Pandas

Hadrien Bodin

Mines

14 octobre 2025

Table des matières

Introduction

Nous avons vu quelques fonctions de numpy, pour manipuler les numpy.ndarray qui sont des tableaux

- multidimensionnels
- homogènes
- de taille fixe

Example

Pour quels types de données numpy ne semble pas adapté ?

Introduction

Il existe d'autres tables de données, très fréquentes en data-science où on a :

- une observation par ligne
- plusieurs informations par observation
- les différentes informations forment les colonnes de la table
- ces colonnes ne sont pas toutes du même type...

Comme un tableau excel, par exemple

Le CSV

Excel est un format propriétaire. Le standard pour stocker ce type de données est souvent le CSV (comma separated values)

- une observation par ligne
- les différentes colonnes sont séparées par des virgules (et non point-virgule pour le format standard)

Exemple

Ouvrir dans un éditeur de texte le fichier des données du titanic

Pandas

Pour lire, mettre en forme et manipuler des données de data-science on utilise la librairie pandas (2008).
Polars commence à concurencer sérieusement Pandas

- numpy ne propose pas directement ces fonctions
- pandas expose un type évolué de table de données : les pandas.DataFrame
- pandas possède un type pour gérer une ligne et une colonne le pandas.Series
- pandas comme numpy favorisent l'efficacité (parfois au détriment de la lisibilité)
- pandas repose entièrement sur numpy (aujourd'hui en tous cas)
- i.e. les données manipulées par pandas sont implémentées comme des tableaux numpy.ndarray
- il peut vous arriver d'utiliser la librairie numpy dans du code pandas

Pandas

```
import pandas as pd
pd.__version__
-> '2.3.1'
```

Ouverture d'un fichier

```
df = pd.read_csv('data/titanic.csv')
type(df)
-> pandas.core.frame.DataFrame
```

pandas.core.frame.DataFrame est le même type que pandas.DataFrame

```
df.head()
-> PassengerId   Survived        Pclass   Name      Sex       Age       SibSp    Parch
0         552          0             2   Sharp, Mr. Percival James R    male      27.0
1         638          0             2   Collyer, Mr. Harvey      male      31.0     1
```

Il est possible de lire des excel. C'est en réalité très rare.

Description

La méthode describe() vous donne un premier aperçu rapide de vos données
Sur une DataFrame, elle vous donne pour chaque colonne de type numérique

- le nombre de valeurs non-manquantes (voir colonne Age)
- la moyenne
- l'écart-type
- le minimum
- les 3 quartiles (les valeurs à 25
- le maximum

Description

on remarque que pandas.DataFrame.describe

- a, par défaut, appliqué les calculs sur les colonnes numériques même quand ça n'a pas forcément beaucoup d'intérêt - voir Survived ou Pclass qui sont plutôt des catégories
- n'a rien fait sur les colonnes non-numériques

Taille mémoire d'un tableau

On peut forcer la méthode à s'appliquer à toutes les colonnes.
Pour les colonnes non-numériques, seront affichés à la place

- le nombre de valeurs
- le nombre de valeurs uniques
- la valeur la plus fréquente top
- sa fréquence freq

Quelques points sur le stockage

- Les lignes et les colonnes ont des index
- Les opérations sur ces index sont le plus efficace possible

On verra par la suite qu'on peut associer un indiçage à une indexation.

Pourquoi cette efficacité ?

- rechercher dans une liste est très inefficace (en moyenne $n/2$ soit $O(n)$)
- accéder au i^{me} élément d'un tableau est très efficace (en temps constant car on ne parcourt pas tous les éléments, soit $O(1)$)

Pourquoi cette efficacité ?

On a donc intérêt à :

- trouver une caractéristique qui identifie une observation de manière unique (ex : PassengerId)
- calculer un index à partir de cette caractéristique qui soit le plus unique possible (pour avoir peu de collisions)
- utiliser cet index comme entrée dans une table
- la recherche peut alors être considérée comme en temps constant comme l'accès à un élément d'un tableau
- c'est la technique des tables de hachage (comme les dict ou set Python)

Pourquoi cette efficacité ?

pandas indexe ses lignes et ses colonnes suivant vos indications dit autrement, c'est à vous de choisir parmi les colonnes celle(s) qui peut servir d'identificateur pour servir d'index (un index en pandas n'est pas obligatoirement unique)

Index

Souvent, seules les colonnes ont un nom

- Les colonnes sont donc indexées par leur nom
- Les lignes sont indexées par leurs indices

Index

l'attribut columns permet d'accéder aux colonnes de la table

```
df.columns
-> Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

On remarque que son type est un objet de type Index, qui est itterable

```
df.columns[0]
-> 'PassengerId'
```

Accès aux colonnes

Une table pandas se comporte comme un dictionnaire

- où les clés sont les noms des colonnes
- où les clés sont les noms des colonnes

Exercice

Accéder à la colonne Age de la df et remarquer :

- le nombre d'entrées (de lignes)
- les indices
- le type des données de cette colonne
- que l'age d'au moins un passager est inconnu

Colonnes

Attention, pandas accepte que plusieurs colonnes portent le même nom. Faire attention à ne pas faire de doublons.

Colonnes

Autre accesseur aux colonnes :

```
df.Age is df['Age']
-> True
```

Colonnes

Les colonnes sont des objects pandas.core.series.Series.
on peut voir la ‘DataFrame‘ comme un dictionnaire qui associe ‘nom-de-colonne‘ $\rightarrow$ ‘un-objet-series‘

Les lignes

C'est une bonne pratique de choisir une colonne comme index (des lignes) de la table quand on le peut.

Par exemple, dans la table du titanic, on remarque que la colonne PassengerId contient un identifiant unique pour chaque passager.

Trois options existent pour faire passer une colonne comme index

```
df = pd.read_csv('data/titanic.csv', index_col='PassengerId
#
df = pd.read_csv('data/titanic.csv')
df = df.set_index('PassengerId')
#
df = pd.read_csv('data/titanic.csv')
df.set_index('PassengerId', inplace=True)
```

Exercice

Observer la nouvelle allure de la table

Indexation des séries

Les séries aussi possèdent un index. Cet index est le même pour toutes les colonnes, ce qui permet de les aligner.

```
df['Name'][552]
    -> 'Sharp, Mr. Percival James R'
```

Index vs Indice

Un indice part de zéro et est incrémenté.

Un index non.

Index vs Indice

Un index peut prendre pour valeur ce que l'on veut, par ex :

- les noms de colonnes
- les id des passagers
- des identifiants de lignes

En l'absence d'index remarquable, pandas prend l'indicage comme index. Il s'agit alors d'un RangeIndex.

Indexation des lignes

L'index est accessible par l'attribut .index

```
df.index
-> RangeIndex(start=0, stop=891, step=1)
df = pd.read_csv('data/titanic.csv').set_index('PassengerId
df.index
-> Int64Index([552, 638, 499, 261, 395, 811, 758, 703, 406,
                ...
                556, 236, 224, 598, 258, 463, 287, 326, 396,
               dtype='int64', name='PassengerId', length=891)
```

Ces deux objets héritent de pandas.Index

Forme d'une dataframe

A l'heure actuelle pandas est basé sur numpy.
On peut récupérer la forme de la table :

```
df.shape
-> (891, 11)
```

Numpy gère le stockage et les calculs, Pandas gère l'indexation et quelques fonctions haut niveau de manipulation des données.

Exercice

Exercice

- Ouvrir le fichier petit-titanic.csv
- Afficher le début avec df.head(nb_lignes)
- Résoudre les problèmes
- Afficher le nb de colonnes, de lignes, et les index
- Traduire l'index des colonnes
- Utiliser 'Identifiant' comme index des lignes

Conditions et masques

Dans les analyses de données, il est fréquent de sélectionner des données par des conditions (qui peuvent s'appliquer, selon le contexte, à tout un tableau, ou un morceau précis comme une colonne, une ligne, un sous-tableau...)

En pandas, comme en numpy, les fonctions sont vectorisées par souci de rapidité du code
$\rightarrow$ il ne faut jamais itérer avec un for-python sur les valeurs d'une table (les itérations se font dans le code des fonctions numpy et pandas)

Conditions et masques

Comme en numpy, une expression conditionnelle va s'appliquer à toute la structure et retourner une structure du même forme, mais avec des résultats booléens
Comme on avec les masques numpy

```
titanic['Age']
titanic['Age'] < 12
```

Combiner les conditions

En pandas comme en numpy pour combiner les conditions

- on utilise (et) | (ou) et (non)
- ou les numpy.logical_and, numpy.logical_or, numpy.logical_not
- et surtout pas and, or et not (opérateurs Python non vectorisés)
- on parenthèse toujours les expressions

```
girls = (df['Age'] < 12) & (df['Sex'] == 'female')
girls.sum()
-> 32
```

Indexation par un masque

En pratique, le plus souvent on est intéressés par les lignes qui correspondent au masque
Pour les “extraire” de la dataframe on va tout simplement indexer la dataframe par le masque

```
girls = (df['Age'] < 12) & (df['Sex'] == 'female')
df[girls]
```

```
->
                 Survived  Pclass
Name       Sex   Age  SibSp  Parch Ticket       Fare     Cabin
Embarked
PassengerId
238                     1       2  Collyer, Miss. Marjorie "Lot
female   8.0   0        2    C.A. 31921   26.250   NaN      S
375                     0       3        Palsson, Mi
```

Une première fonction

value_counts()

Exercice

Utiliser value_counts() pour trouver le nombre d'enfants et d'adultes.

Valeurs manquantes

Souvent, certaines colonnes ont des valeurs manquantes (car non renseingées dans le .csv)
On a souvent besoin de les trouver, les compter, et si nécessaire les éliminer
Quatre méthode existent

- isna()
- notna()
- isnull()
- notnull()

Exercice

Combien manque-t-il de valeurs dans la colonne des ages ?

Cette méthode s'applique à un tableau ou une dataframe

Compter les valeurs manquantes

```
df.isna().sum(axis=0) #valeur par d faut
#vs
df.isna().sum(axis=1)
```

Agrégation

Les méthodes numpy d'agrégation (comme sum() et mean() et min() etc...) s'appliquent sur des pandas.DataFrame et des pandas.Series
On précise l'axis (0 = lignes)

Différence avec numpy :
Si on appelle sans préciser d'axe :

- axis 0 par défaut pour pandas = les lignes
- numpy agrège sur tout le tableau

Exercice

Comment récupérer un résultat global avec pandas ?

Exercice

Exercice

- Relisez la dataframe
- Comptez le nombre de valeurs uniques pour les colonnes 'Survived', 'Pclass', 'Sex' et 'Embarked'. Aidez vous de la fonction pd.Series.unique
- Utilisez l'expression df[cols] pour sélectionner la sous-dataframe réduite à ces 4 colonnes
- et utilisez l'attribut dtypes des pandas.DataFrame pour afficher le type de ces 4 colonnes
- quel type serait plus approprié pour ces colonnes ?

Exercice

- Relisez la dataframe
- Comptez les valeurs manquantes : dans toute la table, par colonne et par ligne
- Calculez le nombre de classes du bateau
- Calculez le taux d'hommes et de femmes
- Calculez le taux de personnes entre 20 et 40 ans (bornes comprises)
- Calculez le taux de survie des passagers
- Calculez le taux de survie des hommes et des femmes par classes

Indexation

Manipuler des parties (vues) de nos données est une opération fréquente en traitement des données, d'où l'importance de savoir localiser dans nos tables pandas des sous-parties afin de leur appliquer une fonction

Il est d'usage d'accéder aux données par leur index et non par leur indice (contrairement à np) En effet :

- l'ordre dans lequel sont les données est généralement secondaire et on préfère faire référence aux données par leur identifiant
- s'il n'y a pas besoin d'index particulier, privilégier np

Copy

Pour dupliquer une dataframe ou une série (ligne ou colonne), on utilise la fonction copy() Une modification de la copie ne modifie pas l'original.

`cf notebook`

Création de nouvelle colonne

On peut ajouter une nouvelle colonne à une df.
On le fait souvent à partir d'une colonne existante. Par exemple :

```
df['Deceased'] = 1 - df['Survived']
df.head(2)
```

Accès aux éléments

En pandas, il faut oublier la logique de numpy pour accéder aux éléments.
Notamment car l'ordre des dimensions est inversé par rapport à numpy.

Accès aux éléments

Pour ce faire, il existe les fonctions loc et iloc.

- loc se base sur les index
- iloc se base sur les indices

```
df.loc[552, 'Name']
-> 'Sharp, Mr. Percival James R'

df.iloc[0, 2]
-> 'Sharp, Mr. Percival James R'

df.iloc[-1, 2]
-> 'Richards, Master. George Sibley'
```

Sélection multiple

Pandas offre les techniques usuelles pour la sélection multiple

- sélection multiple explicite
- slicing

Séléction multiple

Si on ne précise pas les colonnes, on les obtient toutes.
Reste à préciser les index ou indices dans une liste

```
# notebook
```

Séléction multiple

Attention à toujours questionner votre intérêt à requêter un élément spécifique du tableau.
Souvent un filtre (un masque) suffit à récupérer ce dont on a besoin.

Séléction par masque booléen

```
df[ df['Sex'] == 'female' ]
# ou encore
df.loc[ df['Sex'] == 'female' ]
```

En réalité df['Sex']=='female' renvoie une series de booléens. Cette indexation est ensuite lue par df[] ou df.loc[]

Combinaison

Exercice

En une ligne, sélectionner la sous-dataframe des passagers qui ne sont pas en première classe et dont l'age est supérieur ou égal à 70 ans

Combinaison

Les fonctions loc et iloc renvoient des références.
Il faut donc prendre en compte que modifier une vue modifiera la table originale.
A contrario, c'est la seule méthode acceptable pour modifier des valeurs à la volée dans la table).

Groupements

En pandas, une table de données (encore appelée dataframe) a uniquement 2 dimensions.
Mais elle peut indiquer, avec ces deux seules dimensions, des sous-divisions dans les données, qui correspondent à des classes (ex : homme/femmes, adultes/enfants, survécus ou non etc...)
On sent donc qu'il peut être pertinent de grouper les éléments du tableau sur ces valeurs.
Il s'agit du df.groupby() qui renvoie un objet pandas.core.groupby.generic.DataFrameGroupBy

Groupements

On peut ensuite effectuer des opérations sur les groupes de façon indépendante. Cela est particulièrement utilse pour les fonctions d'aggrégation.

```
df.groupby('Sex').Age.max()
-> Sex
female     63.0
male       80.0
```

Il est aussi possible d'accéder à un groupe avec la méthode get_group() qui renvoie un dataframe.

Groupements multi-critères

```
by_class_sex = df.groupby(['Pclass', 'Sex'])
by_class_sex.size()
->
Pclass  Sex
1       female     94
        male      122
2       female     76
        male      108
3       female    144
        male      347
```

Les groupes sont indexés par une double indexation. Les index sont les tuples du produit cartésien des deux ensembles d'index.
Comme avec un index classique, il est possible d'itérer sur ces tuples.

Découpage

Parfois les groupes naïfs sont trop restrictifs, mais on souhaite tout de même regrouper les individus. Pour ce faire, il peut être pertinent de découper la table en groupes arbitraires à partir d'une valeur. En groupe d'âge par exemple.

```
pd.cut(df['Age'], bins=[0, 12, 19, 55, 100])
->
PassengerId
552      (19.0, 55.0]
638      (19.0, 55.0]
261               NaN   <- age inconnu au d part
              ...

832       (0.0, 12.0]
Name: Age, Length: 891, dtype: category
Categories (4, interval[int64, right]): [(0, 12] < (12, 19]
```

Il est possible de donner des noms aux groupes avec l'argument name.

Découpage

La fonction cut renvoie une pd.Series qui est notamment utile pour créer une nouvelle colonne. Une fois cette colonne ajoutée, il peut être possible de grouper sur les valeurs avec la méthode group_by.

Exercice

En utilisant ces nouvelles fonctions, calculez le taux de survie par classe d'age et classe de cabine.

Pivot

Ce type d'opérations est en réalité fréquent, et on veut souvent afficher :

- une valeur (précisément, une aggrégation des valeurs) d'une colonne
- en fonction de deux autres colonnes (catégorielles)

Sex	female	male
Pclass		
1	0.968085	0.368852
2	0.921053	0.157407
3	0.500000	0.135447

Pivot

La méthode pivot_table() fait exactement ce genre de traitement.
Cette méthode s'appelle sur une dataframe.
Elle prend en arguments :

- values : la ou les colonnes que l'on souhaite étudier
- index : la colonne utilisée pour les lignes du résultat
- columns : celle utilisée pour les colonnes
- aggfunc = mean : la fonction d'agrégation utilisée

```
df.pivot_table(values='Survived',
    index='Pclass', columns='Sex')
```

Pivot

Cette méthode renvoie une nouvelle dataframe.
Dans le cas où on fournit plusieurs valeurs ou index ou colonnes, le dataframe en sortie contient des multi$_i$*ndexpourindexerladonne*.

Ceci étant dit, il vaut mieux privilégier des tableaux simples avec ce pivot. Les tableaux en résultat deviennent très vite moins lisibles.

Exercice

Passez des listes au lieu de valeurs simples aux argumetns de pivot. Regardez comment se comporte le résultat, ainsi que l'indexation de ce dernier.

Exercice

A partir du fichier wine.csv :

- Affichez les valeurs min, max, et moyenne, de la colonne ‘magnesium’
- Ddéfinissez deux catégories selon que le magnesium est en dessous ou au-dessus de la moyenne (qu’on appelle mag-low et mag-high) ; rangez le résultat dans une colonne mag-cat
- Affichez cette figure :

	color-intensity		flavanoids		magnesium	
mag-cat	mag-low	mag-high	mag-low	mag-high	mag-low	mag-high
cultivator						
1	4.975000	5.734186	2.748750	3.069302	94.500000	110.744186
2	3.062500	3.231111	2.122692	1.987222	87.500000	116.277778
3	7.446786	7.325500	0.679643	0.924000	91.571429	110.150000

Filtre sur les groupes

Une fois des groupes constitués par la méthode groupby, il est parfois souhaitable de filtrer les lignes du tableau sur ce groupes.

On utilise sur l'objet groupby la méthode filter, qui prend une fonction en argument.

```
titanic = pd.read_csv("data/titanic.csv")
gb = df.groupby(by=['Sex', 'Pclass'])
extract = gb.filter(lambda df: len(df) %2 == 0)
```

Ici on ne garde que les éléments appartenant à des groupes de cardinalité paire.
Attention, extract est ici un dataframe.

Transformation sur les groupes

Au lieu d'appliquer un filtre, on peut choisir d'appliquer une transformation

```
df = titanic.copy()
gb = df.groupby(by=['Sex', 'Pclass'])
df['Age'] = gb['Age'].transform(lambda df: df-df.mean())
```

Résumé

- pour faire des groupements multi-critères on utilise df.groupby() qui renvoie un objet de type GroupBy
- On utilise cet objet comme objet intermédiaire, qui va propager des traitements sur des sous groupes, que l'on peut ensuite agréger
- Dans le cas où on groupe sur plusieurs caractéristiques, l'indexation se fait par des MultiIndex (des tuples)
- Pivot_table permet de synthétiser rapidement les informations d'une table

Exercice

Exercice

Td fin notebook 2-07

Création de dataframe

Les derniers notebook ont permis de montrer comment manipuler des dataframes. Voyons maintenant quelques manières d'en créer.

A partir d'un dictionnaire

A partir d'un dictionnaire : Un ensemble clé/valeur correspond au nom de la colonne et aux valeurs. Le constructeur est celui par défaut. Le paramètre 'index' permet de spécifier l'index des lignes.

```
cols_dict = {'speed' : [0.1, 17.5, 40, 48, 52, 69, 88],
             'lifespan' : [2, 8, 70, 1.5, 25, 12, 28], }

line_ids =  ['snail', 'pig', 'elephant', 'rabbit',
             'giraffe', 'coyote', 'horse']
df = pd.DataFrame(cols_dict, index = line_ids)

df
->          speed   lifespan
snail       0.1     2.0
pig         17.5    8.0
elephant    40.0    70.0
rabbit      48.0    1.5
giraffe     52.0    25.0
coyote      69.0    12.0
horse       88.0    28.0
```

A partir d'un array

Le np array est passé au tableau. Il n'y a pas d'index dans un array donc les indices sont utilisés comme index pour la table.
On peut donner en argument l'index des lignes et les columns

Constructeurs df

Il y a énormément d'autres façons de construire des dataframe. Cherchez sur internet en fonction de votre besoin précis. Il y aura toujours un moyen !

Exercice

Exercice

fin notebook 2-08

Concaténation de tableaux

Il est parfois nécessaire de fusionner plusieurs tableaux de données. La méthode à retenir dépend alors de la question suivante :

Les tableaux partagent-ils les même lignes ou les mêmes colonnes ?

- Mêmes colonnes : pd.concat([df1, df2, ...])
- Mêmes lignes : pd.merge(left, right, on=colonnes communes)

La façon de merger dépend intimement des deux tableaux. On peut notamment choisir

- Quelles colonnes sont utilisées pour le merge
- Quel index doit subsister
- Que faire si une ligne n'existe pas dans un des tableaux
- etc...

Les timeseries

Pour stocker des valeurs temporelles, plusieurs objets pandas existent.

- pd.Timestamp (une date)
- pd.Timedelta (une durée)
- pd.Period (un intervalle)

Ces types héritent des types numpy :

- np.datetime64
- np.timedelta64

Exercice

Notebook 2-10