Chargement et inspection des données dans Pandas

Avant d'analyser quoi que ce soit, il faut **charger des données avec Pandas** dans un `DataFrame`, puis les inspecter. Pandas propose un large éventail de fonctions pour lire des données de sources et de formats très variés.

Lire des données depuis différentes sources (CSV, Excel, SQL, etc.)



```python
import pandas as pd

# Charger un fichier CSV
df_csv = pd.read_csv('data.csv')

# Charger un fichier Excel (nécessite openpyxl)
df_excel = pd.read_excel('data.xlsx')

# Choisir une feuille précise avec le paramètre sheet_name
df_sheet1 = pd.read_excel('data.xlsx', sheet_name='Sheet1')

# Charger les données d'une table SQL via un moteur SQLAlchemy
from sqlalchemy import create_engine
engine = create_engine('sqlite:///mydatabase.db')
query = 'SELECT * FROM mytable'
df_sql = pd.read_sql_query(query, engine)

# Charger les tableaux d'une page web :
# read_html renvoie une LISTE de DataFrames
url = 'https://example.com/data-table.html'
tables = pd.read_html(url)
df_html_table = tables[0]

# Charger un fichier JSON
df_json = pd.read_json('data.json')

# Charger un fichier Parquet (nécessite pyarrow)
df_parquet = pd.read_parquet('data.parquet')
```

Python

Paramètres importants de `read_csv()` à connaître :

- **`filepath_or_buffer`** (le premier argument) : le chemin du fichier CSV. Tu peux fournir un chemin (chaîne de caractères), une URL ou un objet de type fichier ;
- **`names`** : une liste de noms pour remplacer les noms de colonnes ;
- **`sep`** (pour *separator*) : le caractère qui sépare les champs. La valeur par défaut est la virgule (`,`), mais tu peux utiliser une tabulation (`'\t'`), un point-virgule (`';'`) ou tout autre délimiteur ;
- **`index_col`** : la ou les colonnes à utiliser comme index. Tu peux donner un nom de colonne ou une position (à partir de 0) ;

- `skiprows` : le nombre de lignes à sauter au début du fichier, utile quand il commence par des métadonnées ou des commentaires.

Afficher un DataFrame

Charger les données, c'est bien. Les voir, c'est mieux : afficher un `DataFrame` est la première étape pour comprendre son contenu. Dans un notebook, il suffit de taper le nom du `DataFrame` dans une cellule pour voir les premières et les dernières lignes. Pandas propose aussi plusieurs méthodes pour afficher une partie précise :

- `.head(n)` affiche les `n` premières lignes (5 par défaut). C'est un aperçu rapide de la structure des données ;
- `.tail(n)` affiche les `n` dernières lignes, pratique pour vérifier la fin du jeu de données ;
- `.sample(n)` affiche `n` lignes tirées au hasard, pour explorer différentes parties du jeu de données ;
- `.columns` donne les noms des colonnes.

Dans la suite du tuto, on utilise ce petit `DataFrame` d'exemple :

```python
import pandas as pd

df = pd.DataFrame({
    "Name": ["Alice", "Bob", "Charlie", "David", "Emma", "Frank"],
    "Age": [25, 30, 35, 40, 28, 30],
    "City": ["Paris", "Lyon", "Paris", "Nice", "Lyon", "Paris"],
    "Salary": [3200, 4100, 5200, 6100, 3800, 4500],
    "Gender": ["F", "M", "M", "M", "F", "M"],
})

print(df.head(3))
print(df.tail(2))
print(df.columns)
```

Python

```
      Name  Age   City  Salary Gender
0    Alice   25  Paris    3200      F
1      Bob   30   Lyon    4100      M
2  Charlie   35  Paris    5200      M
    Name  Age   City  Salary Gender
4   Emma   28   Lyon    3800      F
5  Frank   30  Paris    4500      M
Index(['Name', 'Age', 'City', 'Salary', 'Gender'], dtype='str')
```

Explorer les données : shape, info, describe

Ce sont les premières choses à vérifier quand tu découvres un jeu de données :

- `.shape` renvoie un tuple : le nombre de lignes, puis le nombre de colonnes ;
- `.info()` résume le `DataFrame` : types de données, nombre de valeurs non nulles et utilisation de la mémoire. Pour voir uniquement les types, utilise `.dtypes` ;
- `.describe()` calcule des statistiques de base pour chaque colonne numérique : nombre de valeurs, moyenne, écart type, minimum, maximum et quartiles.

```python
print(df.shape)
df.info()
print(df.describe())
```

Python

```
(6, 5)
<class 'pandas.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Name    6 non-null      str
 1   Age     6 non-null      int64
 2   City    6 non-null      str
 3   Salary  6 non-null      int64
 4   Gender  6 non-null      str
dtypes: int64(2), str(3)
memory usage: 372.0 bytes
             Age       Salary
count   6.000000     6.000000
mean   31.333333  4483.333333
std     5.354126  1038.107252
min    25.000000  3200.000000
25%    28.500000  3875.000000
50%    30.000000  4300.000000
75%    33.750000  5025.000000
max    40.000000  6100.000000
```

Valeurs uniques, comptages et statistiques de base

Pour les colonnes catégorielles ou discrètes, tu peux explorer les valeurs uniques et leur fréquence :

- `.nunique()` compte le nombre de valeurs uniques de chaque colonne. C'est pratique pour comprendre la diversité des colonnes catégorielles ;
- `.value_counts()` compte les occurrences de chaque valeur d'une colonne ;
- **Accéder à une colonne :** `df.City` ou `df['City']` donnent le même résultat. La notation avec crochets fonctionne toujours, alors que la notation avec un point ne marche que si le nom de la colonne ne contient ni espace ni caractère spécial, et ne porte pas le nom d'une méthode du `DataFrame`.

```python
print(df.nunique())

# Les deux écritures donnent le même résultat
print(df.City.equals(df['City']))
```

Python

```
Name      6
Age       5
City      3
Salary    6
Gender    2
dtype: int64
True
```

Par défaut, `value_counts()` ne compte pas les valeurs manquantes. Ajoute `dropna=False` pour les inclure :

```python
print(df['City'].value_counts())

# dropna=False compte aussi les valeurs manquantes (NaN)
print(df['City'].value_counts(dropna=False))
```

Python

```
City
Paris    3
Lyon     2
Nice     1
Name: count, dtype: int64
City
Paris    3
Lyon     2
Nice     1
Name: count, dtype: int64
```

Tu peux enfin calculer des statistiques sur des colonnes précises : somme, minimum, maximum, moyenne, médiane ou mode.

```

```

```python
print(df.Age.mean())
print(df.Salary.median())
print(df.Gender.mode())
```

Python

```
31.333333333333332
4300.0
0    M
Name: Gender, dtype: str
```

Tu sais maintenant charger un jeu de données et en faire une première inspection. La suite de la série porte sur la sélection et l'indexation des données avec Pandas. Pour les détails de chaque fonction de lecture, consulte aussi la documentation officielle de Pandas sur les entrées/sorties.

Pour aller plus loin, consulte aussi notre guide des 10 astuces Pandas en Python.

Du fichier CSV brut à la première analyse. C'est le chemin que suit le cours Analyse et visualisation de données : tu charges, tu inspectes, tu résumes, puis tu traces.

