Intervenant	**Rakia Jaziri**
Intitulé TD/TP :	**Atelier : Apprentissage non supervisé avec Python**
Contenu	• **Réduction de dimensions (ACP)** • **Visualisation** • **Clustering** • **Evaluation de la qualité d'un clustering**

Dans cet atelier pratique, vous allez expérimenter des algorithmes de traitement de données pour répondre à différents problèmes liés à l'apprentissage non supervisé avec le langage **Python**.

Python est un langage de programmation très polyvalent et modulaire, qui est utilisé aussi bien pour écrire des applications comme YouTube, que pour traiter des données scientifiques. Par conséquent, il existe de multiples installations possibles de Python. L'utilisateur débutant peut donc se sentir dérouté par l'absence d'une référence unique pour Python scientifique. Le plus simple pour ce TP est d'installer, la suite scientifique Anaconda développée par l'entreprise Continuum (http://continuum.io/downloads.html). Anaconda rassemble tout le nécessaire pour l'enseignement de Python scientifique: le langage Python et ses modules scientifiques. Sur le plan des packages Python, vous allez utiliser **Scikit-learn**. Cette libraire montre dans cette situation tout son intérêt. La plupart des techniques récentes d'apprentissage sont en effet expérimentées avec Scikit-learn et le plus souvent mises à disposition de la communauté scientifique.

Pour plus de détails concernant :

- le langage Python vous pouvez aller sur le site suivant : http://www.python-course.eu/index.php
- la librairie Scikit-learn vous pouvez aller sur le site suivant : http://scikit-learn.org

Pour lancer le notebook Python, il faut taper la commande **jupyter notebook** dans votre dossier de travail. Une fenêtre va se lancer dans votre navigateur pour ouvrir l'application Jupyter. Créer un nouveau notebook Python et taper le code suivant dans une nouvelle cellule :

```
import numpy as np
np.set_printoptions(threshold=np.nan)
import pandas as pd
import warnings
import matplotlib.pyplot as plt
warnings.filterwarnings('ignore')
```

I. Réduction de dimensions et Visualisation des données

Le fichier **"villes.csv"** comporte 32 villes françaises décrites par les températures moyennes dans les 12 mois de l'année.

L'objectif dans cette partie est de représenter graphiquement le plus d'informations possibles contenues dans ce fichier de données et de déceler une éventuelle segmentation topologique des villes.

1. Importer ce jeu de données avec la librairie **pandas** (*c.f.* ***read_csv***)

```
data = pd.read_csv('./villes.csv', sep=';')
X = data.ix[:, 1:13].values
labels = data.ix[:, 0].values
```

2. Réaliser une Analyse en Composantes Principales (module **PCA** de Scikit-learn) sur ce jeu de données centrées réduites (**StandardScaler**)
   - Quel est le nombre d'axes à retenir pour conserver un minimum de 70% de l'information représentée dans le nuage initial.
   - Donner une interprétation des deux premiers axes principaux.
   - En suivant le code suivant, donner une visualisation graphique des villes projetées dans le plan principal.

   X_pca étant la matrice des données transformées par l'ACP, **labels** étant le vecteur contenant le nom des instances (ici les villes).

   ```
   import matplotlib
   plt.scatter(X_pca[:, 0], X_pca[:, 1])
   ```

```
for label, x, y in zip(labels, X_pca[:, 0], X_pca[:, 1]):
    plt.annotate(label, xy=(x, y), xytext=(-0.2, 0.2), textcoords='offset points')
plt.show()
```

3. Faire de même pour le fichier **"crimes.csv".** Il s'agit des statistiques de criminalité dans 50 états américains. Dans chaque état, sept types de crimes ou délits sont repérés par leurs nombres annuels de faits constatés rapportés sur 100 000 habitants : *meurtres*(**Meutre**), *enlèvements*(**Rapt**), *vols avec violence*(**Vol**), *agressions*(**Attaque**), *viol* (**Viol**), *vols peu importants* (**Larcin**), *vols de voitures* (**Auto_Theft**).

II. Clustering

La classification (les Anglos-saxons parlent de clustering) est l'opération statistique qui consiste à regrouper des objets (individus ou variables) en un nombre limité de groupes, les classes (ou segments, ou clusters) , qui ont deux propriétés. D'une part, ils ne sont pas prédéfinis par l'analyste mais découverte au cours de l'opération, contrairement aux classes du classement. D'autre part, les classes de la classification regroupent les objets ayant des caractéristiques similaires et séparent les objets ayant des caractéristiques différentes (homogénéité interne et hétérogénéité externe), ce qui peut être mesuré par des critères telle l'inertie interclasse et l'inertie intraclasse.

Nous allons étudier dans la suite deux approches de clustering (*k-moyennes* **"KMeans"** et la *classification Ascendante Hiérarchique* **"AgglomerativeClustering"** du package sklearn.cluster) que nous allons appliquer sur le jeu de données des villes.

1. Appliquez la procédure **KMeans** sur ce jeu de données pour obtenir **3** clusters
   - Donner une visualisation graphique des villes projetées dans le plan principal. Les villes de chaque cluster devraient avoir une couleur différente des villes des autres clusters (voir code ci-dessous).

   X_pca étant la matrice des données transformées par l'ACP, **labels** étant le vecteur contenant le nom des instances (ici les villes), **clustering** étant le clustering obtenu.

   ```
   colors = ['red','yellow','blue','pink']
   plt.scatter(X_pca[:, 0], X_pca[:, 1], c= clustering, cmap=matplotlib.colors.ListedColormap(colors))
   for label, x, y in zip(labels, X_pca[:, 0], X_pca[:, 1]):
       plt.annotate(label, xy=(x, y), xytext=(-0.2, 0.2), textcoords='offset points')
   plt.show()
   ```

2. Appliquez la procédure **AgglomerativeClustering** sur ce jeu de données pour obtenir trois clusters avec différentes méthodes d'agrégation (il faut essayer **ward** et **average**).
   - Donner à chaque fois une visualisation graphique des villes projetées dans le plan principal. Les villes de chaque cluster devraient avoir une couleur différente des villes des autres clusters.

Nous allons maintenant déterminer la meilleure partition (nombre de clusters) pour la méthode **KMeans**. Pour cela, nous allons utiliser le critère **"Silhouette index"** (**metrics.silhouette_score** de scikit-learn).

3. Utiliser ces deux indices dans une boucle de 4 itérations au maximum (voir code ci-dessous). Les 4 itérations correspondent aux 4 partitions possibles *i.e.* en 2, 3, 4 et 5 classes issues de **KMeans**. Déduire la meilleure partition qui correspond à un indice maximal pour l'indice **Silhouette**.

```
from sklearn import metrics
for i in np.arange(2, 6):
    clustering = KMeans(n_clusters=i).fit_predict(X)
    print(metrics.silhouette_score(X, clustering,metric='euclidean'))
    print()
```