EEIA 2024

Initiation à la programmation informatique avec Python

Jour 4 : Traitement de données : les dataframes

26 juillet 2024

Plan

Module Pandas

1. Lecture / Ecriture d'un fichier `csv`
2. La gestion des dates
3. Jouer avec les data frames
4. Agrégation
5. Tracés graphiques
6. Cheat sheets
7. Pour aller plus loin

Le Module `Pandas`

- **`Pandas`** permet de lire et traiter des blocs de données de manière pratique et facile à interpreter.
  Type de données utilisé: `pd.DataFrame`
- Un `DataFrame` se représente comme un tableau. Mais en pratique, il s'agit d'un **dictionnaire** dont les clés sont les noms des colonnes formant le tableau.
  - chaque colonne représente un type de données différent.
  - il est possible de nommer les ligne du tableau.

Le Module `Pandas`

Exemple:

```
import pandas as pd
df = pd.DataFrame({'num_legs': [2, 4], 'num_wings': [2, 0]}, index=['falcon', 'dog'])
df.head()
```

	num_legs	num_wings
falcon	2	2
dog	4	0

Lecture / Écriture d'un fichier `csv`

- En data science, on enregistre souvent les données dans des fichiers `csv`
- Un fichier `csv` est un fichier de **base des données recueillies - sans formatage particulier**. Chaque champ est séparé par une virgule ou un point-virgule (ou un autre séparateur défini par le data scientist).
- Fonction utilisée pour lire un fichier `csv` :

  `pd.read_csv``(file_path: str, sep: str, **kwargs)`
- Fonction utilisée pour écrire dans un fichier `csv` :

  `pd.to_csv``(file_path: str, sep: str, **kwargs)`

Lecture / Écriture d'un fichier `csv`

Exemple: Lecture d'un fichier `csv`

```
adresses = pd.read_csv('adresses.csv', sep=';')
adresses.head()
```

	COL_A	COL_B	COL_ADRESSE
0	JJJJ	21200	ZI des LAUVES 83340 LE LUC FRANCE
1	2095	21253	Carrefour de Fogata 20256 CORBARA FRANCE
2	530	21238	1 Route DE TAVEL C.C. GRAND ANGLE 30133 LES AN...
3	1813	7374	Av de la Libération ARLES FRANCE
4	120	21273	Lotissement Jardins d'Oletta 20232 OLETTA FRANCE

Lecture / Écriture d'un fichier `csv`

Exemple: Écriture dans un fichier `csv`

```
adresses.tail().to_csv('adresses_tail_5.csv', sep=',', index=False)
```

Le fichier `adresses_tail_5.csv` contient alors:

data > sheets > adresses_tail_5.csv

```
1   COL_A,COL_B,COL_ADRESSE
2   XXXX,25501,501 Avenue Gerard Rouviere  Zone Nicolas Appert  11400 CASTELNAUDARY FRANCE
3   BAAA,25600,Rue de Gournay 60490 Ressons Sur Matz FRANCE
4   PPPP,23000,26 Quai Marcel Boyer 94851 Ivry sur Seine cedex FRANCE
5   SA01,8401,62 Bld Louis ARMAND 53940 SAINT BERTHEVIN LES LAVAL  FRANCE
6   ZZZZ,25500,5 Rue Marcelin Berthelot ZAC du Vaulorin 91320 WISSOUS FRANCE
7
```

La gestion des dates

- Il est très fréquent d'avoir des dates dans nos jeux de données.
- Les dates sont représentés par des `str` dans le fichier `csv`

```
date,product,price
1/1/2019,A,10
1/2/2020,B,20
1/3/1998,C,30
```

- Bien manipuler les dates en python nécessite leur conversion en objet `date` ou plus précisément en `datetime`.

La gestion des dates

Exemple: lecture basique sans traitement du `csv`

```
# noter qu'on ne spécifie par l'argument `sep`
# sa valeur par defaut est `sep=','`
date_df = pd.read_csv('dates_1.csv')
date_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   date     3 non-null      object
 1   product  3 non-null      object
 2   price    3 non-null      int64 
dtypes: int64(1), object(2)
memory usage: 72.0+ bytes
```

La gestion des dates

Exemple: lecture basique sans traitement du `csv`

- On remarque que le type de la colonne `date` est `object`. (ce qu'on ne veut pas)

La gestion des dates

Exemple: lecture du `csv` en parsant les dates

```
date_df = pd.read_csv('dates_1.csv', parse_dates=['date'])
date_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype         
---  ------   --------------  -----         
 0   date     3 non-null      datetime64[ns]
 1   product  3 non-null      object        
 2   price    3 non-null      int64         
dtypes: datetime64[ns](1), int64(1), object(1)
memory usage: 200.0+ bytes
```

La gestion des dates

Exemple: lecture du `csv` en parsant les dates

- Cette fois, le type de la colonne `date` est bien `datetime`. (ce qu'on veut)

```
date_df
```

	date	product	price
0	2019-01-01	A	10
1	2020-01-02	B	20
2	1998-01-03	C	30

Jouer avec les data frames

- Accès aux colonnes de la dataframe

```
df.columns
```

- Affichage des 5 premières lignes

```
df.head(5)
# ou encore df.head(). La valeur par défaut est 5.
```

- Affichage des 3 dernières lignes

```
df.tail(3)
```

- Affichage des 4 premières lignes

```
df[:4]
```

Jouer avec les data frames

- Accès à une colonne dont on connaît le nom.
  En reprenant l'exemple des adresses on accède à la colonne `COL_ADRESSE` en faisant:

```
addresses["COL_ADRESSE"]
```

  ou encore

```
addresses.COL_ADRESSE
```

  Ceci est possible car `COL_ADRESSE` est un nom **valable de variable**. `addresses.COL ADRESSE` n'aurait pas marché.

Jouer avec les data frames

- Affichage des 6 premières lignes de la colonne `COL_ADRESSE`.

```
addresses.COL_ADRESSE[:6]
```

- Affichage des 6 premières lignes des colonnes `COL_A` et `COL_B`.

```
addresses[["COL_A", "COL_B"]][:6]
```

- Nombre de lignes et nombre de colonnes.

```
addresses.shape
```

Jouer avec les data frames

- Suppression des lignes sans données.
  Il peut arriver des fois que les données soit corrompues, des fois certaines valeurs manquent juste. Une manière de procéder est de supprimer de telles lignes de notre tableau:

```
df = df.dropna()
```

  ou encore

```
df.dropna(inplace=True)
```

Jouer avec les data frames

- Récupérer les addresses dont `COL_A` vaut 530.

```
adresses[adresses.COL_A == 530]
```

- Rajouter une colonne contenant la différence entre les valeurs de deux colonnes `x` et `y`.

```
df["diff"] = df.x - df.y
```

Agrégation

- Agréger c'est **unir en un tout**
- Une agrégation se suit généralement d'une opération sur chacun des groupes obtenus
  Il s'agit par exemple de `sum()`, `mean()`, `count()`, `max()`, `min()`, etc.
- Avec `pandas`, on utilise `pd.DataFrame.groupby` pour faire une agrégation

```
# grouper par départements
# puis faire la somme des valeurs sur chaque groupe
departements_df.groupby(["Departements"]).sum()
```

Tracés graphiques

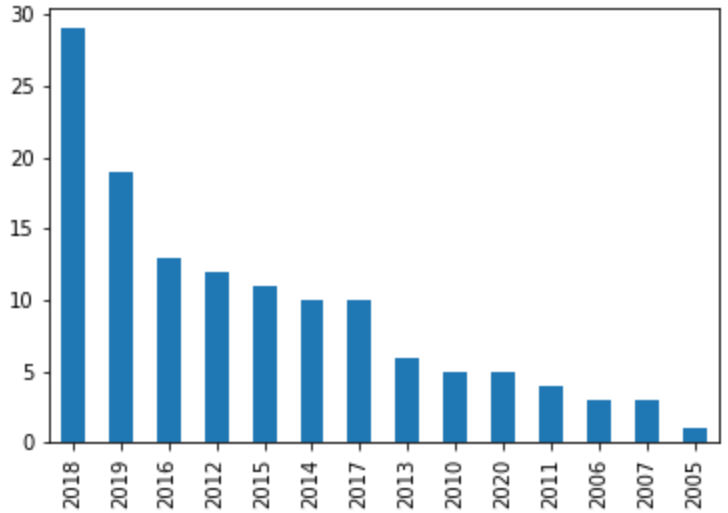
- On peut rapidement faire des tracés de graphes avec `pandas`
- Ces tracés son basés sur la librairie `matplotlib`
- On peut également utiliser directement `matplotlib` pour faire les tracés si on veut faire des choses avancées.

Tracés graphiques

Exemple:

- Données de l'unicef contenant une colonne `year`.
- Faire un barplot du nombre d'années

```
year_counts = data_unicef.year.value_counts()
year_counts.plot(kind="bar")
```



Cheat sheets - Importer des données

```
>>> pd.read_csv(filename) # d'un fichier .csv
>>>
>>> pd.read_excel(filename) # d'un fichier Excel
>>>
>>> pd.read_sql(query, connection_object) # d'une table/database SQL
>>>
>>> pd.read_json(json_str) # d'un str JSON
>>>
>>> # d'un dictionnaire, clés pour les noms des colonnes
>>> # valeurs dans des listes
>>> pd.DataFrame(dict)
```

Cheat sheets - Exporter des données

```
>>> df.to_csv(filename) # dans un fichier .csv
>>>
>>> df.to_excel(filename) # dans un fichier Excel
>>>
>>> df.to_sql(query, connection_object) # dans une table/database SQL
>>>
>>> df.to_json(filename) # dans un fichier JSON
```

Cheat sheets - Inspection des données

```
>>> df.head(n) # n premières colonnes
>>>
>>> df.tail(n) # n dernières colonnes
>>>
>>> df.shape # nombre de lignes et de colonnes
>>>
>>> df.info() # Index, Datatype et Information mémoire
>>>
>>> df.describe() # Résumé statisriques sur les colonnes numériques
>>>
>>> s.value_counts(dropna=False) # compter les donnée sur une série
```

Cheat sheets - Sélection

```
>>> df[col] # colonne ayant le nom col comme une série
>>>
>>> df[[col1, col2]] # les colonnes col1 et col2 en tant que DataFrame
>>>
>>> s.iloc[0] # Sélection par position
>>>
>>> s.loc['index_one'] # Sélection par index
>>>
>>> df.iloc[0,:] # Première colonne
>>>
>>> df.iloc[0,0] # Premier élément de la première colonne
```

Cheat sheets - Nettoyage de données

```
>>> df.columns = ['a','b','c'] # renommer les colonnes
>>>
>>> df.isnull() # vérifier les valeurs nulles ou manquantes
>>>
>>> df.notnull() # contraire de df.isnull
>>>
>>> df.dropna() # supprimer les lignes avec des données manquantes
>>>
>>> df.dropna(axis=1) # supprimer les colonnes avec des données manquantes
>>>
>>> # supprimer les colonnes avec moins de n valeurs non nulles
>>> df.dropna(axis=1,thresh=n)
>>>
>>> df.fillna(x) # remplacer les valeurs nulles par x
>>>
>>> s.fillna(s.mean()) # remplacer les valeurs nulles par la moyenne
```

Cheat sheets - Nettoyage de données

```
>>> s.astype(float) # conversion en float
>>>
>>> s.replace(1,'un') # remplacer les 1 par 'un'
>>>
>>> s.replace([1,3],['un','trois']) # remplacement multiple
>>>
>>> df.rename(columns=lambda x: x + 1) # renommage de colonnes en masse
>>>
>>> df.rename(columns={'old_name': 'new_ name'}) # renommage d'une colonne
>>>
>>> df.set_index('column_one') # changer d'index : column_one devient l'index
>>>
>>> df.rename(index=lambda x: x + 1) # renommage d'index en masse
```

Cheat sheets - Filtre, Trie et Groupby

```
>>> df[df[col] > 0.5] # lignes pour lesquelles la colonne col est > 0.5
>>>
>>> df[(df[col] > 0.5) & (df[col] < 0.7)] # lignes où 0.7 > col > 0.5
>>>
>>> df.sort_values(col1) # trier par la colonne col1 par ordre croissant
>>>
>>> # trier par la colonne col2 par ordre décroissant
>>> df.sort_values(col2,ascending=False)
>>>
>>> # trier par col1 croissant puis par col2 décroissant
>>> df.sort_values([col1,col2],ascending=[True,False])
```

Cheat sheets - Filtre, Trie et Groupby

```
>>> # retourne un objet de type groupby pour les valeurs de col
>>> df.groupby(col)
>>>
>>> df.groupby([col1,col2]) # groupby sur plusieurs colonnes
>>>
>>> # moyenne sur toutes les colonnes, groupé par valeur uniques sur col1
>>> df.groupby(col1).mean()
>>>
>>> # moyenne sur toutes les colonnes, groupé par valeur uniques sur col1
>>> df.groupby(col1).agg(np.mean)
>>>
>>> # appliquer la fonction np.mean() sur toutes les colonnes
>>> df.apply(np.mean)
```

Cheat sheets - Jointures

```
>>> # ajouter les lignes de df1 à la fin de df2 (colones identiques)
>>> df1.append(df2)
>>>
>>> # ajouter les colonnes de df1 à la fin df2 (lignes identiques)
>>> pd.concat([df1, df2],axis=1)
>>>
>>> # jointure style SQL
>>> df1.join(df2,on=col1,how='inner')
```

Cheat sheets - Statistiques

```
>>> df.describe()

>>> df.mean()

>>> df.corr()

>>> df.count()

>>> df.max()

>>> df.min()

>>> df.median()

>>> df.std()
```

Aller plus loin

- Vous pouvez faire bien plus avec Pandas.
- N’hésitez pas à faire des recherches si vous voulez utiliser une fonction qui n’a pas été abordée dans ce chapitre.
- La seule limitation, c'est votre curiosité.

FIN