Cours — DataFrames avec Pandas

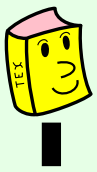
Initiation à l'algorithmique et programmation

Revekka Kyriakoglou

Plan

- Installer / importer Pandas, comprendre *Series* vs *DataFrame*
- Créer / lire un DataFrame, explorer (`head/info/describe`)
- Nettoyer + transformer (valeurs manquantes, types, colonnes)
- Grouper et résumer (`groupby`)
- Mini-TP (lecture CSV $\rightarrow$ nettoyage $\rightarrow$ analyse $\rightarrow$ export)

Installer Pandas



Installer

- Dans un terminal :

```
pip install pandas
```

Si vous avez Anaconda/Miniconda : `conda install pandas`.
Dans ce cours, on suppose Python 3.

Importer Pandas et créer un DataFrame

```python
import pandas as pd

data = {
    "voitures": ["BMW", "Volvo", "Ford"],
    "passages": [3, 7, 2]
}

df = pd.DataFrame(data)
print(df)
```

Un **DataFrame** = un tableau (lignes/colonnes) avec des noms de colonnes.

Exercice 1

Créez un DataFrame `df` à partir du dictionnaire :

- colonne `Nom` :
  `["Alice","Bob","Charles","Diane"]`
- colonne `Age` : `[25,30,35,40]`
- colonne `Ville` :
  `["Paris","Lyon","Marseille","Nantes"]`

Affichez `df`.

Voir rapidement les données

```
print(df.head())       # 5 premières lignes
print(df.tail())       # 5 dernières lignes
print(df.sample(2))    # 2 lignes au hasard
```

`head(10)` affiche 10 lignes. Très utile pour vérifier que l'import est correct.

Structure : taille, types, résumé

```
print(df.shape)    # (nb_lignes, nb_colonnes)
print(df.dtypes)   # types des colonnes
print(df.info())   # résumé (NaN, mémoire, types)
```

```
print(df.describe())   # stats sur colonnes numériques
```

`describe()` = moyenne, écart-type, min/max, quartiles.

Exercice 2

Avec le DataFrame de l'exercice 1 :

1. Affichez `df.shape`
2. Affichez `df.dtypes`
3. Affichez `df.describe()`

Question : **Pourquoi** `describe()` ne montre pas `Nom` et `Ville` ?

Accéder aux colonnes

```python
# Une colonne (Series)
ages = df["Age"]
print(ages)

# Plusieurs colonnes (DataFrame)
sub = df[["Nom", "Ville"]]
print(sub)
```

Filtrer des lignes (masques booléens)

```
# Garder les personnes de plus de 30 ans
df30 = df[df["Age"] > 30]
print(df30)

# Garder seulement "Paris"
df_paris = df[df["Ville"] == "Paris"]
print(df_paris)
```

La condition `df["Age"] > 30` produit une liste de True/False alignée sur les lignes.

loc vs iloc

```
# iloc : positions (0,1,2,...)
print(df.iloc[0])          # 1ère ligne
print(df.iloc[0:2])        # lignes 0 et 1

# loc : labels (index)
df2 = df.copy()
df2.index = ["a", "b", "c", "d"]
print(df2.loc["b"])        # ligne d'index "b"
```

Exercice 3

Avec `df` :

1. Affichez la colonne `Ville`.
2. Affichez les lignes où `Ville == "Lyon"`.
3. Renommez l'index en `["a","b","c","d"]` puis affichez la ligne `"c"` avec `loc`.

Valeurs manquantes : détecter

```
import numpy as np

data = {
   "Nom": ["Alice", "Bob", None, "Diane"],
   "Age": [28, None, 35, 29],
   "Ville": ["Paris", "Lyon", "Marseille", None]
}
df = pd.DataFrame(data)

print(df.isnull().sum())    # nombre de NaN/None par
↪  colonne
```

Valeurs manquantes : supprimer ou remplacer

```python
# 1) Supprimer les lignes avec au moins un NaN
df_drop = df.dropna()

# 2) Remplacer par une valeur fixe
df_fill = df.fillna("Inconnu")

# 3) Remplacer Age manquant par la moyenne (numérique)
m = df["Age"].mean()
df["Age"] = df["Age"].fillna(m)
```

`fillna` ne modifie pas toujours en place : ici on réaffecte la colonne.

Types : convertir proprement

```
# Convertir une colonne en numérique (si besoin)
df["Age"] = df["Age"].astype(float)

# Convertir une colonne en catégorie (utile pour
↪  groupes)
df["Ville"] = df["Ville"].astype("category")
```

Exercice 4

Créez un fichier `data.csv` :

```
Name,Age,City,Level Alice,28,Paris,Advanced
Bob„Lyon,Intermediate Charlie,32,Paris,Beginner
Diane,25,Marseille,
```

1. Lisez-le avec `pd.read_csv`.
2. Affichez `isnull().sum()`.
3. Remplacez Age manquant par la moyenne.
4. Remplacez City manquant par `"Unknown"`.
5. Remplacez Level manquant par `"Unknown"`.

Renommer et supprimer

```
# Renommer une colonne
df = df.rename(columns={"Age": "age"})

# Supprimer une colonne
df = df.drop(columns=["Ville"])

# Supprimer une ligne par index (ex: index 0)
df = df.drop(index=[0])
```

Créer une colonne : opérations vectorisées

```
# Si df["age"] est numérique :
df["age_x2"] = df["age"] * 2
```

Préférer les opérations vectorisées (plus simple + plus rapide) quand c'est possible.

Créer une colonne à partir d'un niveau (sans lambda)

Idée

On utilise un dictionnaire de correspondance (mapping).

```
mapping = {"Beginner": 2, "Intermediate": 4, "Advanced":
↪  6}

# map() transforme chaque valeur avec le dictionnaire
df["Years_of_Study"] = df["Level"].map(mapping)

# Les valeurs non trouvées donnent NaN -> on peut
↪  remplir
df["Years_of_Study"] = df["Years_of_Study"].fillna(0)
```

Alternative (sans lambda) : `apply` avec une fonction nommée

```
def years(level):
    if level == "Beginner":
        return 2
    elif level == "Intermediate":
        return 4
    elif level == "Advanced":
        return 6
    else:
        return 0

df["Years_of_Study"] = df["Level"].apply(years)
```

Exercice 5

À partir du `data.csv` de l'exercice 4 :

1. Ajoutez `Years_of_Study` avec `map` (Beginner=2, Intermediate=4, Advanced=6, sinon 0).
2. Filtrez pour ne garder que les personnes avec `Age > 25`.
3. Exportez le résultat en `processed_data.csv` sans l'index.

Regrouper et calculer : `groupby`

```python
# Moyenne d'âge par ville
moy = df.groupby("City")["Age"].mean()
print(moy)

# Compter le nombre de lignes par ville
cnt = df.groupby("City")["Name"].count()
print(cnt)
```

`groupby` = "on sépare en groupes" puis "on résume" (mean, count, min, max...).

Plusieurs statistiques

```
stats = df.groupby("City")["Age"].agg(["count", "mean",
↪  "min", "max"])
print(stats)
```

Exercice 6

Sur votre DataFrame nettoyé :

1. Calculez la moyenne de `Age` par `City`.
2. Calculez le nombre de personnes par `Level`.
3. Affichez un tableau avec `count` et `mean` de `Age` par `City`.

`concat` : empiler des tableaux

```
df1 = pd.DataFrame({"A": ["A0","A1"], "B": ["B0","B1"]})
df2 = pd.DataFrame({"A": ["A2","A3"], "B": ["B2","B3"]})

vertical = pd.concat([df1, df2], axis=0)
horizontal = pd.concat([df1, df2], axis=1)

print(vertical)
print(horizontal)
```

`merge` : joindre sur une colonne commune

```
cours = pd.DataFrame({
    "City": ["Paris", "Lyon", "Marseille"],
    "Campus": ["A", "B", "C"]
})

# Jointure sur la colonne "City"
df2 = df.merge(cours, on="City", how="left")
print(df2)
```

`how="left"` : on garde toutes les lignes du DataFrame de gauche.

Mini-TP (objectif)

À partir de `data.csv` (exercice 4), produire `final.csv` :

1. Lire le CSV dans un DataFrame.
2. Nettoyer les valeurs manquantes (Age $\rightarrow$ moyenne, City/Level $\rightarrow$ Unknown).
3. Ajouter `Years_of_Study` (mapping).
4. Calculer la moyenne d'âge par ville et l'afficher.
5. Filtrer `Age > 25`.
6. Exporter en `final.csv` (sans index).

Checklist de fin

À vérifier

- `df.head()` ressemble-t-il au fichier source ?
- `df.isnull().sum()` : y a-t-il encore des NaN ?
- Les types sont-ils cohérents ? (`df.dtypes`)
- Le fichier `final.csv` est-il créé et lisible ?