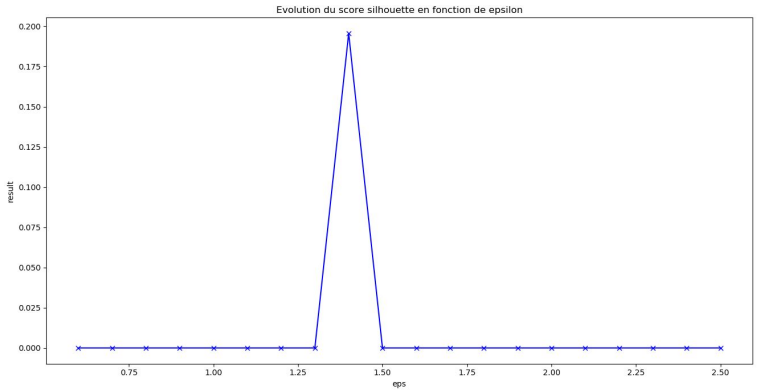
- **`eps_DBSCAN()`** qui trace 3 graphiques au choix :
  - l'<u>évolution du nombre de clusters en fonction de epsilon</u> : nous répétons cette fonction en faisant varier n_samples pour obtenir un epsilon compris entre 0.6 et 1.6 donnant un nombre de clusters supérieur à 2. Nous obtenons un résultat pour n_sample égal à 24 et epsilon égal à 1.4.



  - l'<u>évolution du nombre d'échantillons dans le bon cluster en fonction de l'évolution de epsilon</u> (utilisation du score silhouette) : nous observons un maximum local pour epsilon égal à 1.4. Pour la valeur de epsilon soit 1.4, nous obtenons 111 échantillons bien classés.



- l'évolution du score silhouette en fonction de epsilon : nous observons également un maximum local pour la valeur 1.4. Le score silhouette pour epsilon égal à 1.4 est de 0.195.



Nous fixons donc epsilon à 1.4 et n_sample à 24 dans les paramètres de la méthode DBSCAN. Nous laissons les autres paramètres par défaut. Nous obtenons 2 clusters représentant les pays qui ont ou pas besoin de l'aide humanitaire, nous obtenons également des valeurs aberrantes. Nous pouvons noter que les pays suivants ont le plus besoin d'aide humanitaire :

```
list 1  :
 ['Afghanistan', 'Benin', 'Burkina Faso', 'Cameroon', 'Comoros', "Cote d'Ivoire", 'Gambia', 'Ghana',
'Guinea', 'Guinea-Bissau', 'Haiti', 'Madagascar', 'Malawi', 'Mozambique', 'Niger', 'Tanzania', 'Togo',
'Uganda', 'Zambia']
```

Après comparaison des pays ayant le plus besoin de l'aide humanitaire internationale avec chaque méthode, nous remarquons que 5 pays apparaissent dans toutes les méthodes. Ces pays sont donc ceux qui ont le plus besoin de l'aide humanitaire internationale :
Cote d'Ivoire, Guinea, Mozambique, Zambia, Haiti

Les nouvelles partitions obtenues avec l'acp ont un meilleur score silhouette excepté pour la méthode DBSCAN. Elles sont donc de meilleures qualités.
On peut également noter que la valeur du nombre de clusters optimal est environ la même pour les méthodes K-means, CHA et Mélange de Gaussiennes.

On peut remarquer que des pays apparaissent à la fois dans la liste des 10 pays qui ont le plus besoin de l'aide humanitaire internationale des données initiales et dans celle des données après ACP : ***Cote d'Ivoire, Guinea, Mozambique, Zambia***

En conclusion, les silhouettes scores étant meilleurs pour chacune des méthodes après avoir appliqué l'ACP, nous décidons de choisir les pays identifiés comme ayant le plus besoin de l'aide humanitaire après l'ACP, c'est-à-dire : ***Cote d'Ivoire, Guinea, Mozambique, Zambia, Haiti***