Lire et écrire des fichiers avec Pandas

Il est important que vous appreniez à lire des données à partir de fichiers avec Pandas, avant même de découvrir ses structures de données. En effet, pour la suite, il faut que vous puissiez importer le jeu de données concernant les Jeux olympiques dans votre notebook Jupyter.

1. Lecture de fichiers texte (CSV ou TXT)

Lorsqu'on souhaite utiliser Pandas pour analyser des données, la première étape est généralement d'importer les données dans son notebook. Les fichiers peuvent être de différents formats, mais le plus courant en *Data Science* est le format CSV. CSV signifie *comma-separated values*, traduisez : "valeurs séparées par une virgule". Il s'agit donc d'un format texte, dans lequel les colonnes sont séparées par des virgules. L'autre format très courant est le format TXT, qui lui aussi est un format texte, mais contenant des valeurs séparées par des tabulations, cette fois.

a. Lecture basique d'un fichier

Pour lire un fichier TXT avec Pandas, il faut utiliser la fonction `read_table()`, dont la syntaxe générale est la suivante :

```
import pandas as pd
pd.read_table("mon_fichier.txt")
```

Il est courant d'utiliser l'alias `pd` pour pandas, ce qui permet de réduire le code d'appel des fonctions appartenant à cette librairie.

La fonction `read_table()` possède un argument obligatoire, qui est le chemin complet vers le fichier ainsi que le nom du fichier, ou juste le nom du fichier si celui-ci se trouve dans votre dossier de travail, c'est-à-dire dans le même dossier que votre notebook. Dans

cette syntaxe, le fichier `mon_fichier.txt` se trouve dans le même dossier que le notebook.

Pour lire un fichier CSV avec Pandas, il faut utiliser la fonction `read_csv()`, dont la syntaxe générale est la suivante :

```
import pandas as pd
pd.read_csv("mon_fichier.csv")
```

Ces deux fonctions sont les plus utilisées pour importer des données à partir d'un fichier. La seule différence entre ces deux fonctions est la valeur par défaut que prend l'option `sep`. En effet, l'option `sep`, qui signifie séparateur, permet de dire à Pandas quel est le séparateur du fichier qu'on souhaite lire. Dans la fonction `read_table()`, la valeur par défaut de `sep` est la tabulation (qu'on écrit `\t` en Python), donc cette fonction permet de lire un fichier TXT par défaut. Dans la fonction `read_csv()`, la valeur par défaut de `sep` est la virgule (`,`), donc elle permet de lire un fichier CSV par défaut.

Toutefois, cette valeur de `sep` peut être modifiée, et la fonction `read_table()`, par exemple, peut être utilisée pour lire un fichier CSV, comme suit :

```
import pandas as pd
pd.read_table("mon_fichier.csv", sep=",")
```

Ainsi, la fonction `read_table()` peut s'adapter à tout type de séparateur, que ce soit le point-virgule (`;`), l'espace (" "), etc., et il en va de même pour la fonction `read_csv()`. Il est simplement logique d'utiliser la fonction `read_table()` pour des fichiers TXT et la fonction `read_csv()` pour des fichiers CSV.

À présent, importons notre jeu de données exemple dans notre notebook, avec la commande suivante :

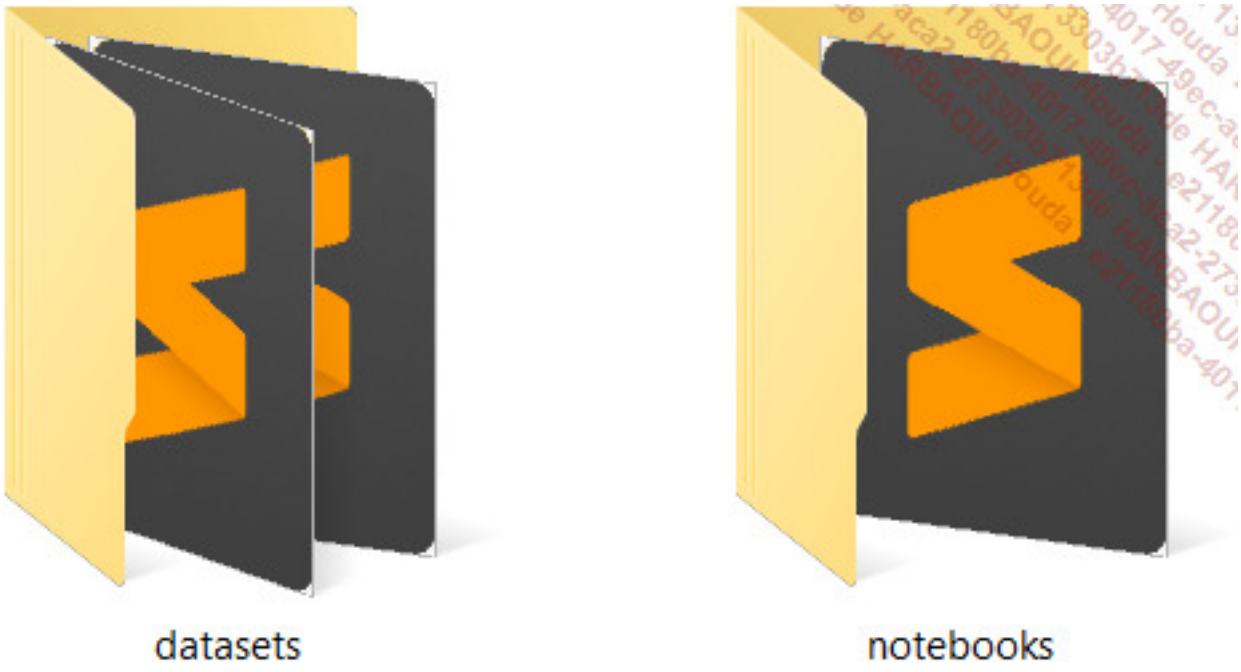
```
import pandas as pd
donnees=pd.read_csv("../datasets/120-years-of-olympic-history
-athletes-and-results.csv")
```

```
donnees.head()
```

Avec cette commande, nous importons la librairie Pandas, en lui donnant l'alias `pd` pour simplifier l'écriture. Ensuite, nous lisons le fichier de données avec la fonction `read_csv()`, et nous le stockons dans un dataframe nommé `donnees`.

Nous verrons plus en détail dans la section suivante ce que sont les dataframes et les séries. Pour l'instant, retenez simplement qu'il s'agit d'un tableau à deux dimensions pour le premier et d'un tableau à une dimension pour le deuxième.

Comme expliqué précédemment, il faut donner le chemin vers le fichier à la fonction `read_csv()` pour qu'elle puisse le lire. Voici la structure de notre dossier de travail :



Dans le dossier `datasets` (qui signifie jeux de données, en français), il y a nos fichiers de données (dont notre fichier exemple ici) et dans le dossier `notebooks`, il y a l'ensemble de nos notebooks. Ainsi, lorsque nous travaillons sur notre notebook, nous nous trouvons

dans le dossier `notebooks` par défaut. Donc, pour que Pandas puisse trouver notre fichier, on utilise les symboles `../` pour dire à Pandas de retourner dans le dossier au-dessus du dossier `notebooks`, puis d'entrer dans le dossier `datasets` et de lire le fichier 120-years-of_olympic-history-athletes-and-results.csv.

Vous pouvez créer la même structure, c'est-à-dire un dossier `datasets` où vous mettez le fichier CSV et un dossier `notebooks` où vous copiez-collez le notebook fourni en téléchargement de ce livre. Vous ouvrez ce notebook dans Jupyter Notebook et les commandes fournies dans ce chapitre fonctionneront correctement. Si vous ne souhaitez pas garder la même structure, n'oubliez pas de changer le chemin vers le fichier afin que l'importation de celui-ci fonctionne correctement.

Enfin, la méthode `head()` de Pandas, qui s'applique à une série ou à dataframe, permet d'avoir un aperçu des cinq premières lignes de notre objet afin de voir à quoi il ressemble. Vous utiliserez souvent cette méthode, car en analyse de données, nous travaillons avec de grands tableaux de données et il n'est pas conseillé ni utile de les afficher en entier.

Résultat de la commande précédente :

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	Sport	Event	Medal
0	1	A Dijiang	M	24.0	180.0	80.0	China	CHN	1992 Summer	1992	Summer	Barcelona	Basketball	Basketball Men's Basketball	NaN
1	2	A Lamusi	M	23.0	170.0	60.0	China	CHN	2012 Summer	2012	Summer	London	Judo	Judo Men's Extra-Lightweight	NaN
2	3	Gunnar Nielsen Aaby	M	24.0	NaN	NaN	Denmark	DEN	1920 Summer	1920	Summer	Antwerpen	Football	Football Men's Football	NaN
3	4	Edgar Lindenau Aabye	M	34.0	NaN	NaN	Denmark/Sweden	DEN	1900 Summer	1900	Summer	Paris	Tug-Of-War	Tug-Of-War Men's Tug-Of-War	Gold
4	5	Christine Jacoba Aaftink	F	21.0	185.0	82.0	Netherlands	NED	1988 Winter	1988	Winter	Calgary	Speed Skating	Speed Skating Women's 500 metres	NaN

On constate que notre fichier a bien été importé dans Python grâce à Pandas et qu'il ressemble à un joli tableau qu'on va pouvoir explorer ensemble : ici, il s'agit d'un dataframe, un tableau à deux dimensions, avec des lignes et des colonnes.

À présent, vous savez comment lire un fichier texte grâce aux fonctions `read_table()` et `read_csv()`. Toutefois, il y a quelques options intéressantes à connaître pour ces deux fonctions.

b. Gestion de l'en-tête

La première option importante est la notion de *header*, que l'on peut traduire par "en-tête" en français. Il s'agit plus simplement des noms de colonnes. Par défaut, Python prendra la première ligne du fichier comme en-tête du tableau, ce qui fonctionne bien dans le cas de notre jeu de données, où la première ligne correspond effectivement aux noms des variables.

Toutefois, si votre tableau ne contient pas de *header*, et que vous ne voulez pas qu'il y en ait, vous pouvez le spécifier avec la syntaxe suivante :

```
import pandas as pd
pd.read_csv("mon_fichier.csv", header=None)
```

Ici, vos noms de colonnes correspondront à des valeurs partant de 0 et qui s'incrémentent de 1 en 1 (0, 1, 2, 3, etc.).

Si vous testez cette commande avec notre fichier, cela générera une erreur car ce fichier contient bien un header. Le fait de mettre l'option `header` à "`None`" fait que Pandas met la première ligne du fichier en tant que ligne du tableau, et non plus en tant qu'en-tête. Cela ne peut pas marcher, car pour les colonnes numériques comme l'âge par exemple, celle-ci se retrouvera avec une valeur de type caractère : nous le verrons plus tard, au sein d'une colonne d'un dataframe, il ne doit y avoir qu'un seul type de données.

C'est là que l'option `skiprows` devient intéressante, car elle permet d'ignorer certaines lignes lors de la lecture du fichier et elle ne les stocke pas dans le tableau. Par exemple, si on veut supprimer l'en-tête du fichier lors de la lecture, on donnera la valeur "`1`" à l'option `skiprows` pour ignorer la première ligne du fichier. Voici la syntaxe :

```
import pandas as pd
pd.read_csv("mon_fichier.csv", skiprows=1, header=None)
```

Si votre fichier ne possède pas d'en-tête, mais que vous voulez en créer un lors de la lecture du fichier, vous pouvez le faire avec l'option `names`. La syntaxe est la suivante :

```
import pandas as pd
pd.read_csv("mon_fichier.csv", names=['colonne_1','colonne_2'])
```

Ici, vous n'avez pas besoin de spécifier que votre fichier ne contient pas de header avec l'option `header=None`, car comme vous passez l'option `names` à la fonction `read_csv()`, Pandas comprend qu'il ne doit pas prendre le *header* dans votre fichier, mais qu'il doit le définir avec la liste que vous lui donnez dans l'option `names`.

Passons à l'exemple pratique pour illustrer ces trois nouvelles options. Imaginons que nous souhaitions supprimer l'en-tête en anglais de notre fichier et que nous voulions en mettre un nouveau, en français.

Voici le code permettant de faire cela :

```
import pandas as pd
donnees_francais=pd.read_csv("../datasets/120-years-of-olympic-
history-athletes-and-results.csv",skiprows=1, names=["Id","Nom",
"Sexe", "Age", "Taille", "Poids", "Equipe", "CNO", "Jeux",
"Annee", "Saison", "Ville", "Sport", "Epreuve", "Medaille"])
donnees_francais.head()
```

Résultat

	Id	Nom	Sexe	Age	Taille	Poids	Equipe	CNO	Jeux	Annee	Saison	Ville	Sport	Epreuve	Medaille
0	1	A Dijiang	M	24.0	180.0	80.0	China	CHN	1992 Summer	1992	Summer	Barcelona	Basketball	Basketball Men's Basketball	NaN
1	2	A Lamusi	M	23.0	170.0	60.0	China	CHN	2012 Summer	2012	Summer	London	Judo	Judo Men's Extra-Lightweight	NaN
2	3	Gunnar Nielsen Aaby	M	24.0	NaN	NaN	Denmark	DEN	1920 Summer	1920	Summer	Antwerpen	Football	Football Men's Football	NaN
3	4	Edgar Lindenau Aabye	M	34.0	NaN	NaN	Denmark/Sweden	DEN	1900 Summer	1900	Summer	Paris	Tug-Of-War	Tug-Of-War Men's Tug-Of-War	Gold
4	5	Christine Jacoba Aaftink	F	21.0	185.0	82.0	Netherlands	NED	1988 Winter	1988	Winter	Calgary	Speed Skating	Speed Skating Women's 500 metres	NaN

Ici, nous avons ignoré la première ligne, c'est-à-dire l'en-tête du fichier, avec l'option `skiprows=1`. Puis nous avons défini un nouvel en-tête grâce à l'option `names`, en lui donnant une liste Python contenant le même nombre d'éléments que le nombre de colonnes dans le tableau. Nous obtenons ainsi un tableau avec un en-tête en français.

c. Gestion des index

Nous le verrons à partir de la section suivante, Pandas propose deux structures de données principales, une correspondant à un tableau à une dimension (une colonne) et une correspondant à un tableau à deux dimensions (avec des lignes et des colonnes). Tout

comme avec NumPy, ces structures de données possèdent des index, permettant de donner un nom aux colonnes (et aux lignes dans le cas d'un tableau à deux dimensions). Ces index permettent aussi d'accéder aux différentes données que les structures contiennent.

Il y a différentes manières de gérer les index lors de la lecture d'un fichier. Celui-ci peut être implicite si on a une entrée en moins dans le *header* par rapport au nombre de colonnes, c'est-à-dire si le nom de la première colonne est vide. Dans ce cas, la fonction `read_csv()` (ou `read_table()`) prend par défaut la première colonne comme étant l'index.

Il est aussi possible de spécifier quelle colonne du jeu de données correspond à l'index, grâce à l'option `index_col`.

Syntaxe d'utilisation de l'option `index_col`

```
import pandas as pd
pd.read_csv("mon_fichier.csv", index_col=0)
```

Ici, Pandas définit la première colonne du fichier comme étant l'index du tableau.

Attention, n'oubliez pas qu'en Python, on compte à partir de 0, 0 correspondant à la première valeur.

Enfin, il est possible de définir plusieurs colonnes comme étant les index, toujours grâce à l'option `index_col`.

Syntaxe

```
import pandas as pd
pd.read_csv("mon_fichier.csv", index_col=[0,1])
```

Ici, la première et la deuxième colonne deviennent les index du tableau après lecture du

fichier.

Passons à l'exemple pratique pour illustrer les index. Lorsqu'on regarde notre tableau `donnees`, on constate qu'il y a une colonne `ID` qui correspond à l'identifiant de chaque athlète, on pourrait vouloir le définir comme index.

Pour cela, on peut utiliser le code suivant :

```
import pandas as pd
donnees_1index=pd.read_csv("../datasets/120-years-of-olympic-history-athletes-and-results.csv", index_col=[0])
donnees_1index.head()
```

Résultat

ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	Sport	Event	Medal
1	A Dijiang	M	24.0	180.0	80.0	China	CHN	1992 Summer	1992	Summer	Barcelona	Basketball	Basketball Men's Basketball	NaN
2	A Lamusi	M	23.0	170.0	60.0	China	CHN	2012 Summer	2012	Summer	London	Judo	Judo Men's Extra-Lightweight	NaN
3	Gunnar Nielsen Aaby	M	24.0	NaN	NaN	Denmark	DEN	1920 Summer	1920	Summer	Antwerpen	Football	Football Men's Football	NaN
4	Edgar Lindenau Aabye	M	34.0	NaN	NaN	Denmark/Sweden	DEN	1900 Summer	1900	Summer	Paris	Tug-Of-War	Tug-Of-War Men's Tug-Of-War	Gold
5	Christine Jacoba Aaftink	F	21.0	185.0	82.0	Netherlands	NED	1988 Winter	1988	Winter	Calgary	Speed Skating	Speed Skating Women's 500 metres	NaN

Ici, on constate que la colonne `ID` du fichier est devenue l'index du tableau. Avec Pandas et ses structures, il est possible d'avoir des index qui ne sont pas uniques. Ainsi, si un athlète est présent plusieurs fois dans le fichier, son ID sera lui aussi présent plusieurs fois dans l'index. Et cela permet de faire des traitements particuliers avec Pandas sur les index, ce que nous verrons plus tard. Nous verrons aussi comment utiliser les multi-index et leur utilité.

d. Création d'un tableau à une dimension à partir du fichier

Comme nous le disions, il existe deux structures de données que nous verrons juste après, correspondant à un tableau à une dimension (`Series`) ou un tableau à deux dimensions (`DataFrame`). Les fonctions de lecture de fichiers de Pandas créeront par défaut un dataframe, même à partir d'un fichier ne contenant qu'une seule colonne, et qui pourrait donc être stockée comme une série. Si on souhaite créer un tableau à une dimension

plutôt que deux, il faudra spécifier l'option `squeeze`.

Syntaxe

```
import pandas as pd
pd.read_csv("fichier.csv", squeeze=True)
```

e. Filtrage des colonnes lors de la lecture du fichier

Il est possible, dès la lecture du fichier, de ne lire que certaines colonnes du tableau si toutes ne nous intéressent pas, grâce à l'option `usecols`.

Syntaxe

```
import pandas as pd
pd.read_csv("mon_fichier.csv", usecols=[3,6])
```

Ici, on demande de n'importer que les colonnes 4 et 7 du fichier.

Il est aussi possible de sélectionner des colonnes en donnant une liste de noms de colonnes plutôt qu'une liste de positions de colonnes.

```
import pandas as pd
pd.read_csv("mon_fichier.csv",usecols=["macolonne_1",macolonne_2"])
```

Passons à l'exemple pratique. Si on ne souhaite sélectionner que les colonnes contenant les noms des athlètes, leur poids et leur taille, le code sera le suivant.

Code

```
import pandas as pd
donnees_usecols_position=pd.read_csv("../datasets/120-years-of-
olympic-history-athletes-and-results.csv", usecols=[1,4,5])
donnees_usecols_position.head()
```

Dans le code précédent, nous sélectionnons les colonnes grâce à leurs positions. Dans le code suivant, sélectionnons les colonnes par leurs noms plutôt que leurs positions.

Code

```
import pandas as pd
donnees_usecols =pd.read_csv("../datasets/120-years-of-olympic-
history-athletes-and-results.csv",
usecols=["Name","Height","Weight"])
donnees_usecols.head()
```

Dans les deux cas, les premières lignes de la sortie sont les suivantes.

Résultat

	Name	Height	Weight
0	A Dijiang	180.0	80.0
1	A Lamusi	170.0	60.0
2	Gunnar Nielsen Aaby	NaN	NaN
3	Edgar Lindenau Aabye	NaN	NaN
4	Christine Jacoba Aaftink	185.0	82.0

Nous avons bien sélectionné les trois colonnes qui nous intéressent directement lors de la lecture du fichier.

f. Les types des différentes colonnes

Il est possible de spécifier le type de chaque colonne afin que Pandas prenne en compte les différents traitements possibles qu'il peut faire sur chaque colonne. En effet, on ne peut

pas faire les mêmes traitements sur une colonne de type numérique que sur une colonne de type caractère. Par exemple, on peut multiplier une colonne de type numérique, mais pas une colonne de type caractère.

La syntaxe générale pour spécifier le type des colonnes est la suivante :

```
import pandas as pd
pd.read_csv("mon_fichier.csv",dtype={"macolonne_1":"Int64",
"macolonne_2":"float64", "macolonne_3":"object"})
```

Ici, il suffit de donner un dictionnaire avec en clé les noms de colonnes et en valeurs le `dtype` de cette colonne.

g. Gestion des dates lors de la lecture du fichier

Enfin, certaines colonnes peuvent contenir des dates et il est important de les *parser* correctement. *Parser* pourrait être traduit par "analyser la syntaxe d'un texte". Pandas va analyser le format de la date et la formater correctement.

Lors de la lecture du fichier, on peut spécifier quelles colonnes traiter comme des dates. Par exemple, si la colonne 5 de notre fichier contient des dates, on le dit lors de la lecture.

Syntaxe

```
import pandas as pd
pd.read_csv("mon_fichier.csv",parse_dates=[4])
```

Lorsque nous visualiserons notre tableau, nous pourrons constater que le `dtype` de notre colonne est `datetime64`.

Passons à l'exemple pratique. On sait que la colonne 9 de notre dataframe correspond à l'année des Jeux olympiques auxquels ont participé les athlètes. On pourrait vouloir dire à Pandas de considérer cette colonne comme une date.

Pour l'instant, si on regarde le type de la colonne 9, on a le résultat suivant.

Code

```
donnees.dtypes
```

Résultat

```
Year      int64
```

Ici, notre colonne représentant l'année des JO est considérée comme un type entier par Pandas. En précisant que la colonne 9 doit être considérée comme une date, on obtient le résultat suivant.

Code

```
import pandas as pd
donnees_dates=pd.read_csv("../datasets/120-years-of-olympic-history-athletes-and-results.csv", parse_dates=[9])
donnees_dates.dtypes
```

Résultat

```
Year      datetime64[ns]
```

La colonne 9 est bien de type `datetime64` maintenant.

2. Lecture de fichiers Excel

Pour lire un fichier Excel, c'est-à-dire un fichier au format XLSX ou XLS, il suffit d'utiliser la fonction `read_excel()` de Pandas.

Syntaxe

```
import pandas as pd
fichier_excel=pd.read_excel('mon_fichier.xlsx')
```

Par défaut, la fonction `read_excel()` lit la première feuille Excel du fichier qu'on lui donne. Si on souhaite importer une autre feuille, il faut donner le nom de la feuille à l'option `sheet_name`, qui signifie 'nom de la feuille', en français.

Syntaxe

```
import pandas as pd
fichier_excel=pd.read_excel('mon_fichier.xlsx',
sheet_name="Nom_de_la_feuille")
```

À chaque fois, la structure de données créée sera un dataframe, contenant les données de la feuille Excel qui nous intéresse. Les autres options de cette fonction sont les mêmes que celles vues précédemment avec les fonctions `read_table()` et `read_csv()`, comme les options `header`, `names`, `index_col`, `usecols`, `squeeze`, ou encore `dtype`.

Passons à l'exemple pratique. En téléchargement avec ce livre, vous pouvez récupérer le fichier "120-years-of-olympic-history-athletes-and-results.xlsx", si vous souhaitez effectuer vous-mêmes ces commandes. Ce fichier contient une première feuille avec les données de JO et une deuxième feuille factice, pour vous montrer comment lire un fichier contenant plusieurs feuilles Excel.

Pour lire la première feuille du fichier Excel, le code est le suivant :

```
import pandas as pd
donnees_excel=pd.read_excel('../datasets/120-years-of-olympic-
history-athletes-and-results.xlsx')
donnees_excel.head()
```

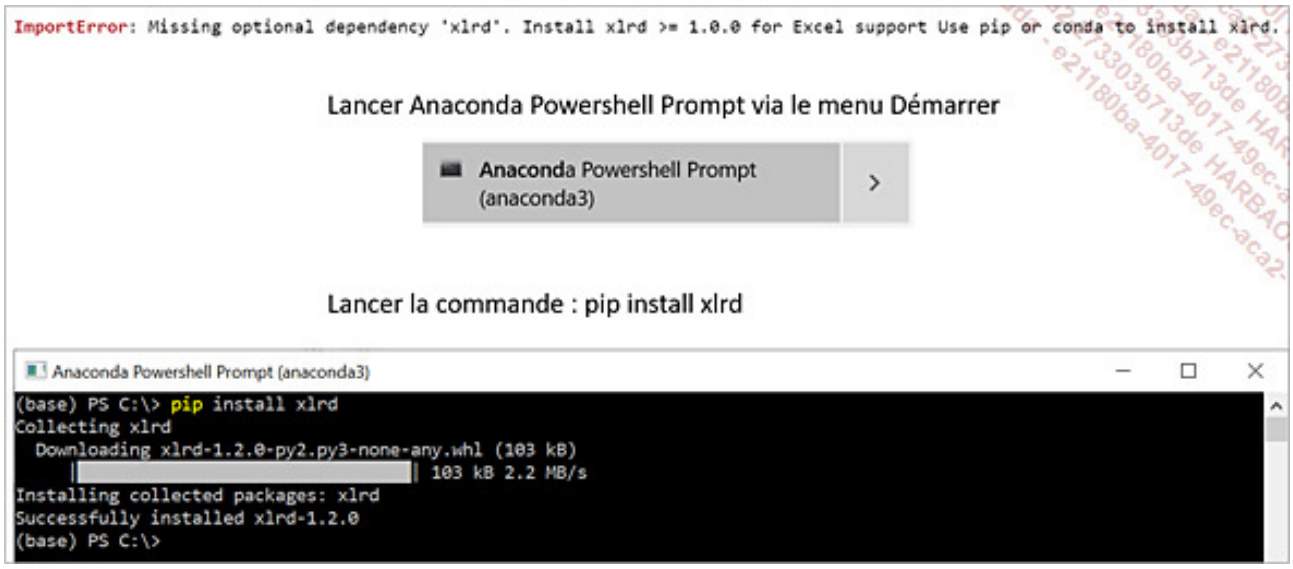
Ces commandes permettent de lire la première feuille du fichier XLS, par défaut, de la stocker dans `donnees_excel` et d'en afficher les cinq premières lignes.

Résultat

ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	Sport	Event	Medal
1	A Dijiang	M	24.0	180.0	80.0	China	CHN	1992 Summer	1992	Summer	Barcelona	Basketball	Basketball Men's Basketball	NaN
2	A Lamusi	M	23.0	170.0	60.0	China	CHN	2012 Summer	2012	Summer	London	Judo	Judo Men's Extra-Lightweight	NaN
3	Gunnar Nielsen Aaby	M	24.0	NaN	NaN	Denmark	DEN	1920 Summer	1920	Summer	Antwerpen	Football	Football Men's Football	NaN

Attention, si vous rencontrez une erreur lors de cette commande, dont la dernière ligne est : `ImportError : Missing optional dependency  xlrd` ..., veuillez procéder de la manière suivante :

Lancez le logiciel **Anaconda Powershell Prompt** puis tapez la commande `pip install xlrd` pour installer le package xlrd, qui réglera ce problème.



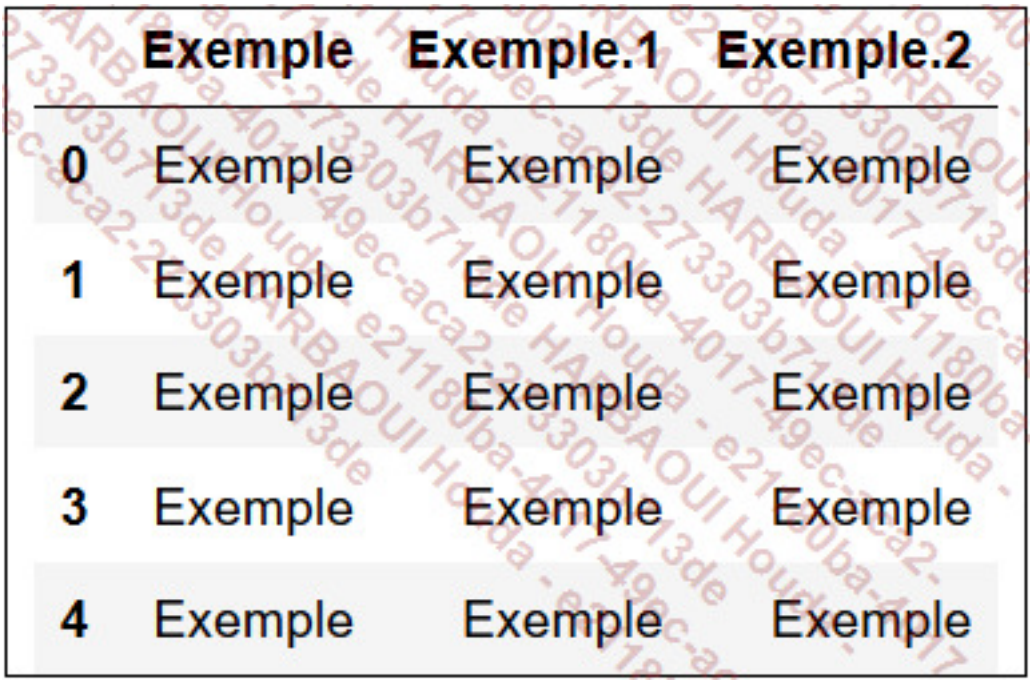
Si on souhaite lire la deuxième feuille de ce fichier, on utilise l'option `sheet_name` avec le nom de la feuille qui nous intéresse.

Code

```
import pandas as pd
donnees_excel=pd.read_excel('../datasets/120-years-of-olympic-
history-athletes-and-results.xlsx', sheet_name='Feuille2')
donnees_excel.head()
```

Ce code stockera les données de la deuxième feuille Excel dans le dataframe : `donnees_excel`.

Résultat



	Exemple	Exemple.1	Exemple.2
0	Exemple	Exemple	Exemple
1	Exemple	Exemple	Exemple
2	Exemple	Exemple	Exemple
3	Exemple	Exemple	Exemple
4	Exemple	Exemple	Exemple

3. Importation des données à partir d'une base de données

Si vous souhaitez importer des données à partir d'une base de données plutôt que d'un fichier plat comme les formats vus précédemment, c'est possible.

Par exemple, pour une base de données SQLite, la syntaxe générale sera la suivante :

```
import pandas as pd
import sqlite3
connexion = sqlite3.connect('base_de_donnees.db')
requete = "SELECT * FROM TABLES;"
resultats = pd.read_sql(requete, con=connexion)
resultats.head()
```

Ici, on importe la librairie `Pandas` ainsi que la librairie `sqlite3`. On crée la connexion à notre fichier contenant la base de données au format `.db` grâce à la fonction

`connect()` de sqlite3. Puis on écrit la requête qui nous permettra de récupérer les données qui nous intéressent dans la base de données qu'on range dans la variable `requete`. Enfin, on lance la requête sur la base de données grâce à la fonction `read_sql()` de Pandas, à laquelle on donne la requête, ainsi que la connexion créée précédemment. De cette manière, on récupère les résultats de la requête sous forme d'un dataframe.

Exemple pratique

Veuillez récupérer le fichier chinook.db disponible en téléchargement avec ce livre, si vous souhaitez effectuer vous-même ces commandes. Il s'agit d'une base de données test SQLite, disponible à l'adresse suivante : http://www.sqlitetutorial.net/sqlite-sample-database/

Nous ne rentrerons pas dans le détail de cette base de données d'exemple, car le but ici est simplement de vous expliquer comment vous connecter à une base de données et y récupérer les informations qui vous intéressent. Une fois ces données récupérées et stockées dans un dataframe, les manipulations que vous pourrez effectuer dessus seront les mêmes que celles que nous verrons par la suite avec le fichier "120-years-of-olympic-history-athletes-and-results.csv".

Pour l'exemple, sachez que cette base de données contient onze tables, dont une table contenant les informations au sujet des employés d'une entreprise. Imaginons que nous ayons envie de récupérer les informations de l'ensemble des employés de la base de données afin de les stocker dans un dataframe pour les analyser, le code serait le suivant.

Code

```
import pandas as pd
import sqlite3
connexion = sqlite3.connect('../datasets/chinook.db')
requete = "SELECT * FROM employees;"
```

```
resultats = pd.read_sql(requete, con=connexion)
resultats.head()
```

Ici, on se connecte à la base de données stockée dans le fichier `chinook.db`, lui-même stocké dans le dossier `datasets`. Puis on crée une requête pour sélectionner l'ensemble des données de la table nommée `employees`. Enfin, on lance cette requête sur la base de données et on stocke les données extraites dans un dataframe nommé `resultats`.

Résultat

	EmployeeId	LastName	FirstName	Title	ReportsTo	BirthDate	HireDate	Address	City	State	Country	PostalCode	Phone	Fax
0	1	Adams	Andrew	General Manager	NaN	1962-02-18 00:00:00	2002-08-14 00:00:00	11120 Jasper Ave NW	Edmonton	AB	Canada	T5K 2N1	+1 (780) 428-9482	+1 (780) 428-3457
1	2	Edwards	Nancy	Sales Manager	1.0	1958-12-08 00:00:00	2002-05-01 00:00:00	825 8 Ave SW	Calgary	AB	Canada	T2P 2T3	+1 (403) 262-3443	+1 (403) 262-3322
2	3	Peacock	Jane	Sales Support Agent	2.0	1973-08-29 00:00:00	2002-04-01 00:00:00	1111 6 Ave SW	Calgary	AB	Canada	T2P 5M5	+1 (403) 262-3443	+1 (403) 262-6712
3	4	Park	Margaret	Sales Support Agent	2.0	1947-09-19 00:00:00	2003-05-03 00:00:00	683 10 Street SW	Calgary	AB	Canada	T2P 5G3	+1 (403) 263-4423	+1 (403) 263-4289
4	5	Johnson	Steve	Sales Support Agent	2.0	1965-03-03 00:00:00	2003-10-17 00:00:00	7727B 41 Ave	Calgary	AB	Canada	T3B 1Y7	1 (780) 836-9987	1 (780) 836-9543

On a bien importé des données à partir d'une base de données SQLite.

4. Lecture de fichiers au format JSON

Le dernier format que nous verrons pour l'importation de données est le format JSON. Ce format est devenu très populaire ces dernières années et vous risquez de le rencontrer. Il est très utilisé pour échanger des données de manière structurée et légère. Ainsi, voyons à présent ensemble comment lire un fichier JSON. Pour cela, nous avons récupéré la table `employees` de la base de données précédente, que nous avons stockée au format JSON, pour l'exemple.

Le fichier employees.json est disponible en téléchargement avec ce livre.

Le format JSON est un format texte, tout comme le TXT, mais ses champs sont organisés d'une manière différente. Voici un aperçu du fichier :

```
[
 {
 "EmployeeId":            1,
 "LastName":           "Adams",
 "FirstName":           "Andrew",
 "Title":          "General Manager",
 "ReportsTo":            null,
 "BirthDate":           "1962-02-18 00:00:00",
 "HireDate":           "2002-08-14 00:00:00",
 "Address":          "11120 Jasper Ave NW",
 "City":          "Edmonton",
 "State":          "AB",
 "Country":           "Canada",
 "PostalCode":            "T5K 2N1",
 "Phone":          "+1 (780) 428-9482",
 "Fax":          "+1 (780) 428-3457",
 "Email":           "andrew@chinookcorp.com"
},
{
 "EmployeeId":            2,
 "LastName":           "Edwards",
 "FirstName":           "Nancy",
 "Title":          "Sales Manager",
 "ReportsTo":            1,
 "BirthDate":           "1958-12-08 00:00:00",
 "HireDate":           "2002-05-01 00:00:00",
 "Address":          "825 8 Ave SW",
 "City":          "Calgary",
 "State":          "AB",
 "Country":           "Canada",
 "PostalCode":            "T2P 2T3",
 "Phone":          "+1 (403) 262-3443",
 "Fax":          "+1 (403) 262-3322",
 "Email":           "nancy@chinookcorp.com"
},
```

Pour expliquer brièvement ce format, l'ensemble des données sont stockées entre crochets `[]`. Puis, chaque individu ou observation est stocké entre accolades `{}`, ces accolades pouvant être comparées aux lignes d'un tableau classique. Enfin, chaque accolade contient l'ensemble des informations (des colonnes) de l'individu/observation sous forme de clés-valeurs séparées par des virgules. Chaque couple clé-valeur représente une colonne de la ligne.

La librairie Pandas propose une fonction appelée `read_json()` qui permet de lire un fichier au format JSON.

Syntaxe

```
import pandas as pd
pd.read_json("mon_fichier.json")
```

Appliquons cette fonction à un exemple concret, en important le contenu du fichier `employees.json`.

Code

```
import pandas as pd
pd.read_json("../datasets/employees.json")
```

On obtient exactement le même résultat que lors de la requête sur la base de données stockée dans le fichier chinook.db.

5. Écriture de fichiers ou exportation de données

Lorsque vos analyses sont terminées, il est possible que vous souhaitiez exporter votre dataframe de résultats afin de le stocker dans un fichier CSV, TXT ou encore XLSX. Les méthodes à utiliser se ressemblent beaucoup.

Pour exporter un dataframe dans un fichier CSV, la syntaxe est la suivante :

```
mon_dataframe.to_csv("mes_resultats.csv")
```

Ici, on utilise la méthode `to_csv()` sur l'objet `mon_dataframe` qui est notre dataframe de résultat, et on l'écrit dans le fichier mes_resultats.csv.

Diverses options existent pour cette méthode, dont les principales sont les suivantes :

- `sep` : si on veut spécifier le séparateur ; par défaut, ce sera la virgule, étant donné qu'on utilise la méthode `to_csv()`.

Syntaxe

```
mon_dataframe.to_csv("mes_resultats.csv", sep=",")
```

- `columns` : une liste avec les colonnes qu'on souhaite écrire dans le fichier, si on ne les veut pas toutes.

Syntaxe

```
mon_dataframe.to_csv("mes_resultats.csv", columns=["ma_colonne1", "ma_colonne3"])
```

- `header` : si on souhaite écrire l'en-tête du dataframe dans le fichier ; par défaut, ce sera le cas.
- `Index` : par défaut, cette option est à `True` (vrai), ce qui spécifie à Pandas d'écrire les labels de lignes (les index de lignes) dans le fichier. À `False`, il ne le fait pas.

Pour écrire le dataframe dans un fichier au format TXT, on pourra là aussi utiliser la méthode `to_csv()`, mais en précisant le séparateur comme étant la tabulation.

Syntaxe

```
mon_dataframe.to_csv("mes_resultats.txt", sep="\t")
```

Enfin, pour écrire votre dataframe dans un fichier Excel, la syntaxe est la suivante :

```
mon_dataframe.to_excel("mes_resultats.xlsx")
```

On peut utiliser l'option `sheet_name` pour définir le nom de la feuille qui contiendra le dataframe.

Syntaxe

```
mon_dataframe.to_excel("mes_resultats.xlsx",
sheet_name="mes_resultats")
```

Les autres options de la méthode `to_excel()` sont assez similaires à celles de la méthode `to_csv()`. N'hésitez pas à consulter la documentation de la méthode pour prendre connaissance de toutes les options possibles.