

Le traitement de données structurées

Après avoir découvert le format CSV, nous allons maintenant, à l'aide de Python, apprendre à effectuer des traitements sur ces données.

Pour traiter des données, nous allons utiliser la bibliothèque Python *Pandas*.

Nous allons utiliser des données très simples au format CSV :

Expérimentation :

-Téléchargez l'archive [CSV_traitement.zip](http://ressource.elec.free.fr/softs.html) (<http://ressource.elec.free.fr/softs.html>) puis décompressez-la dans le dossier de votre choix (par exemple dans un dossier nommé "NSI_pandas").

Vous avez deux fichiers CSV sur lesquels nous allons travailler : **ident_virgule.csv** et **ville_virgule.csv** (nous allons d'abord travailler sur **ident_virgule.csv**)

-Ouvrez Anaconda puis Spyder ou Pycharm. Dans la partie "Éditeur de texte" de Spyder, saisissez le code Python suivant :

```
import pandas
iden=pandas.read_csv("ident_virgule.csv")
```

-Une fois le code saisi, enregistrez le fichier contenant ce code dans le même répertoire que le fichier "ident_virgule.csv" (sous le nom que vous voulez)

Ce que fait le code ci-dessus:

Avec la première ligne, nous importons la bibliothèque *pandas* afin de pouvoir l'utiliser.

À la deuxième ligne, nous créons une variable "**iden**" qui va contenir les données présentes dans le fichier "ident_virgule.csv"

-Exécutez le programme que vous venez de saisir.

-Placez ensuite le curseur de la souris dans la console de Spyder juste à côté d'un "In [X]" (avec X égal à 1, 2, 3..., selon les cas, dans l'exemple ci-dessous nous avons "In [3]"). Tapez alors "iden".

Vous devriez voir apparaître les données contenues dans la variable "iden" rangées sous la forme d'un tableau, un peu comme ce que nous obtenions en ouvrant le fichier "ident_virgule.csv" avec un tableur.

Une colonne a été ajoutée par rapport à ce que nous obtenions avec le tableur :

```
In [3]: iden
Out[3]:
```

	nom	prenom	date_naissance
0	Durand	Jean-Pierre	23/05/1985
1	Dupont	Christophe	15/12/1967
2	Terta	Henry	12/06/1978

Les nombres présents dans cette colonne sont appelés des **index**. Chaque ligne du tableau a un index (première ligne : index 0, deuxième ligne index 1...)

ATTENTION : les index commencent à 0 et pas à 1

Les colonnes possèdent également des index, dans notre exemple ces index correspondent au "nom" (index de la première colonne), au "prenom" (index de la deuxième colonne) et à "date_naissance" (index de la troisième colonne)

En résumé : **les lignes possèdent des index (0,1,2..), les colonnes possèdent aussi des index ("nom", "prenom",...)**

Il est possible de **récupérer certaines données du tableau**, par exemple, certaines lignes, certaines colonnes ou bien encore des valeurs uniques. Pour cela, il suffit d'utiliser l'instruction "**loc**" avec les index des lignes et les index des colonnes.

loc[index_ligne,index_colonne]

Expérimentation :

-Testez le programme suivant :

```
import pandas
iden=pandas.read_csv("ident_virgule.csv")
info=iden.loc[1,'prenom']
```

-Vérifiez que la variable "info" contient bien le prénom "christophe" (tapez *info* dans la console Spyder)

-Modifiez le programme pour que la variable info contienne "12/06/1978"

Il est possible de **récupérer toutes les lignes d'une colonne**, il suffit de remplacer la partie "index_ligne" de "loc" par ":"

-Testez le programme suivant et vérifiez que la variable "info" contient bien toutes les données de la colonne d'index "nom", autrement dit, tous les noms:

```
import pandas
iden=pandas.read_csv("ident_virgule.csv")
info=iden.loc[:, 'nom']
```

Il est possible de **récupérer toutes les colonnes d'une ligne particulière**, cette fois en remplaçant la partie "index_colonne" de "loc" par ":"

-Testez le programme suivant et vérifiez que la variable "info" contient bien toutes les données de la dernière ligne (index 2):

```
import pandas
iden=pandas.read_csv("ident_virgule.csv")
info=iden.loc[2,:]
```

Il est aussi possible de **récupérer seulement certaines lignes et certaines colonnes** en utilisant la notation suivante :

loc[[index_ligne_1,index_ligne_2,...],[index_colonne_1,index_colonne_2,...]]

-Testez le programme suivant et vérifiez que la variable "info" contient bien un tableau avec uniquement les colonnes "nom" et "date_naissance" de la première ligne (index 0) et de la deuxième ligne (index 1) :

```
import pandas
iden=pandas.read_csv("ident_virgule.csv")
info=iden.loc[[0,1],['nom','date_naissance']]
```

Travail avec un fichier CSV un peu plus gros...

Afin d'avoir des exemples plus complexes à traiter, dans la suite, nous allons travailler sur les données contenues dans le fichier *ville_virgule.csv*.

-Testez le programme suivant :

```
import pandas
info_villes=pandas.read_csv("villes_virgule.csv")
```

Vérifiez que la variable "info_villes" contient bien les données contenues dans le fichier *ville_virgule.csv*.

Comme vous pouvez le constater, il manque des données dans le tableau qui s'affiche dans la console spyder (les données manquantes sont symbolisées par des ...), en effet, le tableau contient trop données pour qu'il soit entièrement affiché. Il existe une solution :

Dans spyder, dans la fenêtre située juste au-dessus de la console, vous allez trouver un onglet "**Explorateur de variables**". Cliquez sur cet onglet, vous devriez alors obtenir ceci :

Nom	Type	Taille	Valeur
info_villes	DataFrame	(36700, 12)	Column names: dep, nom, cp, nb_hab_2010, nb_hab_1999, nb_hab_2012

Double-cliquez sur "info_villes" et vous devriez alors voir apparaitre une nouvelle fenêtre qui contiendra un tableau avec l'ensemble des données.

En explorant le tableau, vous devriez, notamment dans les colonnes l'altitude mini et maxi, voir apparaitre un étrange "nan" pour les dernières villes du tableau. **"nan" signifie "not a number"**, ici, cela veut tout simplement dire que certaines données sont manquantes.

Nous allons maintenant introduire des conditions dans la sélection des villes. Imaginez par exemple que vous désirez obtenir un tableau contenant toutes les villes qui ont une altitude minimum supérieure à 1500 m :

-Analysez et testez la ligne suivante :

```
nom_alt=info_villes.loc[info_villes["alt_min"]>1500,["nom","alt_min"]]
```

Dans le "loc", l'expression "info_villes["alt_min"]>1500" est bien avant la virgule, elle concerne donc les index des lignes du tableau. On sélectionnera uniquement les lignes qui auront la valeur du descripteur "alt_min" supérieure à 1500. Nous allons donc bien sélectionner les villes qui ont une altitude minimum supérieure à 1500 m

-En vous inspirant de ce qui a été fait au dessus, écrivez un programme qui permettra d'avoir les villes qui ont une densité d'habitant inférieure à 2 (dans le tableau ainsi créé, on aura 4 colonnes : le nom de la ville, la densité de la population, le code postal et l'altitude minimum) et qui détermine le nombre de villes ayant cette caractéristique.

Il est possible de **combinaer plusieurs facteurs de sélection** en utilisant un **"et"("&")** ou un **"ou"("|")**.

Analysez et testez le programme suivant :

```
nom_alt=info_villes.loc[(info_villes["alt_min"]>1500) &  
(info_villes["dens"]>50),["nom","dens","alt_min"]]
```

Vous devriez constater qu'il y a, en France, une seule ville avec une densité de population supérieure à 50 et une altitude minimum supérieure à 1500 m.

Il est aussi possible d'effectuer des **calculs sur des colonnes**, par exemple des moyennes. Il suffit d'utiliser l'instruction **"mean" pour effectuer une moyenne** :

-Analysez et testez le programme suivant :

```
moyenne_alt_min=info_villes.loc[:, "alt_min"].mean()
```

Vous devriez constater que l'altitude minimum moyenne est de 193 m en France. Je rappelle que dans `loc[:, "alt_min"]` le `:` signifie que l'on considère toutes les lignes du tableau. De plus le `"alt_min"` que le calcul de la moyenne porte bien sur les données du descripteur `"alt_min"`.

-Écrivez un programme permettant de calculer le nombre moyen d'habitants en 2012

Pour l'instant nous avons calculé une moyenne sur l'ensemble des lignes, il est aussi possible **d'imposer une condition sur les lignes** qui seront utilisées pour le calcul.

-Analysez et testez le programme suivant :

```
nbe_hab=info_villes.loc[info_villes["alt_min"]>1500,"nb_hab_2012"].mean()  
print(nbe_hab)
```

Vous devriez constater que les villes ayant une altitude minimum supérieure à 1500 m avaient en moyenne 350 habitants en 2012.

Il est aussi possible de **trier le tableau en fonction des valeurs d'un descripteur**. Il suffit d'utiliser l'instruction **"sort_values"**

-Analysez et testez le programme suivant :

```
tri_alt_min=info_villes.sort_values(by=["alt_min"])
```

Vous devriez obtenir un nouveau tableau de données `"tri_alt_min"` trié dans l'ordre croissant des altitudes minimums. Quelle est la ville ayant l'altitude minimum la plus faible de France ?

Il est aussi possible de trier par ordre décroissant en ajoutant **"ascending=False"** :

-Analysez et testez le programme suivant :

```
tri_alt_min=info_villes.sort_values(by=["alt_min"], ascending=False)
```

Quelle est la ville ayant l'altitude minimum la plus importante de France ?

Il est possible de **fusionner 2 tableaux de données** qui ont une colonne commune :
Afin de travailler sur cette fusion, nous allons travailler avec 2 fichiers au format CSV
: *fiches_client.csv* et *fiches_com.csv*.

- testez le code suivant :

```
import pandas
client=pandas.read_csv("fiches_client.csv")
commande=pandas.read_csv("fiches_com.csv")
```

Utilisez l'explorateur de variables de Spyder afin d'afficher le contenu des variables "client" et "commande"

Vous devriez normalement obtenir pour "client" :

Index	n_client	nom	prenom	ville
0	1212	Lacasse	Aubrey	Annecy
1	1343	Primeau	Angelette	Tours
2	2454	Gabriaux	Julie	Bordeaux
3	895	Gaulin	Élodie	Lyon
4	2324	Jobin	Dorene	Bourges
5	34	Boncoeur	Kari	Nantes
6	1221	Parizeau	Olympia	Metz
7	1114	Païement	Inès	Bordeaux
8	3435	Chrétien	Adèle	Moulin
9	5565	Neufville	Ila	Toulouse
10	2221	Larivière	Alice	Tours

et pour "commande" :

Index	n_commande	date	n_client
0	1111	22/09/18	1343
1	2323	20/03/19	34
2	987	12/09/15	5565
3	454	08/07/14	2324
4	1324	01/02/17	4444
5	1567	05/12/18	2221
6	45	02/02/12	2454
7	123	04/11/13	5565
8	2122	12/02/19	3435
9	1989	04/12/18	1212

Nous avons un tableau qui référence les clients (nom, prénom, ville), chaque client possède un numéro de client.

Le deuxième tableau référence des commandes : pour chaque commande, nous avons un numéro de commande, une date et le numéro du client qui a passé la commande, **ce numéro de client correspond évidemment au numéro de client que l'on trouve dans le premier tableau.**

Sachant que nous avons deux colonnes contenant les mêmes types d'information (numéros de client), nous allons pouvoir fusionner les deux tableaux en un seul :

-Testez le code suivant :

```
client=pandas.read_csv("fiches_client.csv")
commande=pandas.read_csv("fiches_com.csv")
cl_com=pandas.merge(client, commande)
```

Utilisez l'explorateur de variables de Spyder afin d'afficher le contenu de la variable "cl_com"

Vous devriez obtenir ceci :

Index	n_client	nom	prenom	ville	n_commande	date
0	1212	Lacasse	Aubrey	Annecy	1989	04/12/18
1	1343	Primeau	Angelette	Tours	1111	22/09/18
2	2454	Gabriaux	Julie	Bordeaux	45	02/02/12
3	2324	Jobin	Dorene	Bourges	454	08/07/14
4	34	Boncoeur	Kari	Nantes	2323	20/03/19
5	3435	Chrétien	Adèle	Moulin	2122	12/02/19
6	5565	Neufville	Ila	Toulouse	987	12/09/15
7	5565	Neufville	Ila	Toulouse	123	04/11/13
8	2221	Larivière	Alice	Tours	1567	05/12/18

Prenons l'exemple de Mme Julie Gabriaux qui habite à Bordeaux (n° de client 2454) et de la commande effectuée le 02/02/2012 par le client ayant le n° 2454 (commande n° 45). La

cliente qui a passé cette commande n° 45 est bien Mme Gabriaux, nous avons une ligne dans notre tableau "cl-com" :

2	2454	Gabriaux	Julie	Bordeaux	45	02/02/12
---	------	----------	-------	----------	----	----------

Nous avons bien fusionné les 2 tableaux "client" et "commande" en un seul tableau "cl_com" qui regroupe les informations pour chaque commande. Quand on effectue ce genre de fusion, on dit souvent que l'on effectue une jointure.

Il faut prendre garde à l'ordre des arguments de la fonction "merge" :

-Testez le code suivant :

```
client=pandas.read_csv("fiches_client.csv")
commande=pandas.read_csv("fiches_com.csv")
com_cl=pandas.merge(commande, client)
```

Utilisez l'explorateur de variables de Spyder afin d'afficher le contenu de la variable "com_cl"

Vous devriez obtenir ceci :

Index	n_commande	date	n_client	nom	prenom	ville
0	1111	22/09/18	1343	Primeau	Angelette	Tours
1	2323	20/03/19	34	Boncoeur	Kari	Nantes
2	987	12/09/15	5565	Neufville	Ila	Toulouse
3	123	04/11/13	5565	Neufville	Ila	Toulouse
4	454	08/07/14	2324	Jobin	Dorene	Bourges
5	1567	05/12/18	2221	Larivière	Alice	Tours
6	45	02/02/12	2454	Gabriaux	Julie	Bordeaux
7	2122	12/02/19	3435	Chrétien	Adèle	Moulin
8	1989	04/12/18	1212	Lacasse	Aubrey	Annecy

Comme vous pouvez le constater, l'ordre des colonnes est différent. Il faudra donc être attentif à l'ordre des paramètres de la fonction "merge".

Remarque : On trouve Mme Ila Neufville sur 2 lignes, car elle a passé 2 commandes.

Vous avez peut-être remarqué que Mme Élodie Gaulin (n° de client 895) bien que présente dans le tableau "client", est absente du tableau "com_cl" (ou "cl_com"). Pourquoi d'après vous ?

De la même manière, aucun trace de la commande n° 1324 du 01/02/2017 dans le tableau "com_cl" (ou "cl_com"), pourquoi d'après vous ?

Comme nous venons de le voir ci-dessus, **il faut que l'élément qui permet la jointure (ici un numéro de client) soit présent dans les 2 tableaux** : dans nos exemples vous avez sans doute remarqué que :

Mme Gaulin n'a pas passé de commande, son numéro de client est absent du tableau "commande", d'où son absence du tableau "com_cl" (ou "cl_com"). À noter que nous avons 2 autres clientes qui n'ont jamais passé de commande.

Le client qui a passé la commande n°1324 (n° client 4444) est absent du tableau "client", d'où, ici aussi, son absence du tableau "com_cl" (ou "cl_com")

d'autres méthodes que Pandas :

Il existe des autres bibliothèques pouvant nous aider à traiter des fichiers CSV

Parmi celles-ci, une nommée ... CSV .

Exemple de programme utilisant CSV , avec un fichier dont les délimiteurs de ligne sont des espaces, et les séparateurs d'items sont des | :

```
import csv
with open('eggs.csv', newline='') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
    for row in spamreader:
        print(', '.join(row))
```

On peut plus simplement ouvrir notre fichier CSV comme on ouvrirait un fichier quelconque (comme on avait fait par exemple dans notre TP image), en utilisant **os** :

```
import os
mon_fichier = open("exemple.csv", "r")
texte = mon_fichier.read()
mon_fichier.close()
```

Ici, on a ouvert le fichier en mode lecture (**r** pour read)

Le mode est donné sous la forme d'une chaîne de caractères. Voici les principaux modes :

- 'r': ouverture en lecture (Read).
- 'w': ouverture en écriture (Write). Le contenu du fichier est écrasé. Si le fichier n'existe pas, il est créé.
- 'a': ouverture en écriture en mode ajout (Append). On écrit à la fin du fichier sans écraser l'ancien contenu du fichier. Si le fichier n'existe pas, il est créé.

On peut ajouter à tous ces modes le signe **b** pour ouvrir le fichier en mode binaire. Nous en verrons plus loin l'utilité, c'est un mode un peu particulier.

Après, en utilisant la méthode *read* , on récupère une chaîne de caractère dont on fera ce qu'on veut.

Ne pas oublier de fermer le fichier après utilisation.