Python 3 – Outils de Data analyse

Manipulation de données avec Numpy

- Le module (bibliothèque) NumPy (Numeric Python) fournit des routines de base pour manipuler de grands tableaux et matrices de données numériques.
- Les tableaux sont similaires aux listes en Python, sauf que tous les éléments d'un tableau sont de même type.
- Les tableaux « numpy » sont plus performants (en rapidité et gestion de la volumétrie) que les collections usuelles de Python comme les listes
- Un tableau est ensemble d'éléments ordonnées et indexés (à partir de 0)
- Pour créer un tableau on utilise la classe **array**().
- Le constructeur de la classe array prend 1 ou 2 arguments :
  - La liste des éléments
  - La liste à convertir dans le tableau et le type. (float, str, …)
- L'opérateur **in** peut être utilisée pour tester si une valeur est présente dans un tableau.

Les bibliothèques python pour la data analyse

- **Numpy / Pandas**
  Manipulation, analyse et gestion de données sous forme de DataFrame
- **Scikit-learn (sklearn)**
  Fonctions de « machine learning », outils d'apprentissage supervisé et non supervisé, ainsi que des outils de prétraitement et d'évaluation
- **Matplotlib**
  Visualisation de données en 2D, permet de créer des graphiques variés (courbes, histogrammes, diagrammes)
- **Seaborn**
  Extension de Matplotlib dédiée à la création de visualisations statistiques attractives et informatives avec une syntaxe simplifiée.

Creation de tableau

- Créer des tableaux manuellement :
  - La méthode **array** permet de créer des tableaux numpy à partir des scalaires, des listes ou des tuples

- Créer des tableaux paramétrables :
  - **Tableaux constants :** les méthodes **zeros** et **ones** retourneent des tableaux entièrement constitués de 0 et de 1
  - **Tableaux aléatoires :** le sous-module **np.random** possède deux méthodes pour définir des tableaux de nombres au hasard : **random** et **randint**.
  - **Tableaux en progression arithmétique :** La méthode **arange** produit des tableaux unidimensionnels dont les éléments sont en progression arithmétique

Accès aux éléments d'un tableau

- Chaque élément d'un tableau est identifié par un tuple (i , j), où i représente la ligne et j la colonne
  - On accède à la valeur des éléments d'un tableaux A unidimensionnel par A[i]
  - On accède à la valeur des éléments d'un tableaux B bidimensionnel par B[i , j]

- Pour extraire les données (slicing), on utilise la syntaxe **A[début : fin : pas]**
  - Dans l'instruction [début:fin:pas] ces arguments peuvent être omis :
    - par défaut l'indice de début vaut 0 (le 1er élément),
    - l'indice de fin est celui du dernier élément et
    - le pas vaut 1
  - On accède à toutes les valeurs d'une ligne ou une colonne par B[i , :] ou B[: , j]

Explorer et filtrer un tableau

- Pour explorer dans un tableau les données qui vérifient certaines conditions, numpy nous permets de faire des recherches en appliquant des filtres booléens
- Les opérations de comparaisons sont aussi applicables sur toutes les données de l'array et nous retourne le résultat par un array de booléens de même taille
- La classe array possède aussi des méthodes qu'on peut utiliser comme fonctions d'agrégations sur tout le tableau ou ligne par ligne (Avec axis = 0) ou colonne par colonne (Avec axis = 1)

Calcul sur les tableaux

- NumPy permet de réaliser des calculs arithmétiques sur les matrices et les vecteurs :
  - **np.dot()** multiplication matricielle
  - **np.inner** produit scalaire
  - **np.diag** création matrice diagonale
  - **np.transpose** matrice transposée

- Pour plus de fonctions spécifiques on fait recours au sous-module **np.linalg** :
  - **np.linalg.det** déterminant de la matrice
  - **np.linalg.inv** inverse de la matrice
  - **np.linalg.solve** résolution du sytème linéaire A.x = B
  - **np.linalg.eig** valeurs propres, vecteurs propre
  - **np.linalg.eigvals** valeurs propres

Manipulation de données avec Pandas

- Pandas est spécialisée dans l'analyse des données.
- Pandas se base sur NumPy à la différence que Pandas permet d'analyser des données **hétérogènes** alors que NumPy n'est utilisé que pour des données numériques et **homogènes**
- Pandas utilise deux types de structures de données :
  - **Les Series** : objet unidimensionnel de type tableau contenant une séquence de valeurs avec des étiquettes associées appelées index.
  - **Des DataFrames** : objet bidimensionnel de type tableau se comportant comme un dictionnaire dont les clefs sont les noms des colonnes et les valeurs sont des séries, partageant le même index

Créer un DataFrame, Import / Export

- On peut créer un DataFrame à partir de:
  - **Un Dictionnaire :** il est facile de convertir un dictionnaire en DataFrame, les clefs seront les noms des colonnes et les valeurs du dictionnaire joueront le rôle des Séries
  - **Un Array Numpy :** C'est moins facile qu'avec les Dict, car il faut rajouter le nom des indices et des colonnes quand on le crée
  - **Une base de données :** Pandas possède plusieurs outils pour importer des données à partir de différents formats (csv, excel, sql, html, etc.) et les convertir en DataFrame
- Une fois les données ont été analysées avec les DataFrames, on peut les exporter vers différents formats de fichiers (csv, excel, etc.)

Exploration des données

- Pour jeter un coup d'œil sur un DataFrame on peut utiliser la méthode **df.head()**, ce qui affiche les première lignes du tableau, ou **df.tail()** et **df.info()**
- Pour accéder à un sous-ensemble d'un dataframe
  - On sélectionne les noms des lignes et des colonnes en utilisant la méthode **.loc**
  - On sélectionne les numéros des lignes et des colonnes en utilisant la méthode **.iloc**
- Il est également possible d'accéder à une colonne
  - **df['nom_colonne'],**
  - ce qui renvoie la Serie correspondant au nom de la colonne

Modifier les colonnes/lignes

- Pour renommer les colonnes ou les lignes d'un DataFrame, on peut utiliser :
  - **df.columns** = ['A_', 'B_', 'C_', 'D_'] : modifie les noms des colonnes
  - **df.rename**(**columns** = {'A_':'A_new', 'B_':'B_new'}, inplace = True): modifie les noms des colonnes choisies par ceux du dictionnaire
  - **df.rename**(**index** = {'a1':'a1_new', 'a2':'a2_new'}, inplace = True): modifiera les noms des lignes choisies par ceux du dictionnaire

- Pour réordonner des colonnes d'un DataFrame, on peut utiliser :
  - **df.reindex(columns = ['C', 'D', 'B_new', 'A_new'])**: renvoie le Dataframe réordonné
  - **df[['C', 'D', 'B_new', 'A_new']]**: renvoie aussi le Dataframe réordonné

Convertir des Colonnes

- Conversion d’une valeur numérique
  - Attention en France les montants ont une virgule pour les décimales, il faut la remplacer par un point avant de convertir la valeur en réel (float)

```
dfFacture['TotalHT'] = dfFacture['TotalHT'].str.replace(',', '.') .astype(float)
```

- Conversion d’une date

```
dfFacture[‘maDate'] = pd.to_datetime(dfFacture[‘maDate'], format='%d/%m/%Y’)
```

Modifier les colonnes/lignes

- Pour supprimer une colonne d'un DataFrame
  - **del** df['E'] : permet de supprimer la colonne E
  - **df.drop**(columns = ['A', 'C'], inplace = True) : supprime plusieurs colonnes en même temps (ou des lignes)

- Pour rajouter une colonne à un DataFrame, on peut :
  - Donner une Serie dont les noms des individus sont les mêmes que ceux du dataframe :
    df['E'] = pd.Series([1, 0, 1], index = ['a1', 'a2', 'a3']) :
  - Insérer une colonne à un emplacement donné :
    df.insert(0, 'E', [1, 2, 3])
  - Rajouter une nouvlle colonne à partir d'une combinaison de colonnes existantes :
    df['F'] = df['C_'] + df['D_']

Nettoyer les données

- Pour modifier les valeurs d'une colonne
  - df[df['A'] > 5] = 6 # remplace les valeurs supérieures à 5 par 6
  - Newdf = df.loc[df['A']> 5] # filtre les lignes où 'A' > 5
  - Newdf = df.loc[(df['A'] >= 3 & df['A'] <= 5)] # filtre les lignes où 'A' est entre 3 et 5
  - df.loc[df['A']> 5 , 'B'] = 7 # identifie les lignes où 'A' > 5 et remplace ceux de 'B' par 7
  - df.loc[0:2, 'B'] = 7 # remplace une partie (slice) du DataFrame par la valeur 7
  - df['colonne'].str.replace('nom', 'autre') # remplace une Valeur par une autre
  - df['colonne'].astype(float) # change le type de la colonne

- Pour supprimer les valeurs non définies NaN : **df.dropna()**
- Pour remplacer les valeurs non définies NaN: **df.fillna(valeur)**

Concaténation et jointure

- La concaténation (ou juxtaposition) de deux DataFrames permet de les regrouper selon leur index un à côté de l'autre sur l'axe horizontal ou vertical, on utilise la méthode pd.concat([df1, df2])
  - Pour préciser l'axe de concaténation, on utilise pd.concat([df1, df2], axis = 1)

- La jointure donne plus d'options: pd.merge(df1, df2)
  - Pour indiquer la colonne sur laquelle on veut faire la jointure : **df1.merge(df2, on = ['A'])**
  - Pour les jointure et rajoute NaN dans les cases vides : **df1.merge(df2, on = ['A'], how = 'outer')**

Agrégation de données

- Il est possible de récupérer les valeurs uniques d'une colonne
  - Df['A'].unique()
- Pour compter le nombre de valeur distinct d'une colonne
  - Df['A'].value_counts()

- L'agrégation permet d'accéder à des sous-DataFrame associés à chaque item de la variable de regroupement :
  - Pour effectuer une agrégation d'une colonne 'A' : **df.groupby('A')**

- Une fois l'agrégation est faite, Il est possible d'appliquer explicitement différents traitements sur ces sous-ensembles de données.
  - Pour appliquer une fonction sur ces agrégats : **df.groupby('A').mean(numeric_only=True)**
  - Pour appliquer plusieurs fonctions simultanément : **df.groupby('A').agg(['mean', 'std'])**

Présentation de matplotlib

- Créé par John D. Hunter en 2003, distribuée sous licence BSD
- Permet de créer des graphiques et des visualisations de données
  - diagrammes en barres et circulaires,
  - histogrammes,
  - nuages de points,
  - graphiques de contour et en 3D
- La flexibilité et la personnalisation des graphiques
  - Axes , étiquettes, titres
  - couleurs, styles de lignes, polices
- Exporter des graphiques dans une variété de formats, tels que PNG, PDF, SVG

Exemples de Graphique avec mathplotlib

```
total_par_client = df.groupby(‘Refclient')['TotalHT'].sum()

# Création d'un graphique en barres
total_par_client.plot(kind='bar', figsize=(10, 6))

# Ajouter un titre et des étiquettes d'axes
plt.title('Montant total facturé par client')
plt.xlabel('Client ID')
plt.ylabel('Montant total (€)')

# Affichage du graphique
plt.show()
```

Les étapes du traitement de la donnée

- Observation des données
  - nb ligne, colonnes, minimum, maximum et moyenne
  - volume des valeurs des différentes colonnes
- Prétraitement des données
  - Regroupement des données
  - Gestion des données erronées
  - Gestion des données manquantes (usage de la moyenne)
- Visualisation graphique des données
  - Correction des données (suppression des bords)

Etapes du traitement de clustering

- Remplacement des informations alphabétique en valeurs numériques
  - (ex single:1, married: 2)
- Définir une matrice de corrélation
  - Relation linéaire entre les variables (valeurs en 1 et -1)

    +1 : les valeurs sont parfaitement corrélées

    0 : les valeurs ne sont pas du tout corrélées

    -1 : les valeurs sont inversement corrélées
  - Une corrélation forte correspond à des valeurs absolues entre 0.7 et 1
- Sélection de variables dont on souhaite analyser la corrélation
  - Détermination du nombre de cluster
  - Génération du graphique de visualisation des clusters