

Pratique de pandas: un exemple complet

Table des matières

I.	Introduction	2
II.	Exploration de la structure des données	3
A.	Exercice 1: Afficher des données	4
B.	Exercice 2: structure des données.....	5
III.	Les indices	7
A.	Exercice 3: Les indices.....	7
B.	Exercice 4: performance des indices	7
IV.	Restructurer les données	8
A.	Exercice 5: Restructurer les données: wide to long.....	8
B.	Exercice 6: long to wide.....	9
V.	Combiner les données	9
A.	Exercice 7: Calculer l'empreinte carbone par habitant	10

I. Introduction

Dans ce TP, nous allons utiliser deux sources de données :

Les émissions de gaz à effet de serre estimées au niveau communal par l'**ADEME**. Le jeu de données est disponible sur [data.gouv](#) et requêtable directement dans python avec [cet url](#). pandas offre la possibilité d'importer des données directement depuis un url. C'est l'option prise dans ce tutoriel. Si vous préférez, pour des raisons d'accès au réseau ou de performance, importer depuis un poste local, vous pouvez télécharger les données et changer les commandes d'import avec le chemin adéquat plutôt que l'url.

Idéalement, on utiliserait directement les données [disponibles sur le site de l'Insee](#) mais celles-ci nécessitent un peu de travail de nettoyage qui n'entre pas dans le cadre de ce TP. Pour faciliter l'import de données Insee, il est recommandé d'utiliser le package [pynsee](#) qui simplifie l'accès aux données de l'Insee disponibles sur le site web [insee.fr](#) ou via des API1.

Après avoir installé la librairie **pynsee** nous suivrons les conventions habituelles dans l'import des packages :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pynsee
import pynsee.download
```

II. Exploration de la structure des données

Commencer par importer les données de l'**Ademe** à l'aide du package **pandas**. Vous pouvez nommer le DataFrame obtenu **df**.

```
df = pd.read_csv("https://koumoul.com/s/data-fair/api/v1/datasets/igt-pouvoir-de-  
rechauffement-global/convert", sep=",")
```

Pour les données de cadrage au niveau communal (source Insee), le package **pynsee** facilite grandement la vie. La liste des données disponibles est [ici](#). En l'occurrence, on va utiliser les données **Filosofi (données de revenus)** au niveau communal de 2016. Le point d'entrée principal de la fonction **pynsee** est la fonction **download_file**. Le code pour télécharger les données est le suivant :

```
df_city = pynsee.download.download_file("FILOSOFI_COM_2016")
```

Note

La fonction **download_file** attend un identifiant unique pour savoir quelle base de données aller chercher et restructurer depuis le **site insee.fr**.

Pour connaître la liste des bases disponibles, vous pouvez utiliser la fonction

```
meta = pynsee.get_file_list().
```

Celle-ci renvoie un DataFrame dans lequel on peut rechercher, par exemple grâce à une recherche de mot clé:

```
meta = pynsee.get_file_list()
```

```
meta.loc[meta['label'].str.contains(r"Filosofi.*2016")]
```

Ici, `meta['label'].str.contains(r"Filosofi.*2016")` signifie: *“pandas trouve moi tous les labels où sont contenus les termes Filosofi et 2016.”*

(.* signifiant “peu m’importe le nombre de mots ou caractères entre”)

A. Exercice 1: Afficher des données

L'objectif de cet exercice est de vous amener à afficher des informations sur les données dans un bloc de code (notebook) ou dans la console

Commencer sur **df**:

1. Utiliser les méthodes adéquates pour les 10 premières valeurs, les 15 dernières et un échantillon aléatoire de 10 valeurs
2. Tirer 5 pourcent de l'échantillon sans remise
3. Ne conserver que les 10 premières lignes et tirer aléatoirement dans celles-ci pour obtenir un DataFrame de 100 données.
4. Faire 100 tirages à partir des 6 premières lignes avec une probabilité de 1/2 pour la première observation et une probabilité uniforme pour les autres
5. Faire la même chose sur `df_city`.

Cette première approche exploratoire donne une idée assez précise de la manière dont les données sont organisées. On remarque ainsi une différence entre `df` et `df_city` quant aux valeurs manquantes : la première base est relativement complète, la seconde comporte beaucoup de valeurs manquantes. Autrement dit, si on désire exploiter `df_city`, il faut faire attention à la variable choisie.

B. Exercice 2: structure des données

La première chose à vérifier est le format des données, afin d'identifier des types de variables qui ne conviennent pas.

Ici, comme c'est pandas qui a géré automatiquement les types de variables, il y a peu de chances que les types ne soient pas adéquats mais une vérification ne fait pas de mal.

1. Vérifier les types des variables. S'assurer que les types des variables communes aux deux bases sont cohérents. Pour les variables qui ne sont pas en type float alors qu'elles devraient l'être, modifier leur type.

Ensuite, on vérifie les dimensions des DataFrames et la structure de certaines variables clés. En l'occurrence, les variables fondamentales pour lier nos données sont les variables communales. Ici, on a deux variables géographiques: un code commune et un nom de commune.

2. Vérifier les dimensions des **DataFrames**
3. Vérifier le nombre de valeurs uniques des variables géographiques dans chaque base. Les résultats apparaissent-ils cohérents ?
4. Identifier dans **df_city** les noms de communes qui correspondent à plusieurs codes communes et sélectionner leurs codes. En d'autres termes, identifier les CODGEO tels qu'il existe des doublons de LIBGEO et les stocker dans un vecteur x (conseil: faire attention à l'index de x)

On se focalise temporairement sur les observations où le libellé comporte plus de deux codes communes différents

5. Regarder dans df_city ces observations
6. Pour mieux y voir, réordonner la base obtenue par ordre alphabétique



7. Déterminer la taille moyenne (variable nombre de personnes: **NBPERSMENFISC16**) et quelques statistiques descriptives de ces données. Comparer aux mêmes statistiques sur les données où libellés et codes communes coïncident
8. Vérifier les grandes villes (plus de 100 000 personnes), la proportion de villes pour lesquelles un même nom est associé à différents codes commune.
9. Vérifier dans `df_city` les villes dont le libellé est égal à Montreuil. Vérifier également celles qui contiennent le terme 'Saint-Denis'

Ce petit exercice permet de se rassurer car les libellés dupliqués sont en fait des noms de commune identiques mais qui ne sont pas dans le même département. Il ne s'agit donc pas d'observations dupliquées. On se fiera ainsi aux codes communes, qui eux sont uniques.

III. Les indices

Les indices sont des éléments spéciaux d'un **DataFrame** puisqu'ils permettent d'identifier certaines observations. Il est tout à fait possible d'utiliser plusieurs indices, par exemple si on a des niveaux imbriqués.

A. Exercice 3: Les indices

A partir de l'exercice précédent, on peut se fier aux codes communes.

1. Fixer comme indice la variable de code commune dans les deux bases. Regarder le changement que cela induit sur le display du DataFrame
2. Les deux premiers chiffres des codes communes sont le numéro de département. Créer une variable de département `dep` dans `df` et dans `df_city`
3. Calculer les émissions totales par secteur pour chaque département. Mettre en log ces résultats dans un objet `df_log`. Garder 5 départements et produire un barplot
4. Repartir de `df`. Calculer les émissions totales par département et sortir la liste des 10 principaux émetteurs de CO2 et des 5 départements les moins émetteurs. Sans faire de merge, regarder les caractéristiques de ces départements (population et niveau de vie)

B. Exercice 4: performance des indices

Un des intérêts des indices est qu'ils permettent des agrégations efficaces.

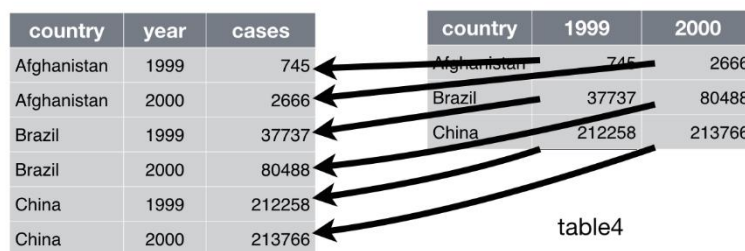
1. Repartir de `df` et créer une copie `df_copy = df.copy()` et `df_copy2 = df.copy()` afin de ne pas écraser le DataFrame `df`
2. Utiliser la variable `dep` comme indice pour `df_copy` et retirer tout index pour `df_copy2`
3. Importer le module `timeit` et comparer le temps d'exécution de la somme par secteur, pour chaque département, des émissions de CO2

IV. Restructurer les données

On présente généralement deux types de données :

- format wide: les données comportent des observations répétées, pour un même individu (ou groupe), dans des colonnes différentes
- format long: les données comportent des observations répétées, pour un même individu, dans des lignes différentes avec une colonne permettant de distinguer les niveaux d'observations

Un exemple de la distinction entre les deux peut être pris à l'ouvrage de référence d'Hadley Wickham, R for Data Science:



country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

L'aide-mémoire suivante aidera à se rappeler les fonctions à appliquer si besoin:

Le fait de passer d'un format wide au format long (ou vice-versa) peut être extrêmement pratique car certaines fonctions sont plus adéquates sur une forme de données ou sur l'autre. En règle générale, avec Python comme avec R, les formats long sont souvent préférables.

A. Exercice 5: Restructurer les données: wide to long

1. Créer une copie des données de l'ADEME en faisant `df_wide = df.copy()`
2. Restructurer les données au format long pour avoir des données d'émissions par secteur en gardant comme niveau d'analyse la commune (attention aux autres variables identifiantes).
3. Faire la somme par secteur et représenter graphiquement
4. Garder, pour chaque département, le secteur le plus polluant

B. Exercice 6: long to wide

Cette transformation est moins fréquente car appliquer des fonctions par groupe, comme nous le verrons par la suite, est très simple.

1. Repartir de **df_wide = df.copy()**
2. Reconstruire le **DataFrame**, au format long, des données d'émissions par secteur en gardant comme niveau d'analyse la commune puis faire la somme par département et secteur
3. Passer au format **wide** pour avoir une ligne par secteur et une colonne par département
4. Calculer, pour chaque secteur, la place du département dans la hiérarchie des émissions nationales
5. A partir de là, en déduire le rang médian de chaque département dans la hiérarchie des émissions et regarder les 10 plus mauvais élèves, selon ce critère.

V. Combiner les données

Une information que l'on cherche à obtenir s'obtient de moins en moins à partir d'une unique base de données. Il devient commun de devoir combiner des données issues de sources différentes. Nous allons ici nous focaliser sur le cas le plus favorable qui est la situation où une information permet d'apparier de manière exacte deux bases de données (autrement nous serions dans une situation, beaucoup plus complexe, d'appariement flou). La situation typique est l'appariement entre deux sources de données selon un identifiant individuel ou un identifiant de code commune, ce qui est notre cas.

On utilise de manière indifférente les termes **merge** ou **join**. Le deuxième terme provient de la syntaxe SQL. En pandas, dans la plupart des cas, on peut utiliser indifféremment **df.join** et **df.merge**

A. Exercice 7: Calculer l'empreinte carbone par habitant

1. Créer une variable **emissions** qui correspond aux émissions totales d'une commune
2. Faire une jointure à gauche entre les données d'émissions et les données de cadrage.
Comparer les émissions moyennes des villes sans match (celles dont des variables bien choisies de la table de droite sont NaN) avec celles où on a bien une valeur correspondante dans la base Insee
3. Faire un **inner join** puis calculer l'empreinte carbone (l'émission rapportée au nombre de ménages fiscaux) dans chaque commune. Sortir un histogramme en niveau puis en log et quelques statistiques descriptives sur le sujet.
4. Regarder la corrélation entre les variables de cadrage et l'empreinte carbone.
Certaines variables semblent-elles pouvoir potentiellement influencer sur l'empreinte carbone ?