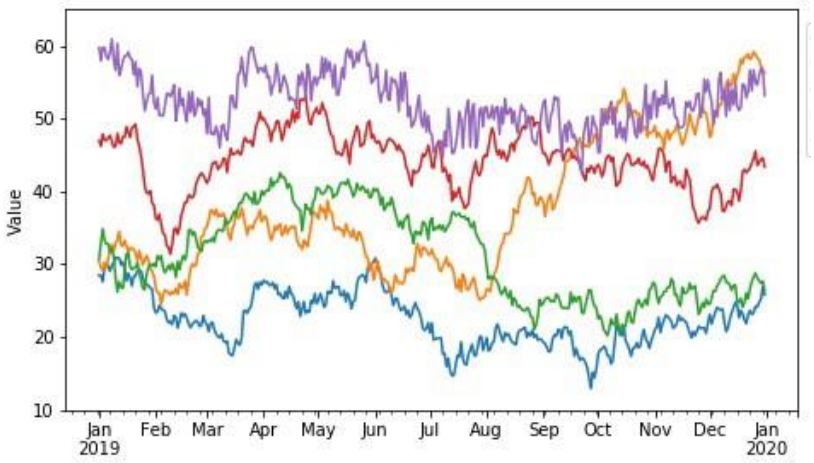
Université Paris Cité

Rapport du Projet d'apprentissage non supervisé

(Classification des Time Series)

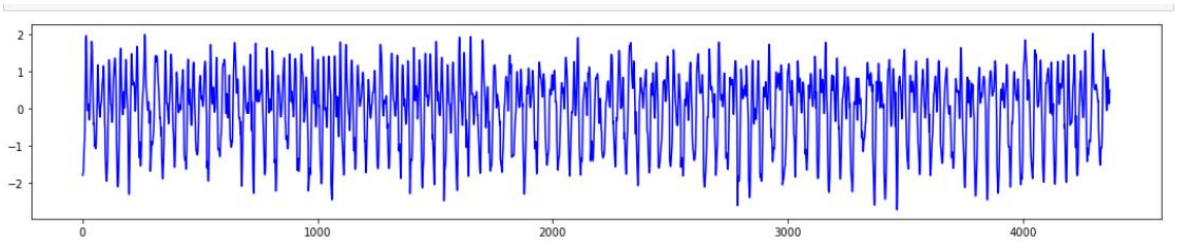


Réalisé par :

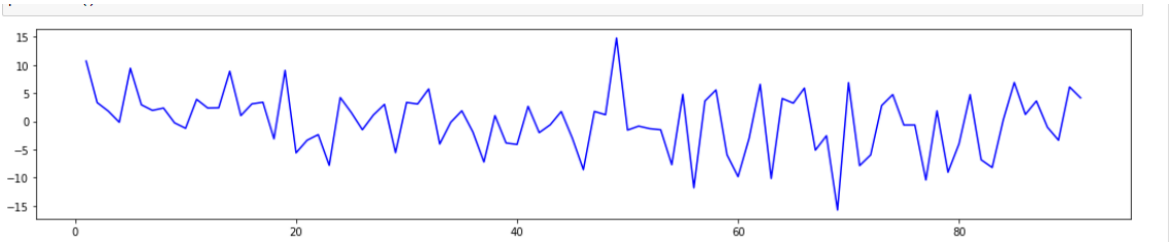
HILMI EL Mehdi

Année scolaire 2022/2023

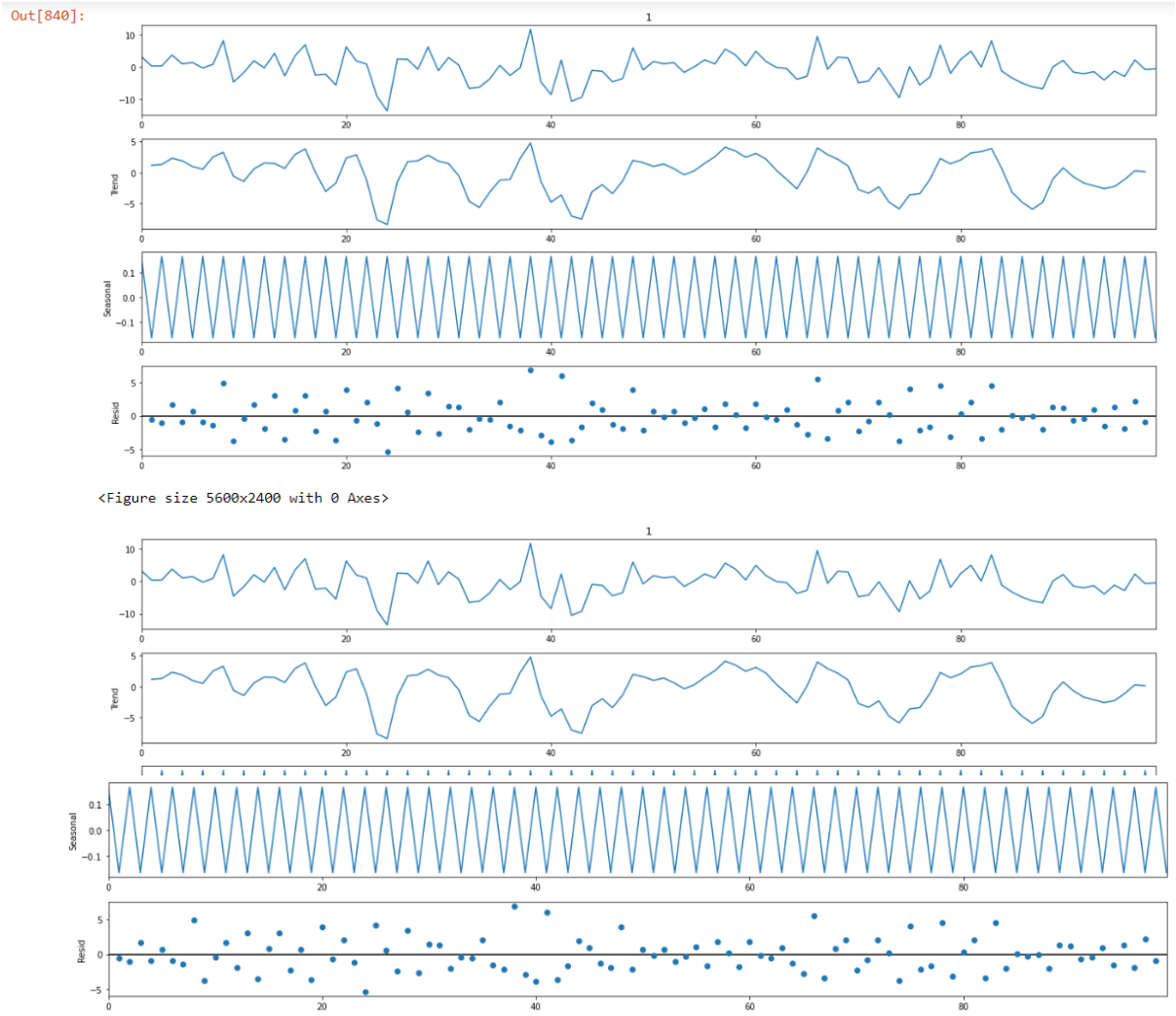
Données de taille 100 × 4368 :



Passer à l'échelle journalière :



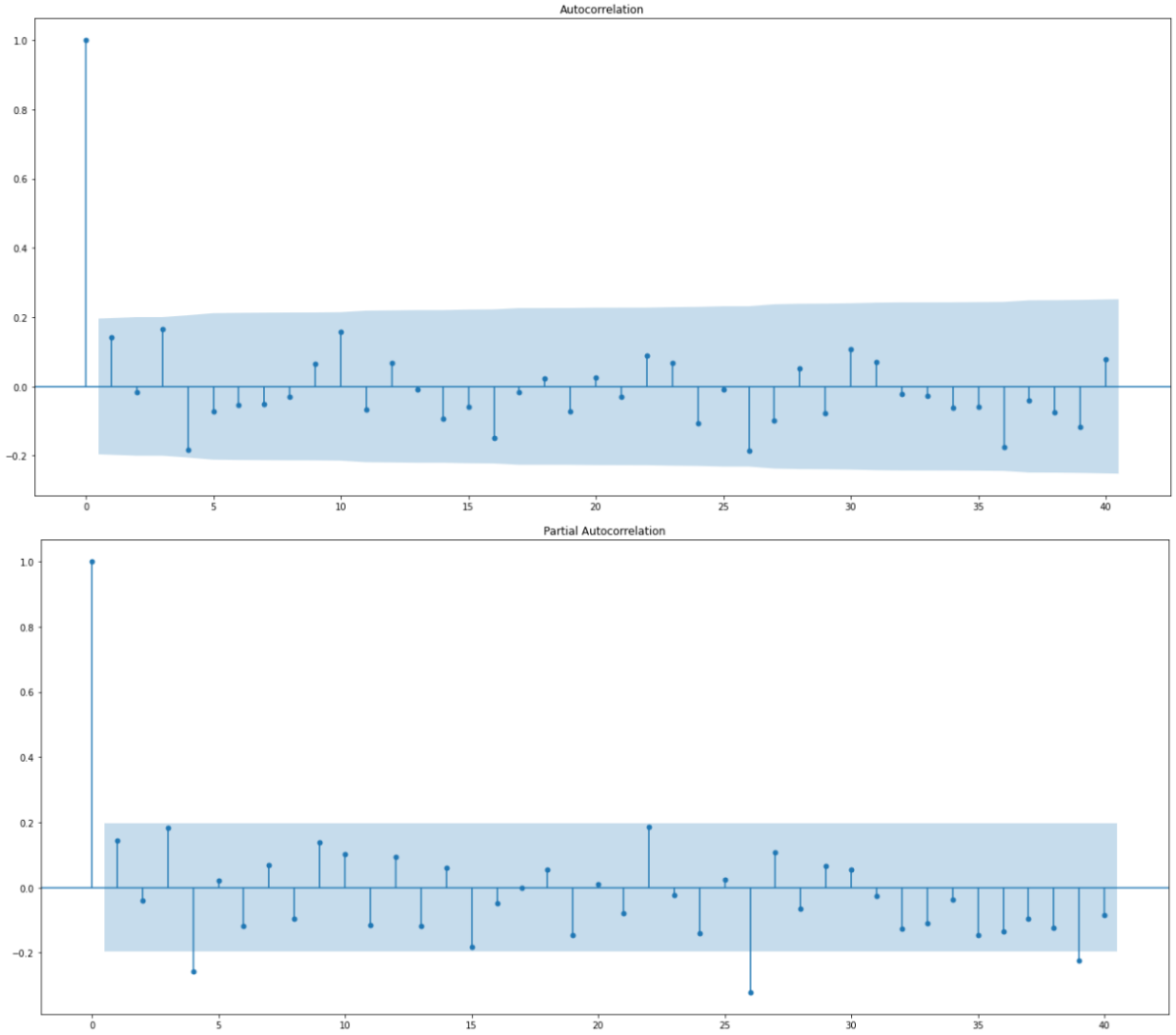
Décomposition d'une serie temporelle (y(t) = Level + Trend + Seasonality + Noise) (2eme serie comme exemple).



- **Autocorrélation :**

Graphiques d'autocorrélation et d'autocorrélation partielle Les graphiques d'autocorrélation montrent comment les valeurs au temps t sont corrélées avec les valeurs suivantes au temps t+1,t+2,..t+n. Si les données ne sont pas stationnaires, les valeurs d'autocorrélation seront fortement corrélées avec des points distants dans le temps montrant des saisonnalités ou des tendances possibles.

Les valeurs d'autocorrélation des séries stationnaires diminueront rapidement au cours du temps t. Cela nous montre qu'aucune information n'est reportée dans le temps et que la série doit alors être constante dans le temps.



- **Test pour vérifier la stationnarité des séries :**

- **Hypothèse nulle (H0)** : si elle n'est pas rejetée, cela suggère que la série chronologique a une racine unitaire, ce qui signifie qu'elle est non stationnaire. Il a une structure dépendante du temps.
- **Hypothèse alternative (H1)** : L'hypothèse nulle est rejetée ; cela suggère que la série chronologique n'a pas de racine unitaire, ce qui signifie qu'elle est stationnaire.

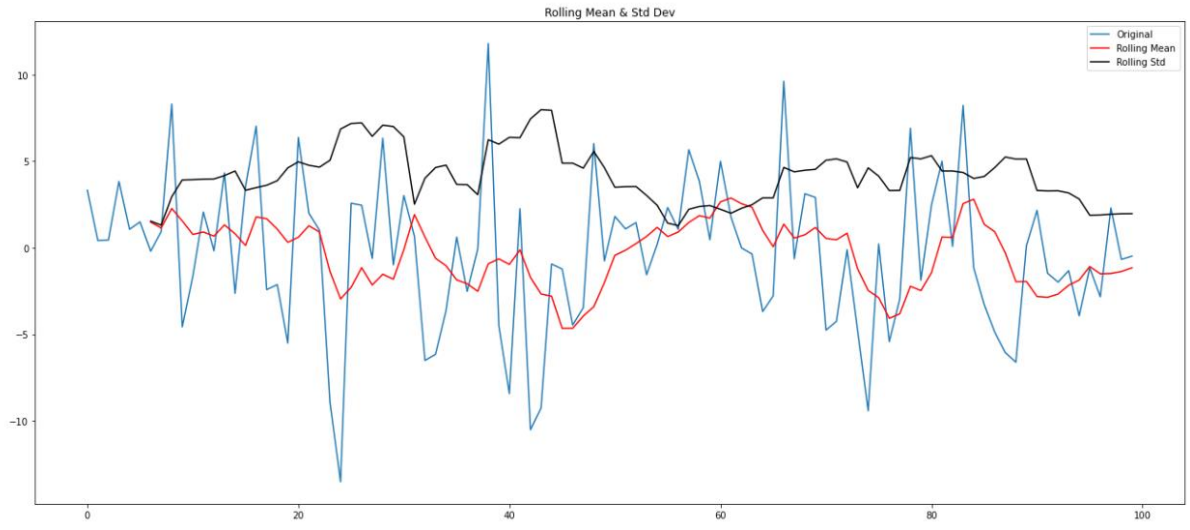
```
Entrée [844]: X = ser.values
              result = adfuller(X)
              print('ADF Statistic: %f' % result[0])
              print('p-value: %f' % result[1])
              print('Critical values:')
              for k, val in result[4].items():
                  print('\t%s: %.3f' % (k, val))

              ADF Statistic: -5.302997
              p-value: 0.000005
              Critical values:
                      1%: -3.500
                      5%: -2.892
                      10%: -2.583
```

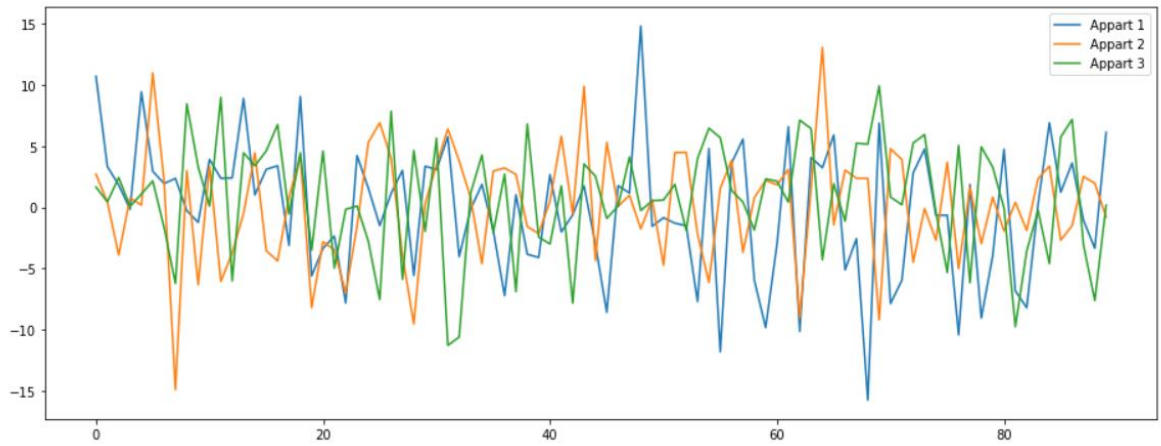
D'où la série est stationnaire, car p_value < 0,05 d'une part, et d'autre part la mesure ADF statistique est inferieur a la valeur critique, donc on peut effectuer notre traitement (on a vérifié la stationnarité des autres séries)

- **<u>Smoothing average :</u>**

Nous parlions de la façon dont notre moyenne et notre écart type devraient être constants dans le temps afin d'avoir une série temporelle stationnaire,



- **Exemple des trois premiers ménages :**



Dans les premiers jours, on remarque que l'appartement 2 (orange) consomme moins d'électricité, et l'appartement 1 (bleu) consomme le plus. En revanche, à partir de 50ème jours l'appartement 1 commence à consommer le moins d'électricité. Ainsi qu'on remarque que l'appartement 3 (vert) est presque la plus stable dans la consommation tout au long des 91 jrs.

- **Preprocessing :**

```
Entrée [129]: ts_days.shape
   Out[129]: (100, 91)

Entrée [134]: df.isnull().sum().sum()
   Out[134]: 0
```

We have any missing values

We will scale our time series (centre-reduit)

```
Entrée [985]: # we will scale our time series (centre-reduit)
              X_train = TimeSeriesScalerMeanVariance().fit_transform(ts_days)

Entrée [986]: X_train.shape
   Out[986]: (100, 91, 1)
```

- **Clustering**

Dans cette partie nous allons faire une classification automatique des times series, on va commencer par un :

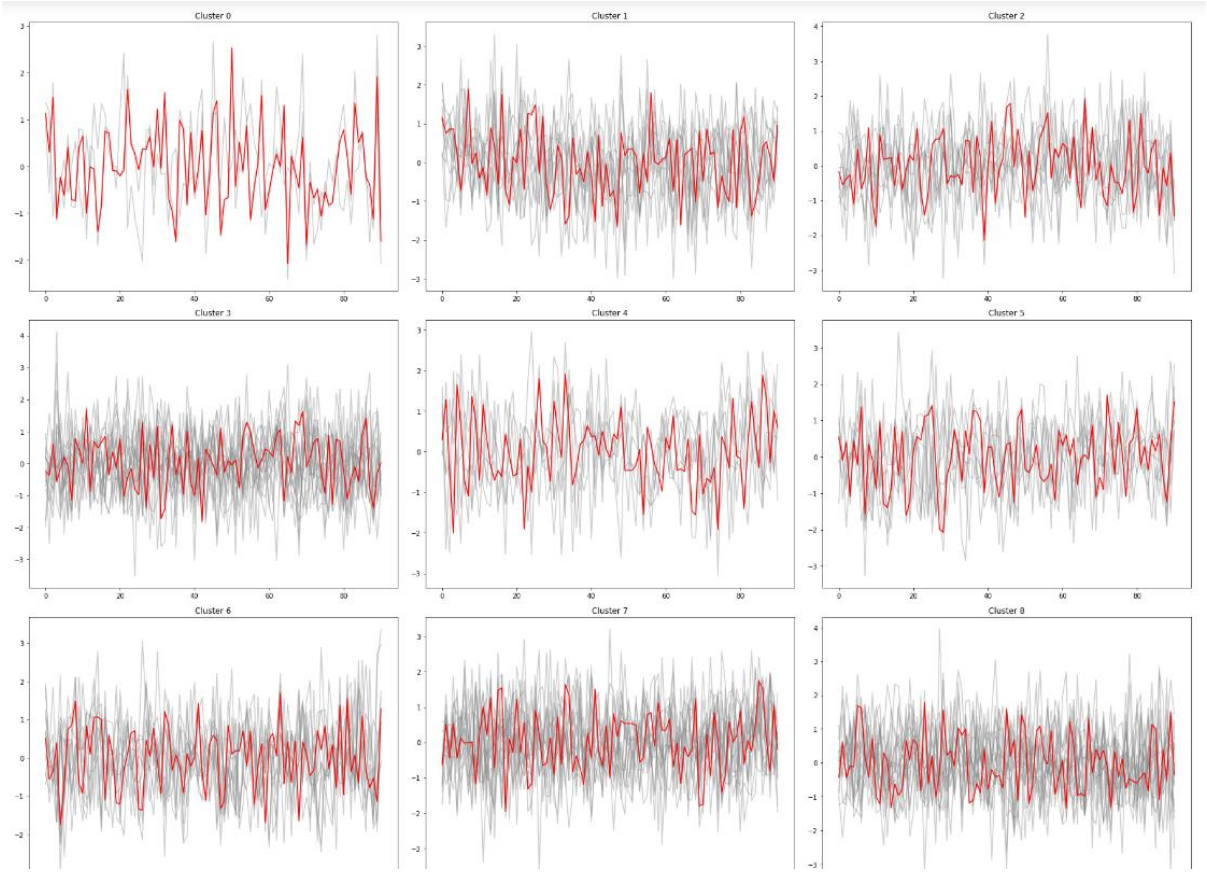
- **K-Means** : Metrics = (Dynamic time warping && Euclidian Distance)
- **SOM** (Self-organizing maps)
- **CAH** (clustering ascendant hiérarchique)
- **ACP** + K-Means

Pour chaque modèle, nous allons effectuer le clustering, calculer le nombre de chaque classe et trouver le nombre optimal de cluster en utilisant le ElboW.

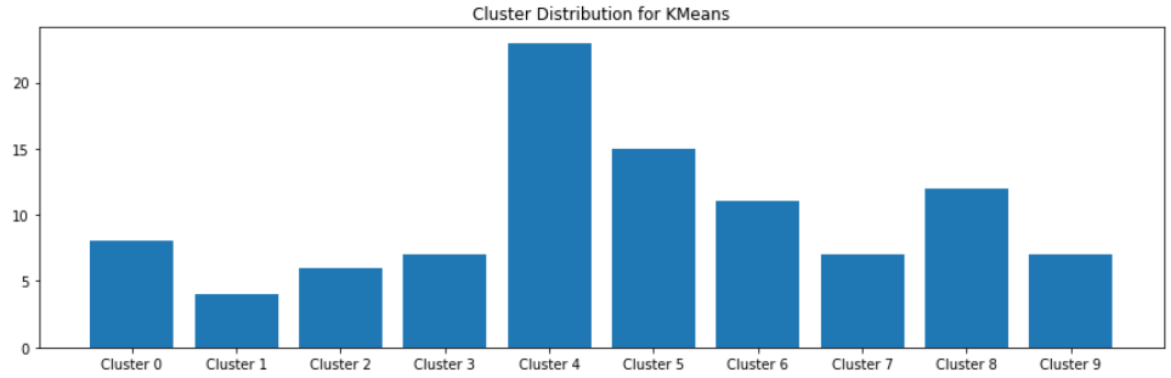
NB : Dans toutes les étapes on a choisi de travailler avec les données centrées-réduites

1- K-means :

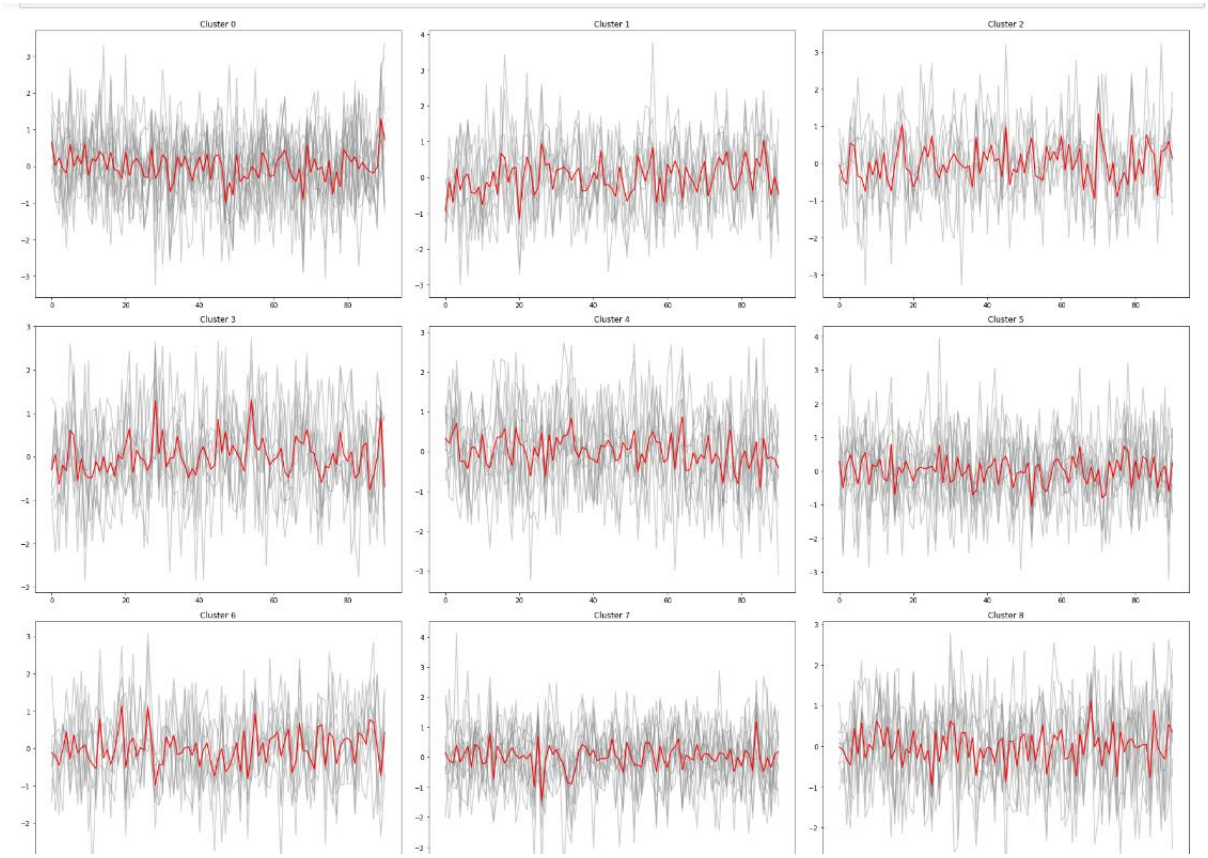
1-1 Dynamic time warping (DTW) distance :



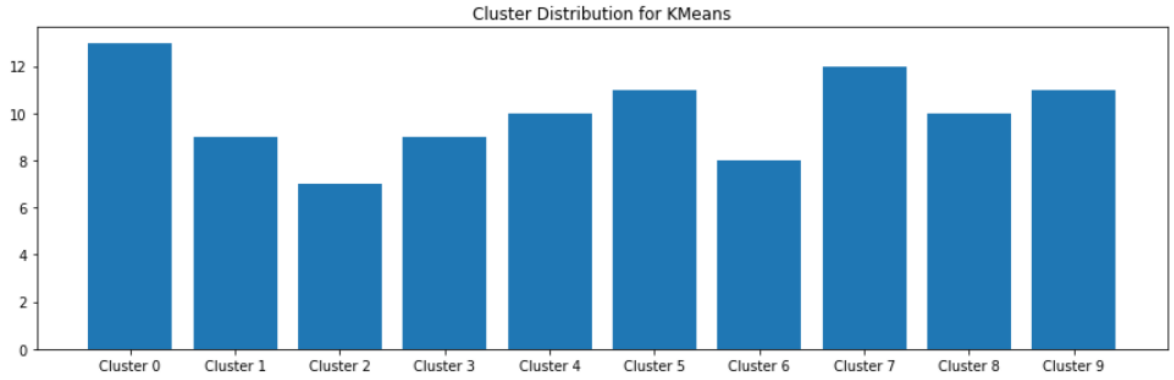
==> Dans les figures ci-dessus, on a affiché les series temporelles appartenant à chaque classe en gris et le rouge représente le centre de chaque cluster



1-2 Euclidian Distance:

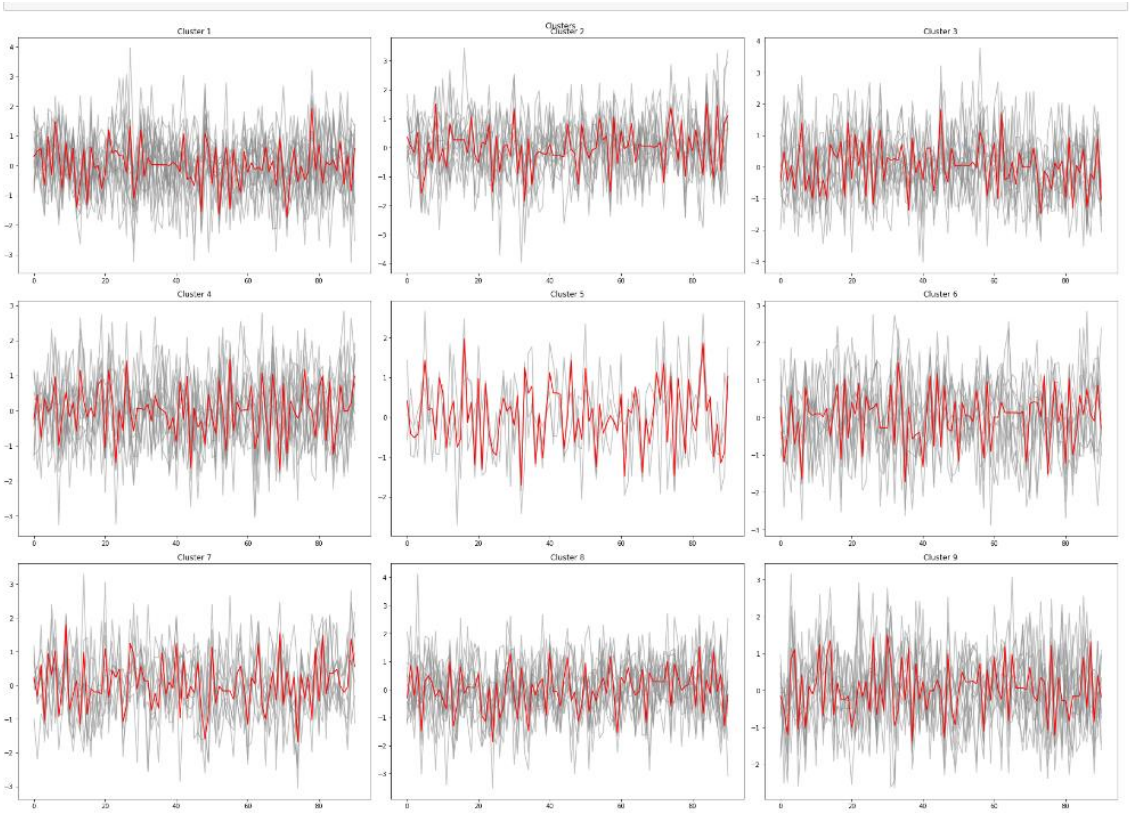


==> Dans les figures ci-dessus, on a affiché les series temporelles appartenant à chaque classe en gris et le rouge représente le centre de chaque cluster



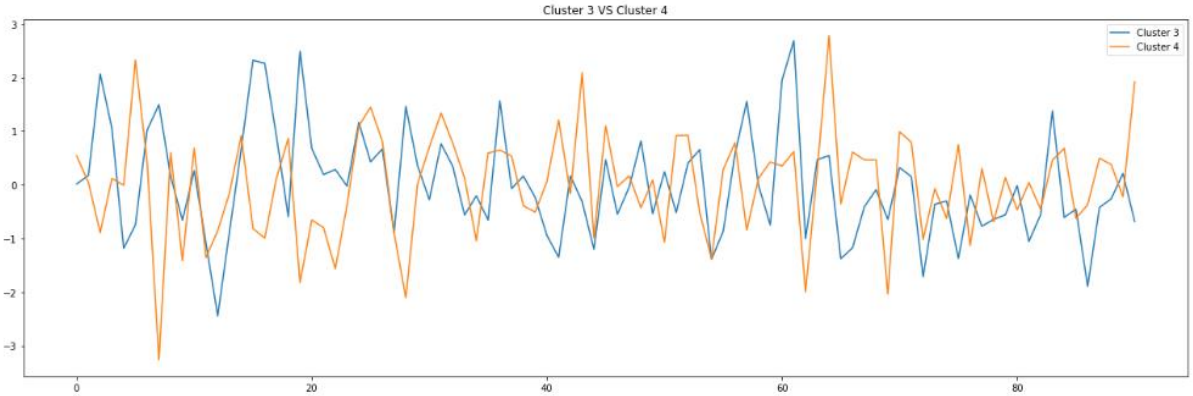
2- **SOM (Self-organizing maps) :**

Pour rendre la carte carrée, on a choisi de calculer la racine carrée de la taille de la carte qui est la racine carrée du nombre de séries pour le nombre de lignes et de colonnes de som.



- **<u>Comparaison des séries temporelle de classes différentes :</u>**

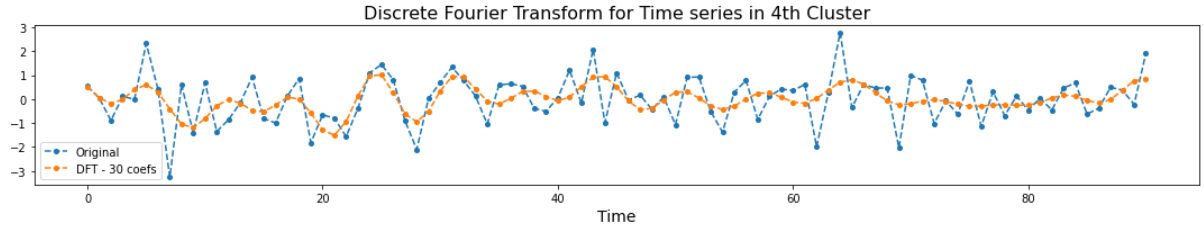
On peut remarquer que la série temporelle de la classe 1 consomme moins d'éléctricité que celle de la classe 2 tout au long des 91jrs, sauf dans des cas particuliers comme le jour 24.

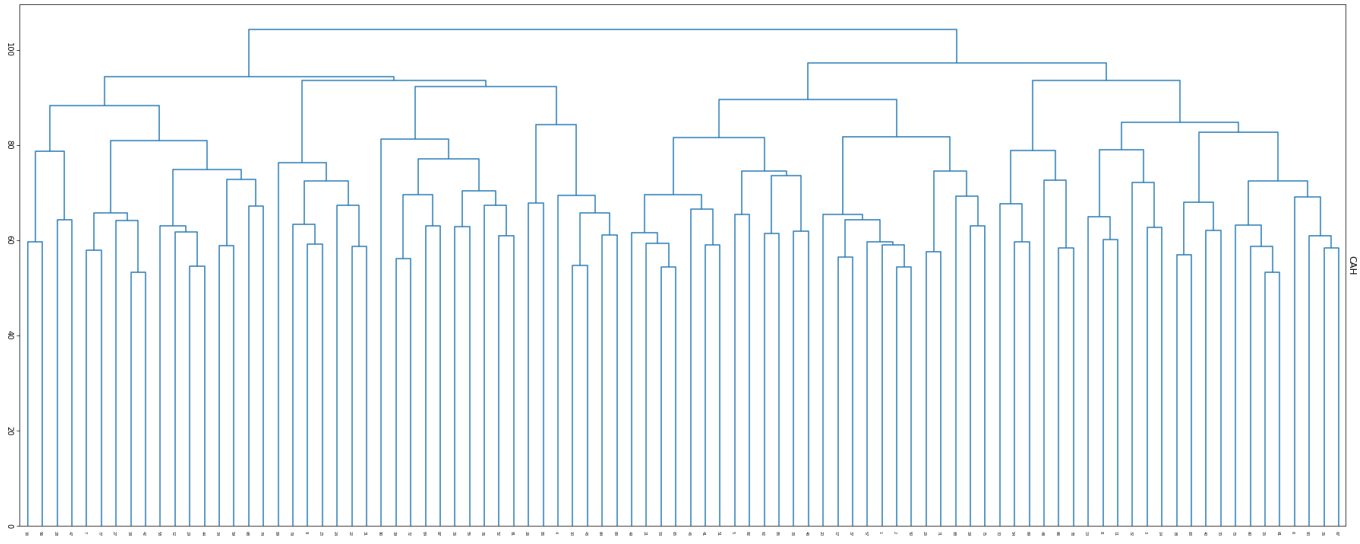


On peut remarquer que dans les premiers 40 jours, la série temporelle de la classe 3 consomme plus d'électricité que celle de la classe 4. Et dans le reste des jours, on remarque le contraire, le cluster 4 consomme plus d'électricité.

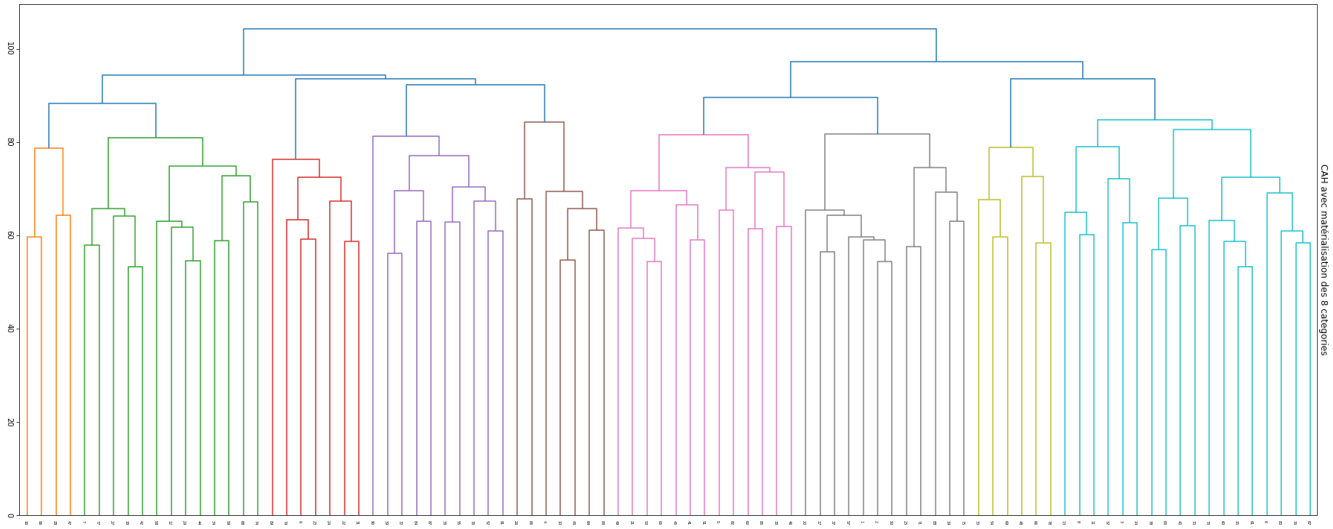
Remarque : On constate que ce n'est pas évident de trouver les caractéristiques entre les classes obtenues !!!

Clustering Ascendant Hierarchical

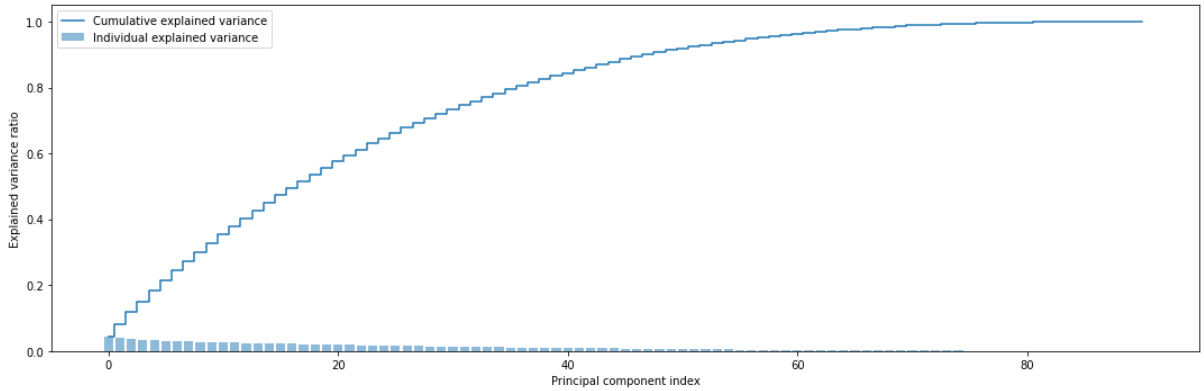




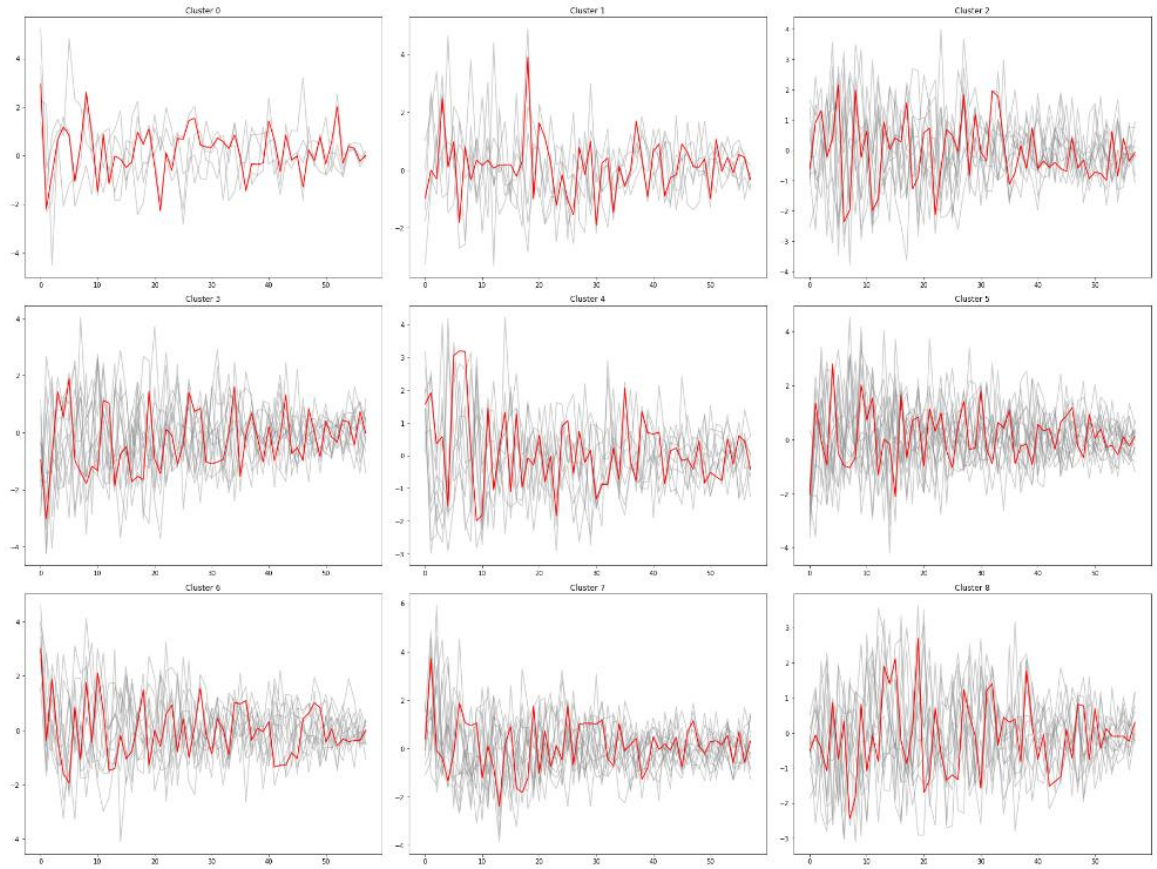
D'apres le critère d'agrégation Dmin, on aura 7 partitions , les différentes classes sont affichées dans le dendrogram au-dessous



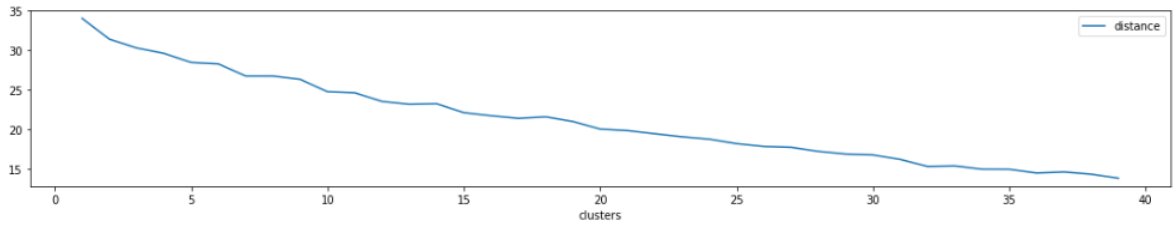
- **ACP (Analyse des composantes principales) :**



95% de l'information est capté par 58 composantes principales
Nous pensons donc à faire une classification automatique de notre nouvelle dataset avec le Kmeans(metric=DWT)



- **ElboW :**



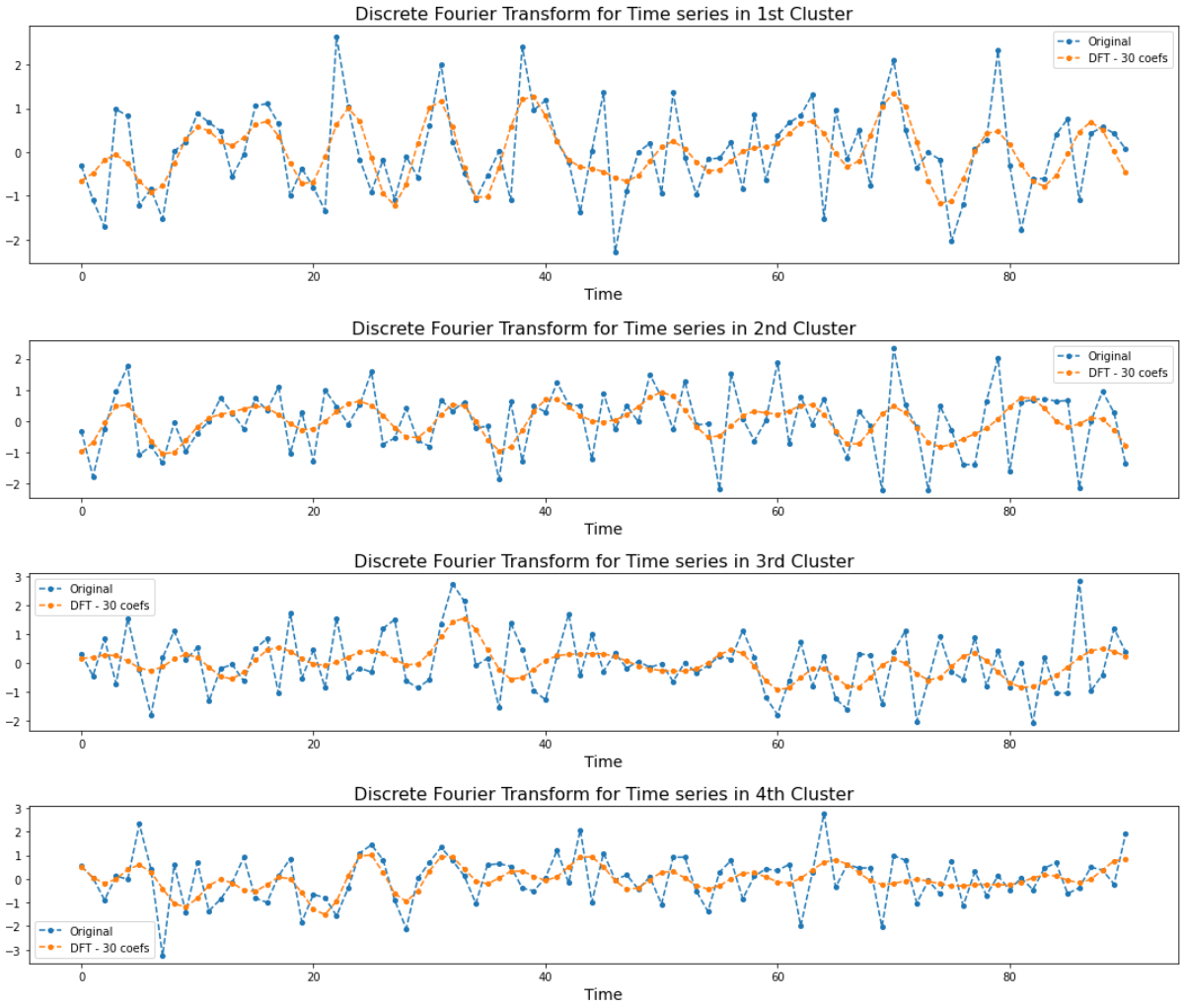
Avec le ACP, on a pu obtenir des figures intéressantes, mais il reste toujours délicat d'observer clairement le comportement de chaque classe. Il se peut que cette difficulté dépende aussi de l'ensemble de données qui n'est pas riche.

(Plus de détails dans le code)

Segmentation

- **Decomposition de Fourrier :**

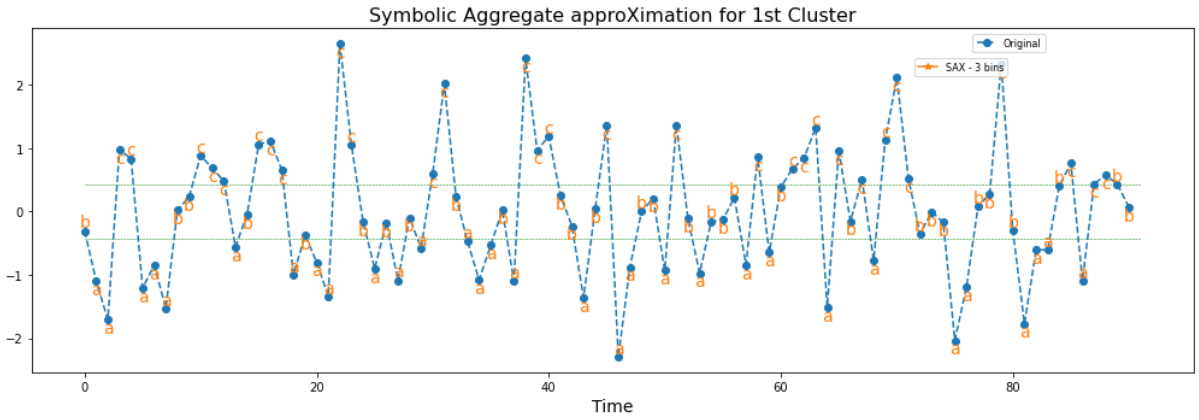
On a affiché la transformation de Fourrier pour une série temporelle des 4 premiers Clusters

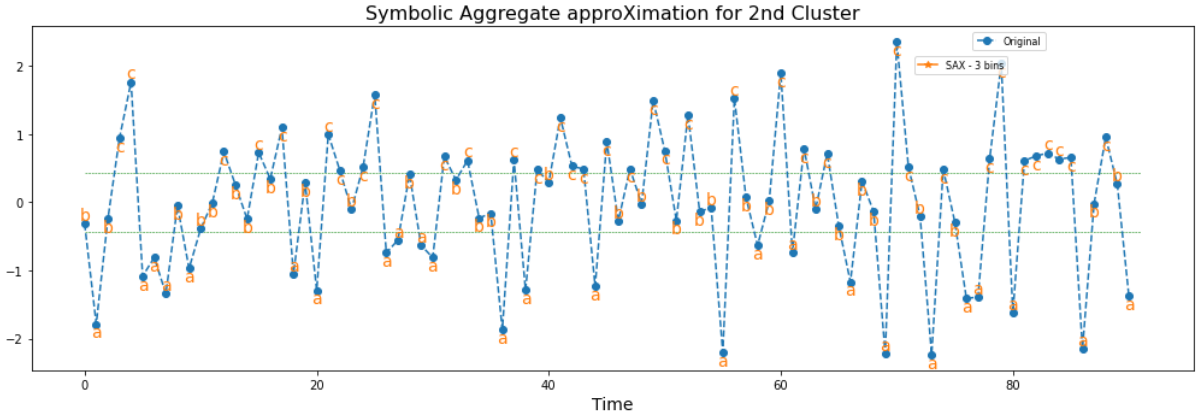


- **<u>SAX (Symbolic Aggregate Approximation) :</u>**

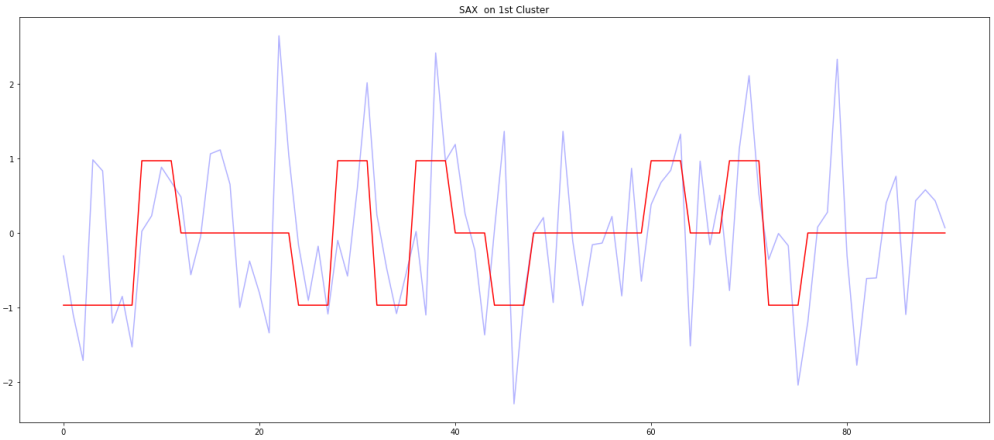
SAX est un moyen de transformer une série chronologique (une séquence de nombres) en une séquence de symboles. Cette implémentation prend un ensemble d'une ou plusieurs séries chronologiques comme entrée. Ensuite, il transforme la série chronologique en leur représentation SAX.

⇨ Le SAX nécessite que les données soient centré-réduire, et il est bien notre cas.





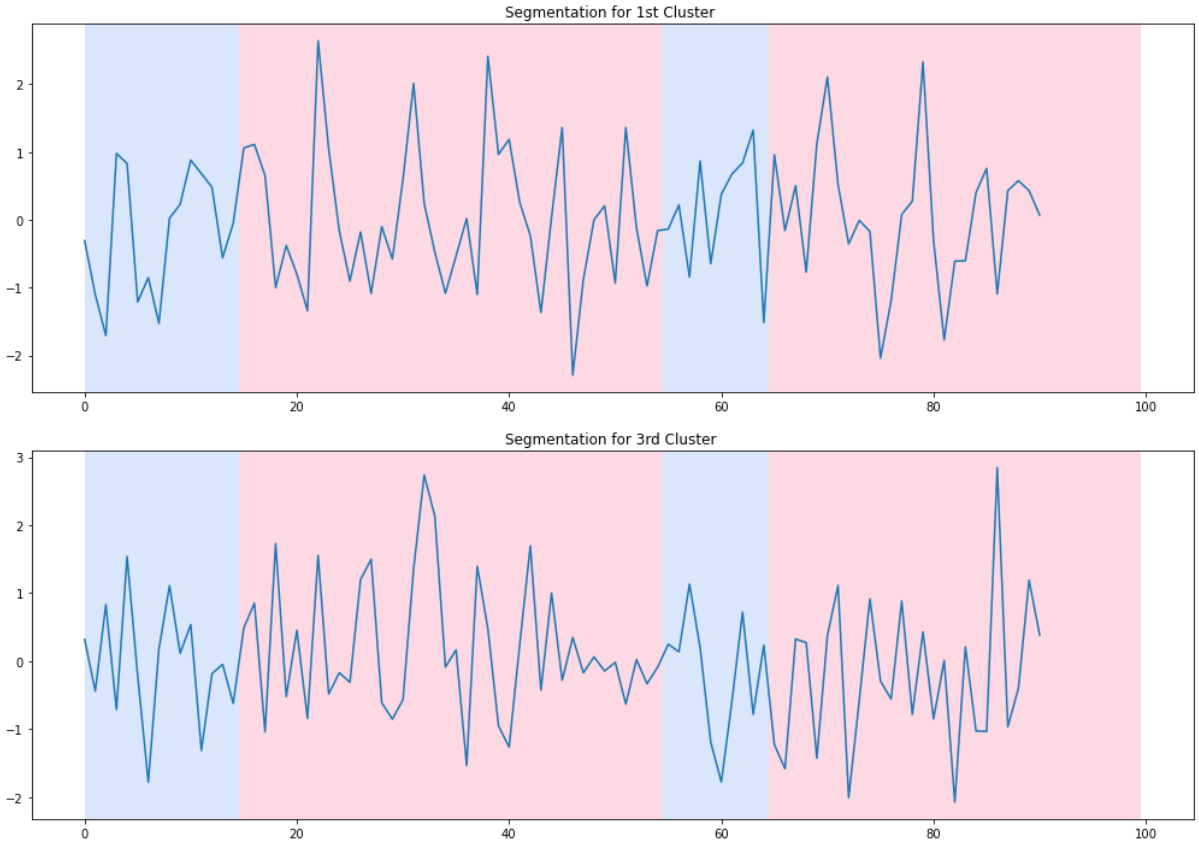
- **SAX using tslearn package :**



- **Package Ruptures :**

Ruptures est une bibliothèque Python pour la détection de points de changement hors ligne. Ce package fournit des méthodes pour l'analyse et la segmentation de signaux non stationnaires. Les algorithmes implémentés incluent la détection exacte et approximative pour divers modèles paramétriques et non-paramétriques. Ruptures se concentre sur la facilité d'utilisation en fournissant une interface bien documentée et cohérente.

Ce package utilise la méthode de recherche basée sur la programmation dynamique.



- ***PELT :***

Détection pénalisée des points de changement.

Pour un modèle et un niveau de pénalité donnés, il calcule la segmentation qui minimise la somme contrainte des erreurs d'approximation.