Programmation Python

Manipulation de données avec **Pandas**

SAÏDI MASSINISSA

Table des matières

Présentation

- Library permettant la manipulation de données
- Construit sur numpy
- Introduction des objets DataFrame :
  - Tableaux multidimensionnels.
  - Ayant des étiquettes par lignes et colonnes.
  - Gérant des données de différents types.
  - Avec possiblement des données manquantes.
- Possibilité de faire différentes opérations rapidement et
- efficacement sur les données du dataframe.

- La version 2.0 Avril 2023 qui a augmenté la vitesse d’éxécution

Les objets Pandas

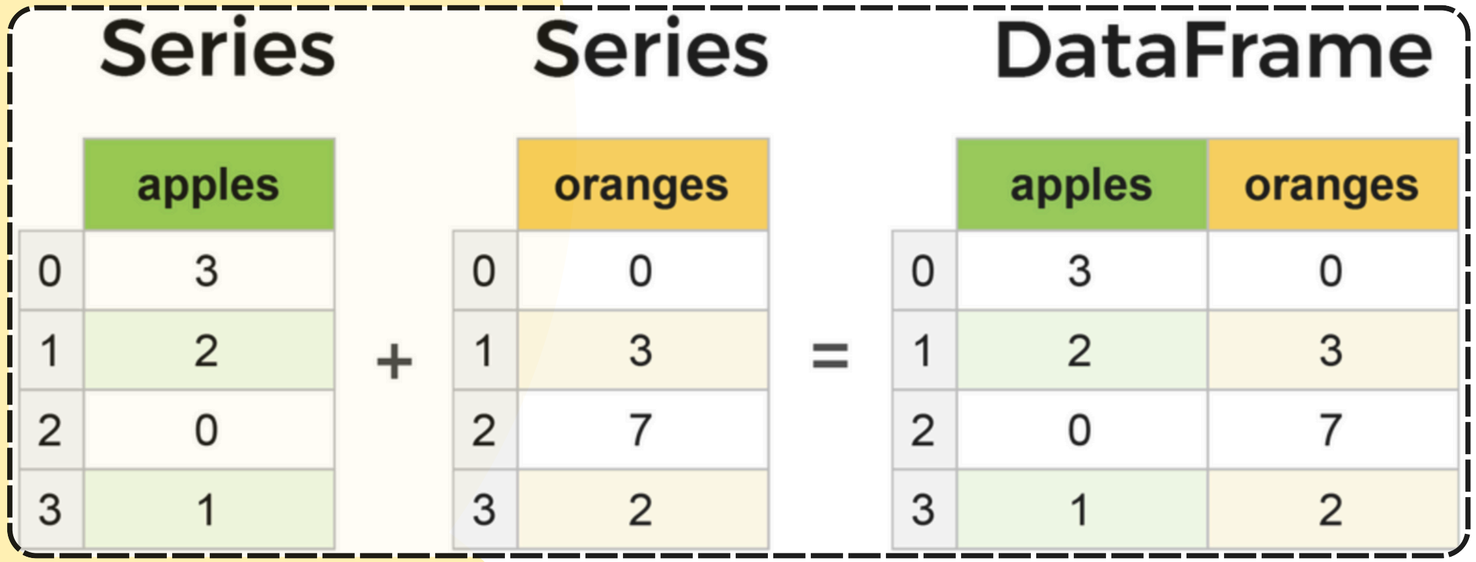
Series

	apples
0	3
1	2
2	0
3	1

- Tableau de données indicées en dimension 1.
- Objet avec deux attributs :
  - values,
  - index.
- La sélection des éléments se fait de la même façon que sous Numpy.
- Indiçage possible de manière explicite et implicite
  - index pas forcément des *intergers*
- *x = pd.Series(data, index=index)*

Dataframe

- Tableau de données indicées en dimension 2.
- Séquence d'objets *Series* alignés
- Deux attributs : *index* et *columns*.
- Un DataFrame associe un nom de colonne (une clé) à une Series (les données).



Index

- Contenu à la fois dans les objets Series et DataFrame.
- Information explicite pour référencer et modifier les données.
- Il peut contenir des valeurs qui se répètent.
- Partage les conventions des structure de données *set* afin de faciliter les opérations d'unions, intersections, différences, etc

Indiçage et sélection

Cas unidimensionel: *Series*

- Analogie du dictionnaire : application d'une collection de clés à une collection de valeurs.

- La clé est alors l'indice.

- Possibilité de modifier la *Series* en assignant une nouvelle association clé/valeur, comme ce serait le cas avec un dictionnaire.

Cas unidimensionel: *Series*

- Analogie du tableau de dimension 1 : tableau de données Numpy avec les mêmes mécaniques de sélection.
  - slicing : data['a':'c']
  - masking : data[(data > 0.3) & (data < 0.8)]
  - fancy indexing : data[['a', 'e']]

- Il existe deux attributs pour sélectionner:
  - loc : fait toujours référence à l'indice explicite.
  - iloc : fait toujours référence à l'indice implicite.

"Explicite est mieux qu'implicite" ⇒ Préférer cette façon de coder.

Cas multidimensionnel : DataFrame

- Analogie du dictionnaire :
  - clé = nom de la colonne
  - Possibilité d'ajouter une colonne par assignement.

- Analogie du tableau de dimension 2 :
  - Accès aux valeurs brutes via l'attribut values.
  - Opérations possibles sur le DataFrame (e.g. transposée :data.T, data.mean(), ...)
  - Sélection comme sur un objet ndarray.

Cas multidimensionnel : DataFrame

- On retrouve les deux schémas d'indiçage
  - loc : l'indice explicite, en utilisant les noms des indices et des
  - colonnes.
  - iloc : l'indice implicite, comme si le DataFrame était un
  - tableau Numpy.
- Toujours une virgule pour séparer les deux dimensions
- Tous les méchanismes de sélection Numpy peuvent s'utiliser avec ces façons d'indicer.

- NB : il existait auparavant un troisième schéma, hybride, ix, qui est maintenant déprécié.

Opérations

Opérations

- Pandas hérite des fonctionnalités de Numpy et de l'utilisation des *ufuncs*
  { sum(series), np.exp(series), ...}

- Les indices et étiquettes des colonnes sont préservés.

- Les indices sont alignés automatiquement

Opérations entre dataframe et series

- Suit les règles de broadcasting : opération similaire à celle entre un objet Numpy en dimension 2 avec un en dimension 1.
- L'alignement entre les colonnes et les indices est maintenu
- L'axe est précisé si besoin dans la méthode via l'option ***axis=***

Données manquantes

Conventions

Définir une valeur manquante :

- en utilisant un masque qui indique globallement les valeurs manquantes (un tableau de booléens séparé),
- via une valeur sentinelle qui indique une entrée absente (e.g. NaN)

Pandas :

- NaN pour les float.
- L'objet Python None

Données manquantes avec pandas

Traiter les valeurs manquantes

- Certaines opérations d'aggrégations deviennent impossible.
- Utilisation de fonction particulière permettant d'ignorer les NaN
  - **isnull**() : Génère un masque booléen indiquant les valeurs manquantes.
  - **notnull**() : Génère un masque booléen indiquant les valeurs qui ne sont pas manquantes.
  - **dropna**() : Retourne une version filtrée des données, sans les valeurs manquantes
  - **fillna**() : Retourne une copie des données avec les valeurs manquantes remplies
  - numpy : **np.nansum,** ...

Données manquantes avec pandas

dropna

- Enlever des NA implique supprimer toute une ligne ou toute une colonne.
- Possibilité de spécifier l'axe : ***axis=1 ou 0***
- Deux paramètres pour contrôler plus finement le nombre de
- données jetées :
  - ***how*** : par défaut any. Possibilité de choisir all (supprimer si une NA vs supprimer si tout NA).
  - ***thresh*** : nombre minimal de valeurs non nulles pour conserver une ligne/colonne.

Données manquantes avec pandas

- **fillna**
- Méthode permettant de remplir directement les valeurs manquantes par la valeur voulue : e.g. *df.fillna(0)*

- Différentes options pour différentes stratégies :
  - ffill : Modifier par la valeur précédente.
  - bfill : Modifier par la valeur suivante.

- D'autres stratégies possibles selon ce qui est recherché
  - (interpolation, moyenne, etc...)

Fusionner des datasets

Méthodes pandas

- concat
- merge
- ~~append~~

Pandas : concat

- pd.concat, similitude avec np.concatenate
- Fonctionne avec les Series et les DataFrame.
- On passe les objets à concaténer comme une liste en argument de la fonction : np.concat([obj1, obj2]).
- Les indices sont préservés.

Pandas : concat - paramètres

- join : si l'on veut une union ou une intersection des colonnes : 'inner' / 'outer'
- ignore_index : Création de nouveaux indices pour le résultat.
- verify_integrity : si True, affiche une erreur s'il y a des indices similaires

Pandas : merge

- concatenation de deux dataframes a la manière de la jointure dans une bdd

- Utile quand on a des informations complémentaires dans deux DataFrame différents sur des mêmes sujets et qu'on veut les regrouper.

- Regroupe les informations à partir d'une ou plusieurs clés

Pandas : merge - paramètres

- on : (la clé de jointure) un nom de colonne ou une liste de nom de colonnes.
- left_on et right_on : si une même clé a un nom différents sur chaque DataFrame, afin de créer la correspondance.
- left_index et right_index : utiliser les indices comme clé en place d'une colonne (booléen)
- how : *inner*, *outer*, *left*, *right*
- suffixes : suffixe pour les noms de colonnes en cas de conflit

Aggrégation et groupement

Simples Opérations

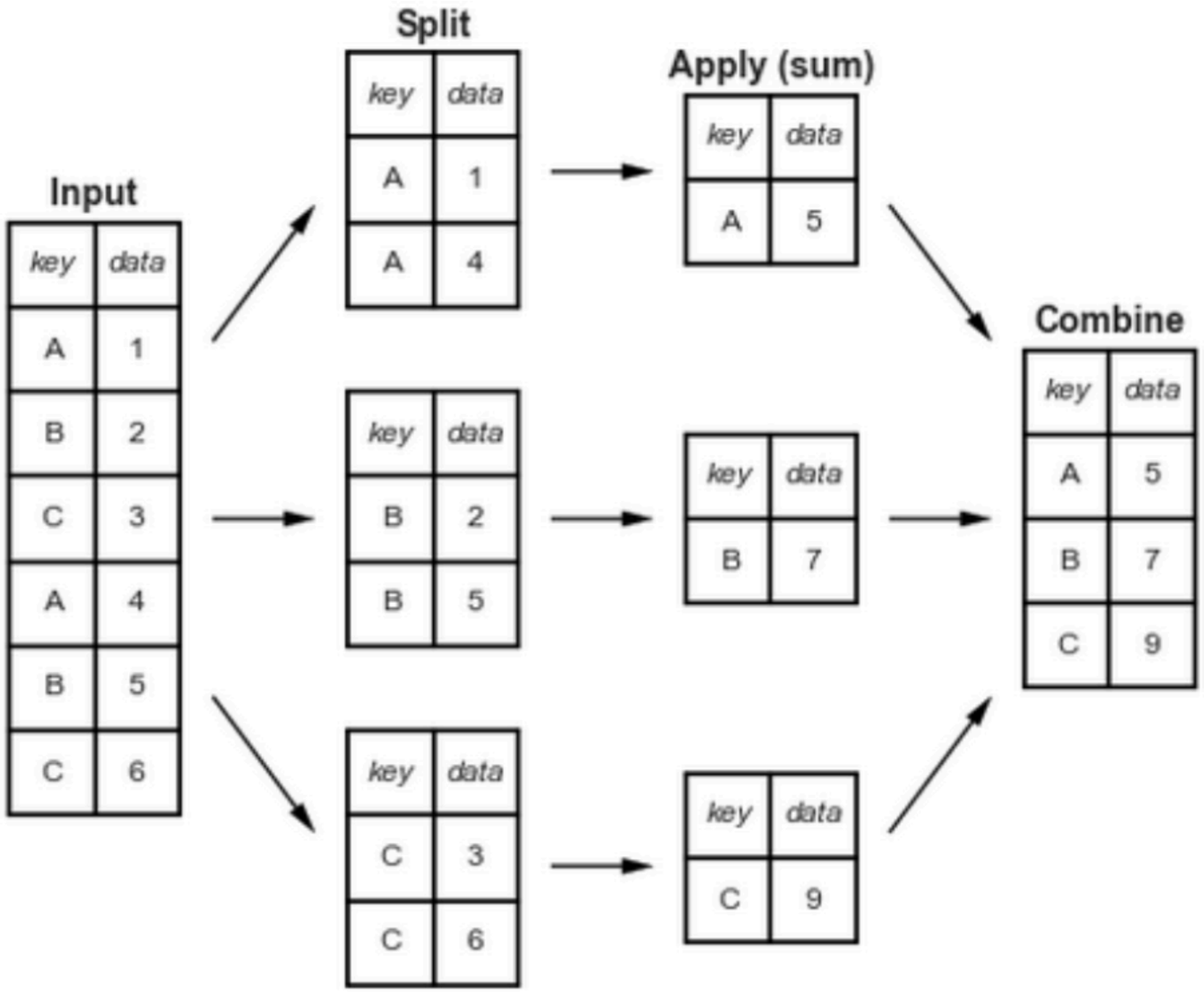
- On retrouve les fonctions d'aggrégation de Numpy en méthode.
- Avec un objet Series, pas de paramètre particulier.
- Avec un objet DataFrame, porssibilité de préciser l'axe.
- La méthode describe permet de calculer directement pour chaque colonne des statistiques descriptives

Aggregation	Description
`count()`	Total number of items
`first()`, `last()`	First and last item
`mean()`, `median()`	Mean and median
`min()`, `max()`	Minimum and maximum
`std()`, `var()`	Standard deviation and variance
`mad()`	Mean absolute deviation
`prod()`	Product of all items
`sum()`	Sum of all items

GroupBy

- Permet d’aggréger conditionnellement à une variable ou indice.
- La logique derrière : split, apply, combine.
- Diviser et grouper un DataFrame selon la valeur d’une clé.
- Calcul d’une fonction sur les différents groupes.
- Regroupement des résultats dans un tableau

GroupBy


Input
key data
A 1
B 2
C 3
A 4
B 5
C 6
Split
key data
A 1
A 4
key data
B 2
B 5
key data
C 3
C 6
Apply (sum)
key data
A 5
key data
B 7
key data
C 9
Combine
key data
A 5
B 7
C 9

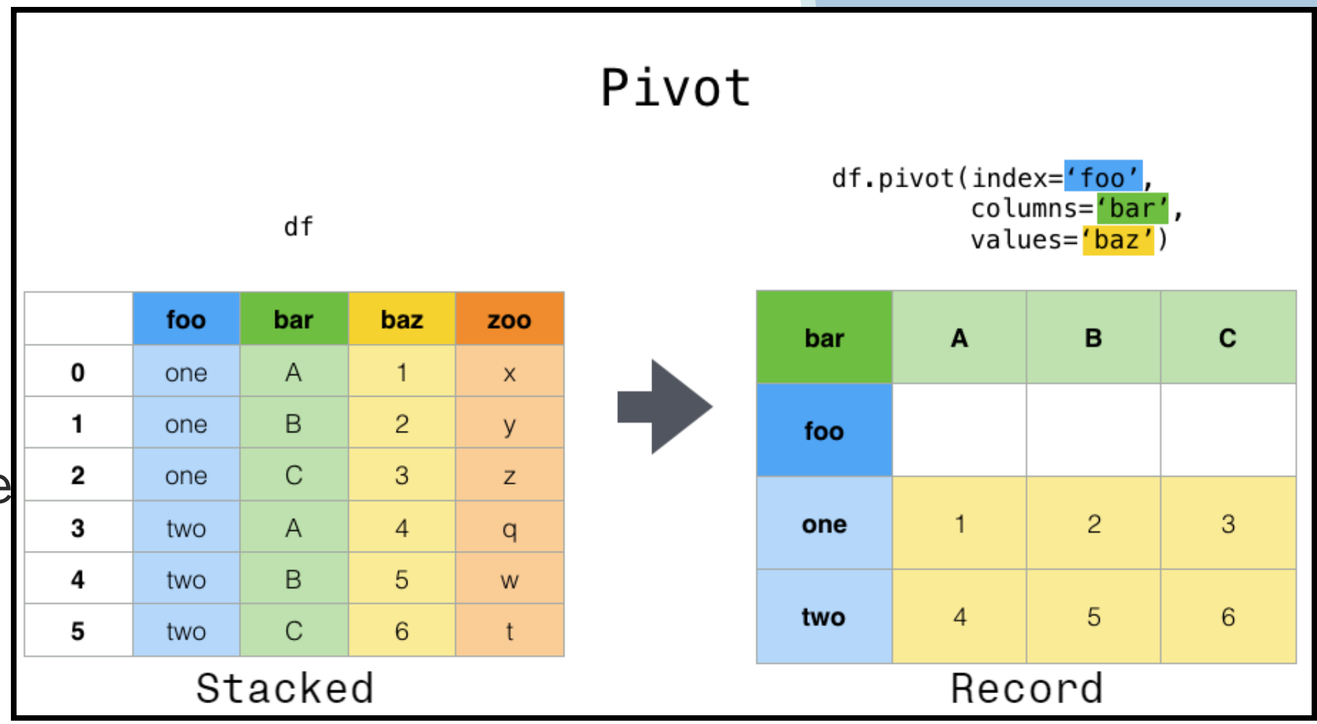
Pivot Table

- Utilisation de la méthode ***pivot_table***.
- Permet de faire des calcul d'aggrégation en dimension supérieure, version multidimensionnelle de GroupBy.

- split-apply-combine pas seulement sur les indices mais sur les deux dimensions.

Pivot Table

- On spécifie la/les variables, le/les indices et la/les colonnes en sortie.
- Chaque niveau des clés de groupement constituera une ligne/colonne.
- Spécification via aggfunc les fonctions appliquée (par défaut, moyenne).
- margins=True pour avoir les totaux par groupe.



	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

Opérations vectorisées sur des strings

Introduction aux opérations sur des strings

- Numpy ne permet pas de vectoriser les opérations sur des strings.
- Obligation à repasser par des boucles, plus verbeuses et moins efficientes.
- Problème pour gérer les valeurs manquantes.

- Les objets Series et Index ont un attribut **str** qui va permettre de correctement gérer les opérations vectorisées et les valeurs manquantes de ce type.
- La plupart des méthodes strings de Python se retrouve avec Pandas de manière vectorisée (*capitalize*, *lower*, *upper*...)

Méthodes Pandas

Method	Description
`match()`	Call `re.match()` on each element, returning a boolean.
`extract()`	Call `re.match()` on each element, returning matched groups as strings.
`findall()`	Call `re.findall()` on each element
`replace()`	Replace occurrences of pattern with some other string
`contains()`	Call `re.search()` on each element, returning a boolean
`count()`	Count occurrences of pattern
`split()`	Equivalent to `str.split()`, but accepts regexps

Méthodes Pandas

len()	lower()	translate()	islower()
ljust()	upper()	startswith()	isupper()
rjust()	find()	endswith()	isnumeric()
center()	rfind()	isalnum()	isdecimal()
zfill()	index()	isalpha()	split()
strip()	rindex()	isdigit()	rsplit()
rstrip()	capitalize()	isspace()	partition()
lstrip()	swapcase()	istitle()	rpartition()

Travailler avec des séries temporelles

Python native

- *datetime et dateutil*
- possibilité d'effectuer des opérations + et - entre datetimes

Pandas

- Objet DatetimeIndex pour indicer par le temps.
- Même façon de sélectionner les indices temporels que les autres.
- Possibilité de sélectionner qu'avec une partie de l'information temporelle (e.g. l'année).

- pd.to_datetime() : formattage de date

Pour créer des séquences temporelles régulières :

- pd.date_range() pour un horodatage.
- pd.period_range() pour une période.
- pd.timedelta_range() pour une duré

Aide Mémoire

Code	Description	Code	Description
D	Calendar day	B	Business day
W	Weekly		
M	Month end	BM	Business month end
Q	Quarter end	BQ	Business quarter end
A	Year end	BA	Business year end
H	Hours	BH	Business hours
T	Minutes		
S	Seconds		
L	Milliseonds		
U	Microseconds		
N	nanoseconds		

Code	Description	Code	Description
MS	Month start	BMS	Business month start
QS	Quarter start	BQS	Business quarter start
AS	Year start	BAS	Business year start