Pandas

Analyse de données en Python

Introduction

Dans le chapitre sur les fichiers CSV, on a chargé une table sous forme de **liste de dictionnaires**, puis on l'a parcourue avec des boucles pour filtrer, trier ou calculer une moyenne. Chaque traitement demandait d'écrire sa propre boucle.

Pandas est une bibliothèque Python consacrée à l'analyse de données. Elle charge une table dans un objet qui sait déjà faire ces traitements : une seule ligne suffit pour calculer la moyenne d'une colonne, garder les lignes qui vérifient une condition ou trier la table. On l'importe sous le nom court `pd` :

```
import pandas as pd
```

La table d'exemple

Le fichier `notes.csv` contient les notes de six élèves de deux classes. Chloé n'a pas de note de physique et Farid n'a pas de note de NSI.

```
prénom,classe,maths,nsi,physique
Alice,1A,14,17,12
Bruno,1B,9,13,11
Chloé,1A,16,15,
David,1B,11,18,14
Emma,1A,8,12,10
Farid,1B,15,,13
```

Lire un fichier CSV

La fonction `read_csv()` renvoie un **DataFrame**, c'est-à-dire un tableau à deux dimensions : chaque colonne porte le nom donné par l'en-tête, et chaque ligne reçoit un numéro, appelé **index**, qui commence à 0.

```
df = pd.read_csv("notes.csv")
print(df)
```

```
  prénom classe  maths   nsi  physique
0  Alice     1A     14  17.0      12.0
1  Bruno     1B      9  13.0      11.0
2  Chloé     1A     16  15.0       NaN
3  David     1B     11  18.0      14.0
4   Emma     1A      8  12.0      10.0
5  Farid     1B     15   NaN      13.0
```

Pandas reconnaît seul les nombres. Une valeur manquante s'affiche `NaN` (*Not a Number*, « pas un nombre ») ; une colonne qui en contient est stockée en flottants, d'où l'écriture `17.0`. Avec le point-virgule comme délimiteur, on écrit `pd.read_csv("notes.csv", sep=";")`.

Un premier aperçu

`df.shape` donne le tuple (nombre de lignes, nombre de colonnes), ici `(6, 5)`, et `list(df.columns)` la liste des noms de colonnes. `df.head(n)` renvoie les `n` premières lignes et `df.tail(n)` les `n` dernières.

Les colonnes

On obtient une colonne en écrivant son nom entre crochets. Le résultat est un objet **Series** : une suite de valeurs, chacune accompagnée de son index. Une série connaît les calculs usuels : `mean()` (moyenne), `max()`, `min()`, `sum()`, `count()` (nombre de valeurs présentes).

```
print(f"Moyenne : {round(df['maths'].mean(), 1)}")
print(f"Note la plus haute : {df['maths'].max()}")
```

```
Moyenne : 12.2
Note la plus haute : 16
```

Pour garder plusieurs colonnes, on donne la liste de leurs noms : `df[["prénom", "nsi"]]`.

Les lignes

`df.loc[i]` renvoie la ligne d'index `i`, et `df.loc[i, "nsi"]` une seule case : `df.loc[2, "nsi"]` vaut `15.0`.

Sélectionner selon une condition

```
print(df[df["maths"] >= 12])
```

```
  prénom classe  maths   nsi  physique
0  Alice     1A     14  17.0      12.0
2  Chloé     1A     16  15.0       NaN
5  Farid     1B     15   NaN      13.0
```

L'expression `df["maths"] >= 12` compare chaque note à 12 et produit une série de booléens ; seules les lignes associées à `True` sont gardées. Pour combiner deux conditions, on remplace `and` par `&` et `or` par `|`, et on entoure chaque condition de parenthèses :

```
print(df[(df["classe"] == "1A") & (df["nsi"] >= 15)])
```

Trier

`df.sort_values(by="nsi", ascending=False)` trie les lignes par note de NSI décroissante ; les valeurs manquantes sont placées à la fin. Cette méthode ne modifie pas `df` : elle renvoie un **nouveau** DataFrame, qu'il faut affecter à une variable pour le conserver.

Les valeurs manquantes

`df.isna().sum()` compte les cases vides de chaque colonne. `df.fillna(v)` remplace chaque valeur manquante par `v`, et `df.dropna()` supprime les lignes incomplètes. Le choix dépend de la situation : une note absente n'est pas une note nulle, et `df.fillna(0)` serait ici injuste.

Ajouter une colonne

On crée une colonne par une affectation ; le calcul s'applique à toutes les lignes à la fois. `mean(axis=1)` calcule la moyenne de chaque ligne en ignorant les valeurs manquantes, et `round(1)` arrondit au dixième :

```
df["moyenne"] = df[["maths", "nsi", "physique"]].mean(axis=1).round(1)
```

La moyenne de Chloé vaut alors 15,5 : elle est calculée sur ses deux notes seulement.

Regrouper

`groupby()` rassemble les lignes qui ont la même valeur dans une colonne, puis applique un calcul à chaque groupe :

```
print(df.groupby("classe")[["maths", "nsi", "physique"]].mean().round(1))
```

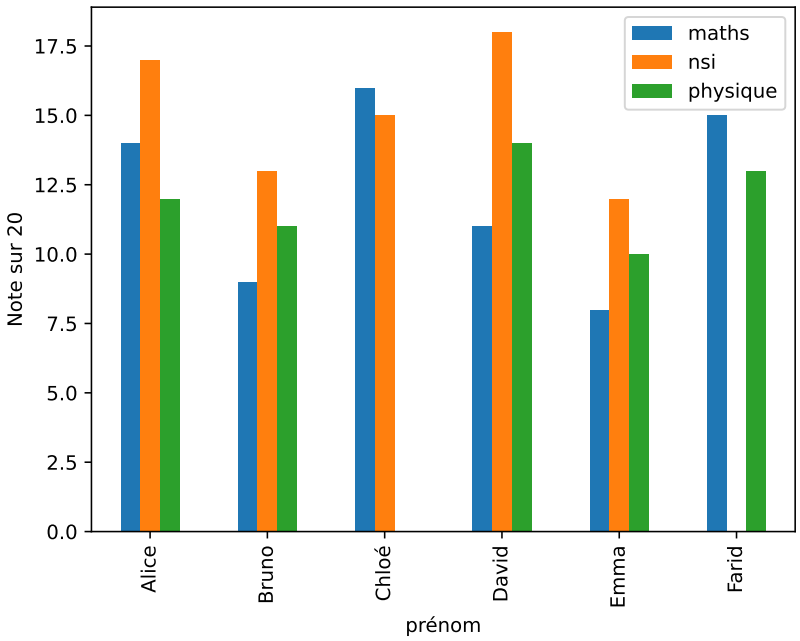
```
         maths   nsi  physique
classe
1A        12.7  14.7      11.0
1B        11.7  15.5      12.7
```

Résumé statistique et graphique

`df.describe()` donne pour chaque colonne numérique l'effectif des valeurs présentes, la moyenne, l'écart type, le minimum, les quartiles et le maximum. La méthode `plot()` trace un graphique grâce à `matplotlib` ; le paramètre `kind` choisit le type : `"bar"`, `"line"` ou `"scatter"`.

```
import matplotlib.pyplot as plt

df.plot(kind="bar", x="prénom", y=["maths", "nsi", "physique"])
plt.ylabel("Note sur 20")
plt.show()
```



Chaque élève reçoit un groupe de trois barres, une par matière. Une note manquante ne donne pas de barre.