Traitement des données en table

M. Tellene

1 Création de l'environnement de travail

Avant de commencer le TP, vous allez créer votre environnement de travail. Pour ce faire vous allez aller dans votre dossier personnel et créer un dossier « **TP_traitement_donnees_table** ».

Une fois le dossier créé, aller à l'adresse suivante : https://f24.link/fBIJG [1] ou sur Pronote, et télécharger le dossier « **ressources** » dans le dossier précédemment créé.

2 Données en table

Le stockage de l'information est actuellement un enjeu majeur : administration, big data ou bien même les clouds Internet. Pour simplifier la lecture et l'utilisation de ces données, celles-ci peuvent être stockées sous forme de tableau, on parle alors de **données tabulaires** ou plus simplement de **données en table**.

Exemple de données en table :

Nom	Prénom	Date de naissance
Martin	Jade	23/07/2002
Bernard	Léo	13/03/1970
Thomas	Henry	03/03/1972

3 CSV

Le format CSV est, au même titre que texte ou PDF, un format de fichier. Le format CSV permet de stocker des données sous forme de tableau.
Voici l'exemple précédent au format CSV :

```
nom,prenom,date_de_naissance
Martin,Jade,23_07_2002
Bernard,Léo,13_03_1970
Thomas,Henry,03_03_1972
```

Ici, vous pouvez voir que :

- *nom*, *prenom* et *date_de_naissance* sont les colonnes du tableau précédent, elles sont appelés **descripteurs**
- `Martin`, `Bernard` et `Thomas` sont des **valeurs du descripteur** *nom*
- la virgule sépare les éléments, il est donc appelé **séparateur**. La virgule est une norme pour les données anglo-saxonnes. Le problème est qu'en français la virgule est utilisée pour les chiffres décimaux alors qu'en anglais on utilise un point (cf le type `float` en Python). Il serait donc impossible de différencier une virgule d'un chiffre décimal, d'une virgule de séparation. C'est pour cela que l'on utilise **le séparateur point-virgule** (;).

1. Si ça ne marche pas : https://mega.nz/folder/4TE2jTxC#sGoxGh4hDkj6oSa6yoiS1A

4 Premières manipulations de fichiers CSV

Avec un ami vous avez créé un jeu vidéo. Lors d'une partie, un joueur peut, s'il le veut, sauvegarder son meilleur score, cette action ajoutera le score et le pseudo du joueur à un fichier CSV. Quelques joueurs ont déjà entré leur meilleur score dans le fichier se trouvant dans le dossier « ressources » sous le nom « highscore.csv ». Afin de ne pas à avoir à gérer le fichier manuellement, vous décidez de créer un programme Python permettant de gérer le fichier contenant les meilleurs scores.

Il est important de comprendre qu'un programme informatique peut effectuer des actions sur des éléments extérieurs à lui-même comme des fichiers (textes, CSV, images,...), d'autres programmes et bien d'autres choses.

Pour cette partie, il faudra créer un fichier **manipulation_1.py**. Pour réaliser toutes les fonctions demandées, vous allez utiliser la librairie `pandas`. Cette librairie permet de traiter les données d'un fichier CSV. **Penser donc à importer cette librairie.**

4.1 Lecture d'un fichier CSV

Une première action possible sur les fichiers est la lecture. Il peut être intéressant d'automatiser la lecture d'un fichier notamment si celui-ci contient beaucoup de données. Dans cette première partie, nous allons nous pencher sur la lecture de fichier CSV.

EXERCICE 1
Écrire une fonction `lecture_fichier` qui ne prend pas d'argument. Cette fonction devra afficher le contenu du fichier « highscore.csv ». Pour ce faire, vous utiliserez la fonction `read_csv` de la librairie `pandas`. Vous trouverez ici [2] un exemple d'utilisation de cette fonction.

Une fois que vous y êtes arrivé répondre aux questions suivantes :

1. Quels sont les noms des deux colonnes du fichier ? .....
2. Combien de ligne contient le fichier ? .....
3. Comment s'appelle le premier joueur du fichier ? .....
4. Quel est le meilleur score obtenu ? Quel joueur possède ce score ? .....

Il est également possible de récupérer seulement une ou plusieurs colonnes du fichier. Cela peut être très utile quand nous voulons lire des colonnes précises d'un fichier en contenant beaucoup.

EXERCICE 2
Écrire une fonction `lecture_colonne` qui prend en argument un tableau de noms de colonnes (ce tableau peut ne contenir aucun nom, une nom ou bien une multitude de noms). La fonction devra afficher les colonnes du fichier entrées en argument. Pour rappel, un exemple est disponible ici [3].

2. https://towardsdatascience.com/how-to-read-csv-file-using-pandas-ab1f5e7e7b58
3. https://towardsdatascience.com/how-to-read-csv-file-using-pandas-ab1f5e7e7b58

Exemple de rendu :

```
>>> lecture_colonne(["joueur"])
            joueur
0        bactigerm
1          valance
2  dad_of_the_dead
3         belizard
4         anarkiss
5        abominate
6           xxbgxx
```

Le problème de la méthode précédente est qu'elle ne gère pas les erreurs de noms, notamment si on met, dans le tableau de nom, un nom de colonne n'existant pas, il y aura une erreur. Nous allons rectifier ça.

EXERCICE 3
Copier la fonction `lecture_colonne_v2` qui se trouve dans le fichier « fonctions.py » du dossier « ressources ». Remplacer les « ... » par le code approprié pour que cette fonction affiche les colonnes demandées même s'il y a des colonnes n'existant pas dans le fichier.

Enfin, il est également possible de définir le nombre de lignes à afficher.

EXERCICE 4
Copier la fonction `lecture_n_premieres_lignes` qui se trouve dans le fichier « fonctions.py » du dossier « ressources ». Remplacer les « ... » par le code approprié pour que cette fonction affiche les n lignes où `n` est l'argument de la fonction.

Exemple de rendu :

```
>>> lecture_n_premieres_lignes(5)
            joueur  score
0        bactigerm     31
1          valance     56
2  dad_of_the_dead     65
3         belizard     88
4         anarkiss     25
```

4.2 Écriture dans un fichier CSV

Une autre action possible sur les fichiers CSV est l'écriture. Comme pour la lecture, il peut être intéressant d'automatiser l'écriture dans un fichier. La librairie `pandas` offre un moyen d'écrire dans un fichier CSV. Voici un exemple fonctionnel pour notre fichier de données :

```
data = pandas.DataFrame([["falgon", "64"]], columns=["joueur", "score"])
data.to_csv("ressources/highscore.csv", mode="a", header=False, index=False)
```

Il est également possible de rajouter plusieurs en même temps, voici un exemple :

```
data = pandas.DataFrame([["pgm123", "72"], ["sayo", "99"]], columns=["joueur",
    "score"])
data.to_csv("ressources/highscore.csv", mode="a", header=False, index=False)
```

Répondre aux questions suivantes (pour vous aider, vous pouvez vous servir de ce site [4]) :

1. Quel est le type des données ajoutées dans le fichier ? .....
2. Quelle est l'utilité de `mode="a"` ? Que se passerait-il si nous mettions `mode="w"` à la place ?

   .....
3. Quelle est l'utilité de `header=False` ? .....
4. Quelle est l'utilité de `index=False` ? .....

EXERCICE 5
Écrire une fonction `ecriture_donnees` qui prend un argument un tableau de tableau (permettant d'ajouter les données) et un tableau (permettant de spécifier les colonnes). Cette fonction devra ajouter les données dans les colonnes spécifiées.

Enfin cette fonction appelle `lecture_fichier()` afin d'afficher le fichier. Cette action servira à prouver que les données ont bien été ajoutées au fichier. **La fonction à écrire peut rajouter une ligne comme plusieurs, faîtes attention à la spécification du premier argument.**

4.3 Autres actions sur les fichiers CSV

La lecture et l'écriture sont les deux actions principales sur des fichiers, mais il existe d'autres actions très utiles dans certains cas. Dans notre exemple, nous gérons un fichier de meilleurs scores, mais vous pouvez remarquer que ce fichier n'est pas trié. En effet, il serait plus judicieux de mettre en premier le meilleur score, puis le deuxième meilleur...

EXERCICE 6
Écrire une fonction `tri_donnees` qui ne prend pas d'argument. Cette fonction devra :

1. ouvrir le fichier comme si vous alliez l'afficher (`read_csv`)
2. trier le fichier par score et mettre le résultat dans une variable. Vous trouverez une aide pour trier le fichier ici [5]
3. écrire le contenu de la variable dans un nouveau fichier

   Aide : `variable.to_csv(nouveau_fichier.csv, index=False)`
4. afficher le nouveau fichier pour montrer que les données ont été triées

Une autre action très utile est la recherche dans une table. Cette action consiste simplement à chercher une valeur particulière dans une table afin d'appliquer des actions en sur les données.

EXERCICE 7
Écrire une fonction `recherche_score_avec_joueur` qui prend en argument un nom de joueur. La fonction devra :

1. ouvrir le fichier comme si vous alliez l'afficher

4. https://stackabuse.com/reading-and-writing-csv-files-in-python-with-pandas/#writingcsvfileswithto_csv
5. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sort_values.html

2. récupérer le colonne *joueur* du fichier et transformer cette colonne en tableau. Pour récupérer la colonne *joueur* du fichier, réfléchissez à comment nous pouvons récupérer ou extraire certaines données d'un tableau ou d'une chaîne de caractères. Pour vous aider à transformer une variable en liste, aidez-vous de ce site [6].
3. faire la même chose que l'étape précédente mais pour la colonne *score*
4. récupérer l'indice du nom du joueur dans la liste faîtes à l'étape 2 (pensez au TP précédent sur les listes)
5. afficher l'élément de le tableau trouvé à l'étape 3 qui est à l'indice trouvé à l'étape 4

Exemple de rendu :

```
recherche_score_avec_joueur("bactigerm")
31
```

Faire la même fonction mais avec les scores (rechercher un joueur en entrant un score) n'est pas très intéressant. En effet, un nom de joueur est unique et un joueur a un seul meilleur score, mais il est possible que plusieurs joueurs aient le même score. Nous allons donc voir une dernière action possible sur les fichiers CSV : **les filtres**. Un filtre permet de sélectionner des données seulement si celles-ci répondent à une certaine condition. L'utilisation d'un filtre est donc plus adaptée comparé à l'utilisation de la recherche pour rechercher un utilisateur avec le score.

EXERCICE 8
Écrire une fonction `filtre_score` qui prend en argument un nombre. La fonction devra afficher les lignes dont le *score* **est strictement inférieur** au nombre entré en argument. Pour vous aider, vous pouvez consulter ce post [7].

Exemple de rendu :

```
>>> filtre_score(50)
      joueur  score
0  bactigerm     31
4   anarkiss     25
6     xxbgxx     14
```

5 Pour aller plus loin - Manipulations d'un gros fichier CSV

Après avoir créé les fonctionnalités de votre jeu, vous vous lancez dans un projet d'une autre envergure. Vous êtes tombé sur un fichier contenant toutes les infrastructures sportives de Montréal. Ce très gros fichier est disponible dans le dossier « ressources » sous le nom « infra_sport_Montreal.csv ». Pour cette partie, vous créerez un fichier **montreal.py**, c'est dans ce fichier que vous écrirez toutes les fonctions demandées. Pour réaliser le travail demandé, vous utiliserez encore une fois la librairie `pandas`, pensez à l'importer.

EXERCICE 9
Écrire une fonction `affichage_fichier` qui prend en argument un nom de fichier. La fonction devra afficher le contenu du fichier.

En commentaire du programme, répondre aux questions suivantes :

6. https://www.codegrepper.com/code-examples/python/how+to+convert+a+variable+to+list+in+python
7. https://stackoverflow.com/questions/42313507/searching-for-a-value-in-csv-and-returning-the-row-multiple-times#answer-42313583

1. Comment le fichier s'affiche ?
2. Combien de lignes contient le fichier ? Combien de colonnes contient le fichiers ? Combien de données cela fait-il au total ?

EXERCICE 10
Écrire une fonction `recherche_infra_par_type` qui prend en argument un nom de fichier. Cette fonction devra

1. lister tous les types différents d'infrastructure du fichier. Les types d'infrastructure sont donnés par le délimiteur *TYPE*. Voici une aide [8]
2. demander à l'utilisateur d'entrer un type d'infrastructure
3. vérifier que le type est présent dans le fichier
   — s'il n'est pas présent la fonction devra afficher un message d'erreur et renvoyer `None`).
   — s'il est présent la fonction devra retourner toutes les infrastructures qui ont ce type

EXERCICE 11
Sur le modèle de la fonction précédente, écrire une fonction `recherche_infra_par_lieu` qui prend en argument un nom de fichier. Cette fonction devra à la place d'afficher les infrastructures par type, affichera les infrastructures par endroit (délimiteur *ARROND*). La logique de cette fonction est la même que celle de la fonction précédente.

EXERCICE 12
Écrire une fonction `recherche_infra` qui prend en argument un nom de fichier. Cette fonction devra :

1. appeler la fonction `recherche_infra_par_type` et récupérer le résultat dans une variable
2. écrire le contenu dans un fichier CSV (pensez à mettre `index=False`)
3. appeler la fonction `recherche_infra_par_lieu` sur le fichier créé à l'étape 2 et récupérer le résultat dans une variable
4. écrire le résultat dans le fichier créé à l'étape 2

EXERCICE 13
En vous aidant de ce site [9], écrire une fonction `nombre_elements_par_type` qui prend en argument un nom de fichier. Cette fonction devra faire la somme des lignes ayant le même type. Le résultat sera donc 5 lignes (car il y a 5 types d'infrastructure différents) avec deux éléments par ligne : le premier sera le type de l'infrastructure et le second le nombre d'infrastructures ayant ce type. Vous noterez en commentaire quel est le type d'infrastructure le plus répandu et le type d'infrastructure le moins répandu.

EXERCICE 14
Sur le principe de la fonction suivante, écrire une fonction `nombre_elements_par_ville` qui prend en argument un nom de fichier. Cette fonction devra faire la somme des lignes ayant la même ville (délimiteur *ARROND*). Vous noterez en commentaire la ville qui possède le plus d'infrastructure et la ville ayant le moins d'infrastructure.

8. https://moonbooks.org/Articles/Comment-trouver-toutes-les-valeurs-uniques-dune-colonne-dans-une-dataframe-avec-pandas-en
9. https://dfrieds.com/data-analysis/value-counts-python-pandas.html