TP 1 : Introduction aux librairies Python pour la fouille de données

`Numpy`, `pandas`

IMT Atlantique – FIL A3

Apprentissage Automatique

Objectifs :
`NumPy` et `Pandas` sont des bibliothèques logicielles très utiles dans la résolution de problèmes de modélisation à partir de données. Cette séance de TP vise à vous donner les connaissances de base nécessaires mais est loin de couvrir toutes les fonctionnalités de ces deux librairies. Pour aller plus loin, vous êtes invités à regarder les références en ligne indiquées ci-dessous.

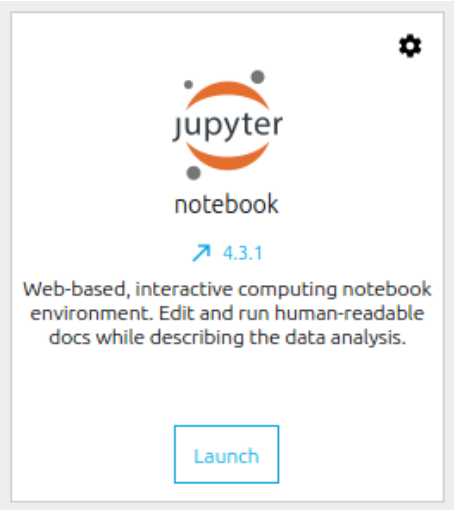
Références utiles :
- Documentation `NumPy` [1]
- Documentation `Pandas` [2]
- Documentation `Matplotlib` [3]

1 Anaconda

1.1 Installation

Anaconda est une distribution Python, faite pour la Data Science. Elle contient tous les packages dont nous aurons besoin : `Matplotlib`, `NumPy`, `Pandas` et le notebook Jupyter, que je vous conseille d'utiliser. La procédure d'installation est détaillée à l'URl `https://docs.anaconda.com/anaconda/install/`. Une fois l'installation terminée, il suffit de lancer le programme Jupyter :
- Sous Linux Ubuntu, ouvrez une console et lancez la commande `anaconda-navigator`.
- Sous Windows, lancez Anaconda Navigator en cliquant sur démarrer > (Programmes) > Anaconda > Anaconda Navigator.

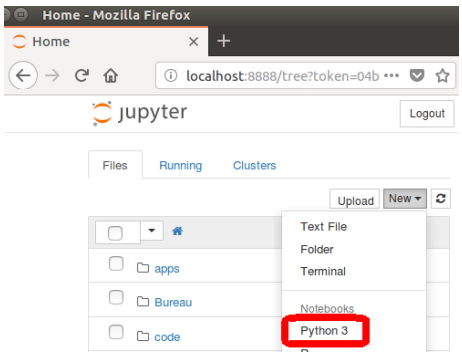
Une fois le navigateur Anaconda lancé, plusieurs applications vous sont proposées. Cliquez sur Jupyter :



1. `https://numpy.org/doc/stable/user/quickstart.html`
2. `https://pandas.pydata.org/pandas-docs/stable/reference/index.html#api`
3. `https://matplotlib.org`

1.2 Créer un nouveau notebook

A partir de la fenêtre principale de Jupyter, cliquez sur "New" puis sur "Python 3", comme ceci :



Votre navigateur devrait vous afficher le notebook créé dénommé `Untitled`. Les commandes Python doivent être tapées dans la case devant l'étiquette `In [ ]`. Vous pouvez taper plusieurs instructions à la fois. Vous pouvez aussi définir des fonctions. Les variables générées dans chaque case seront disponibles dans toutes le cases du notebook. Pour exécuter une série de commandes, il suffit d'appuyer sur `Shift+Entree`.

2 Calcul avec `NumPy`

`NumPy` est une bibliothèque de calcul scientifique dédié à la manipulation de matrices et de tableaux en multiple dimensions. Les tableaux `NumPy` sont au cœur de presque tout l'écosystème de data science en Python. L'utilisation de fonctionnalités de `NumPy` commence par l'importation de cette librairie

```
import numpy as np
```

Le type de base dans `NumPy` est le tableau unidimensionnel ou multidimensionnel composé d'éléments de même type. La classe correspondante est `ndarray`. les Principaux attributs de `ndarray` sont :

ndarray.ndim	dimension du tableau (nombre d'axes)
ndarray.shape	tuple d'entiers indiquant la taille dans chaque dimension
ndarray.size	nombre total d'éléments du tableau
ndarray.dtype	type de (tous) les éléments du tableau
ndarray.data	les données du tableau

2.1 Création de tableaux

De nombreuses méthodes de création de tableaux sont disponibles. D'abord, un tableau peut être créé à partir **d'une liste** Python, à condition que tous les éléments soient de même type (le type des éléments du tableau est déduit automatiquement du type des éléments de la liste ou tuple).

```
import numpy as np
panda_numpy = np.array([100,5,20,80])
print(panda_numpy)
panda_numpy.dtype
```

Les listes sont transformées en simples tableaux unidimensionnels, les listes de listes (de même taille) en tableaux bidimensionnels, et ainsi de suite. L'exemple ci-dessous montre comment créer une liste de pandas :

```
famille_panda = [
     [100, 5 , 20, 80], # maman panda
     [50 , 2.5, 10, 40], # bébé panda
     [110, 6 , 22, 80], # papa panda
]
famille_panda_numpy = np.array(famille_panda)
print(famille_panda_numpy)
famille_panda_numpy[2,0]
```

On accède aux éléments du tableau `NumPy` avec la même syntaxe que pour accéder aux éléments d'une liste Python. Pour un tableau multidimensionnel (par exemple, une matrice), il suffit d'indiquer la liste des indices de l'élément recherché.

2.1.1 Création de tableaux directement

Plutôt que de spécifier manuellement toutes les valeurs du tableau, `NumPy` dispose de fonctions préintégrées pour générer des tableaux et des matrices courantes.

```
# Un tableau de longueur 10, rempli d'entiers qui valent 0
np.zeros(10, dtype=int)
# Un tableau de taille 3x5 rempli de nombres à virgule flottante de valeur 1
np.ones((3, 5), dtype=float)
# Un tableau 3x5 rempli de 3,14
np.full((3, 5), 3.14)
# Un tableau rempli d'une séquence linéaire commençant à 0 et qui se termine à 20, avec un pas de 2
np.arange(0, 20, 2)
# un tableau dont les éléments sont générés aléatoirement
np.random.random((3, 3))
# La matrice identité de taille 3x3
np.eye(3)
```

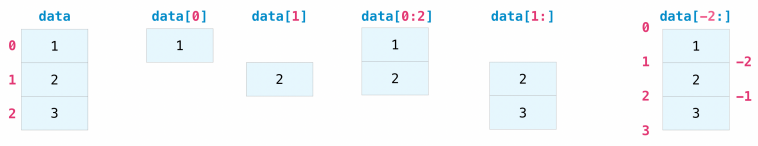
Les tableaux peuvent être redimensionnés en utilisant `reshape` :

```
tr = np.arange(20)
print(tr)
# Réordonne le tableau en une matrice à 4 lignes et 5 colonnes
tr.reshape(4,5)
```

2.1.2 Accès aux composantes d'un tableau, extraction de parties d'un tableau

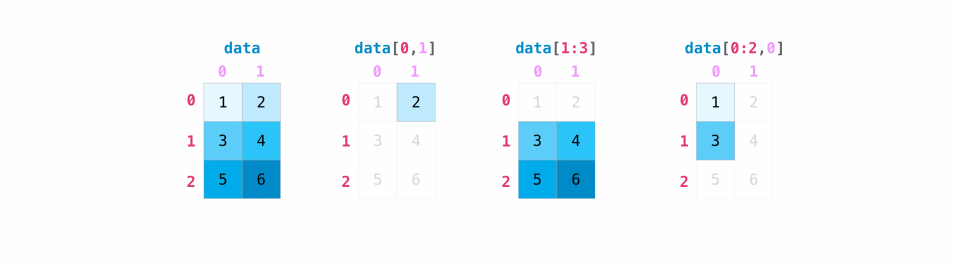
Pour accéder à un ensemble d'éléments d'un tableau, il est possible de combiner les opérateurs `[]` et `:`. La syntaxe suit une règle simple : `x[début:fin:pas]`.

```
data = np.array([1, 2, 3])
data[1]
data[0:2]
data[1:]
data[-2:]
```



On peut procéder de la même façon pour les tableaux multidimensionnels :

```
data = np.array([[1, 2], [3, 4], [5,6]])
# Extraction de la première ligne
print(data[0,:])
# Extraction des deux premiers éléments de la colonne 1
data[0:2, 0]
```



> **Question**
> Faites l'extraction des colonnes d'indice impaire de `famille_panda_numpy`.

Il est possible d'ajouter un tableau unidimensionnel comme colonne à un tableau bidimensionnel. `newaxis` permet de créer une nouvelle dimension dans un tableau existant :

```
from numpy import newaxis
print(data[:,newaxis])
```

3 Manipulation de données avec `Pandas`

Avec `NumPy`, la librairie `Pandas` fait partie des librairies de base pour la data science en Python. Elle fournit des structures de données puissantes et simples à utiliser, ainsi que les moyens d'opérer rapidement des opérations sur ces structures. Dans cette section, nous nous intéresserons à l'objet phare de cette librairie, le **DataFrame**.

Une DataFrame est une structure de données étiquetée en 2 dimensions avec des colonnes de types potentiellement différents. Elle permet de stocker et de manipuler des données tabulaires, telles que des données stockées dans des feuilles de calcul ou des bases de données.

3.1 Instanciation d'un objet DataFrame

Pour instancier un tel objet (et pour lui donner de la donnée), on lui transmet une liste de rang 2, c'est à dire une liste de listes. Comme la librairie `Pandas` se base en grande partie sur la librairie `NumPy` dans son fonctionnement interne, on peut alors transmettre à l'objet DataFrame de la donnée au format `ndarray` et même indiquer les noms de colonnes et les noms des lignes :

```
import pandas as pd
famille_panda_df = pd.DataFrame (famille_panda, index =['maman', 'bebe', 'papa'],
                                 columns = ['pattes', 'poil', 'queue', 'ventre'])
famille_panda_df
```

Voici quelques petites fonctionnalités des Dataframes :

1. Pour accéder à une colonne de la table, il suffit d'utiliser une des deux syntaxes suivantes :
   — `famille_panda_df.ventre`
   — `famille_panda_df["ventre"]`

   > **Note**
   > L'objet que renvoie `famille_panda_df["ventre"]` est de type `pandas.Series`. Pour obtenir les valeurs de la colonne ventre au format `numpy`, il faut saisir `famille_panda_df["ventre"].values`.

2. On peut accéder à une ligne de la table, soit par sa position, ou par par son nom :
   — `famille_panda_df.loc["papa"]` ➠ indexation par label
   — `famille_panda_df.iloc[2]` ➠ indexation par position

   > **Question**
   > Faites l'extraction des colonnes `pattes` et `queue`.

3. Les fonctions `tail` et `head` permettent d'afficher respectivement les premier et les derniers éléments du DataFrame :
   — `famille_panda_df.head(1)`
   — `famille_panda_df.tail(2)`
4. Les fonctions ci-dessous permettent d'accéder à certaines informations utiles du DataFrame :
   — `famille_panda_df.columns` : les noms des colonnes,
   — `famille_panda_df.columns.values` : le nom des colonnes sous forme d'array `numpy`,
   — `famille_panda_df.index` : les noms des lignes (individus),
   — `famille_panda_df.index.values` : les noms des lignes (individus) sous forme d'array `nympy`,
   — `famille_panda_df.values` : les noms des lignes (individus) : pour récupérer le dataframe sous forme d'array `numpy 2d`,
   — `famille_panda_df.describe` : renvoie un dataframe donnant des statistiques sur les valeurs (nombres de valeurs, moyenne, écart-type, ...), mais uniquement sur les colonnes numériques (faire `famille_panda_df.describe(include = 'all')` pour avoir toutes les colonnes).
5. Dimension d'un dataframe :
   — `famille_panda.shape` : renvoie la dimension du dataframe sous forme (nombre de lignes, nombre de colonnes),
   — `len(famille_panda)` : renvoie le nombre de lignes,
   — `len(famille_panda.colums)` : renvoie le nombre de colonnes.

3.2 Sélection de lignes/colonnes spécifiques à partir d'un DataFrame

Il est possible de parcourir les éléments d'un DataFrame, grâce à la méthode `iterrows` qui renvoie (à chaque itération d'une boucle for) un tuple dont le premier élément est l'index de la ligne, et le second le contenu de la ligne en question.

> **Question**
> Écrivez une boucle for permettant d'afficher les pandas un à un.

Pour sélectionner des lignes en fonction d'une expression conditionnelle, utilisez une condition entre les crochets de sélection `[]`. L'exemple ci-dessous montre comment sélectionner uniquement les pandas dont le ventre est de 80cm :

```
condition = famille_panda_df["ventre"] == 80
pandas_80 = famille_panda_df[condition]
```

> **Note**
> La sortie de l'expression conditionnelle (>, but also ==, ! =, <, ≤, . . .) est une série pandas de valeurs booléennes (True ou False) avec le même nombre de lignes que le DataFrame d'origine. Une telle série de valeurs booléennes peut être utilisée pour filtrer le DataFrame en le plaçant entre les crochets de sélection []. Seules les lignes pour lesquelles la valeur est True seront sélectionnées.

Similaire à l'expression conditionnelle, la fonction conditionnelle `isin()` renvoie un True pour chaque ligne dont les valeurs figurent dans la liste fournie. Pour filtrer les lignes en fonction d'une telle fonction, utilisez la fonction conditionnelle à l'intérieur des crochets de sélection [].

> **Question**
> Écrivez la condition permettant de sélectionner les pandas dont la taille des poils est comprise entre 5 et 6.

Lorsque vous utilisez les noms de colonne ou une expression de condition, utilisez l'opérateur `loc` devant les crochets de sélection `[]`. Pour la partie avant et après la virgule, vous pouvez utiliser une seule étiquette, une liste d'étiquettes, une expression conditionnelle ou un signe deux-points. L'utilisation d'un signe deux-points indique que vous souhaitez sélectionner toutes les lignes ou colonnes.

```
condition = famille_panda_df["ventre"] == 80
pandas_80 = famille_panda_df[condition]
```

Maintenant, ajoutons des lignes à notre dataframe. Il y a plusieurs méthodes pour cela, la plus consiste à assembler ensemble deux dataframes.

```
quelques_pandas = pd.DataFrame ([[105,4,19,80],[100,5,20,80]],
                                columns = famille_panda_df.columns
tous_les_pandas = famille_panda_df.append(quelques_pandas)
tous_les_pandas
```

> **Question**
> créer une nouvelle colonne "Sexe", composée de chaînes de caractères "f" pour femelle et "m" pour mâle. la maman et le bébé sont des femelles, le papa est un mâle.

3.3 Lecture de données avec `Pandas`

La librairie `Pandas` permet de lire des données à partir de fichiers stockés dans différents formats. Dans cette partie, nous allons voir comment lire un fichier CSV avec Pandas. Pour cela, il suffit de créer un dataframe à partir d'un CSV (dans notre cas, on utilisera les données de `iris.csv`[4]) :

```
iris = pd.read_csv("iris.csv",header= None)
iris_df =pd.DataFrame(iris, columns = iris.columns)
```

> **Note**
> La méthode `read_csv` offre différents arguments permettant de spécifier en autres le type de délimiteur entre colonnes (`,` , `;` ou tabulation), le délimiteur des décimales pour les nombres (virgule ou le point), l'existence ou pas d'un en-tête, etc.

4 Graphiques avec `Seaborn`

`Seaborn` est une bibliothèque de visualisation de données Python basée sur `Matplotlib`. Il fournit une interface de haut niveau pour générer des graphiques statistiques.

4. Le fichier est téléchargeable à partir de Moodle.

Dans cette section, vous allez utiliser `Seaborn` pour générer certains graphiques permettant la visualisation de données. Pour cela, vous utiliserez le données du fichier `iris.csv`.

1. `Pairplot` : pour visualiser les relations par paires d'un dataframe pandas. Seules les colonnes avec des valeurs numériques sont prises en compte.
   — `sns.pairplot(data = iris_df)`
   — s'il y a une colonne avec des catégories, on peut colorer les points selon la catégorie : `sns.pairplot(data = iris_df, hue='species', palette= {'setosa': 'green', 'virginica': 'red', 'versicolor': 'blue'})`
2. `Stripplot` : permet de représenter un nuage de points 1d pour chaque valeur de catégorie :
   — `sns.stripplot(x = 'species', y='petal_length', data = iris_df, jitter= 0.2)`
3. `countplot` : permet de représenter sous forme de barres le nombre d'instances d'un attribut donnée : `sns.countplot(x = 'species', data = iris_df)`
4. `scatterplot` : permet de représenter un nuage de points pour chaque ligne du dataframe (plusieurs options possibles) : `sns.scatterplot(data = iris_df, y = 'sepal_width', x = 'petal_length')`

5 Références

1. Découvrez les librairies Python pour la Data Science [5].
2. Documentation `Pandas` [6]

5. `https://mooc-francophone.com`
6. `https://pandas.pydata.org/pandas-docs/stable/reference/index.html#api`