Document Data Pipeline - Exercices

Nicolas Klutchnikoff

Mars 2021

Format JSON

Conversion

Nous considérons ici un jeu de données artificiel pour manipuler les fonctions du package `jsonlite` :

```
df1 <- data.frame(x = runif(8), label = sample(c("A", "B", NA), 8, replace = TRUE))
df2 <- data.frame(x = c(3.14, NaN, Inf))
```

1. Convertir `df1` au format JSON avec la fonction `toJSON` et stocker le résultat dans une variable `df1_json`. Quel est le type de cette variable ? Que sont devenus les `NA` ?
2. Convertir `df1_json` en un objet **R** avec `fromJSON`. Le résultat est-il identique à l'objet initial ?
3. Faire la même manipulation avec `df2`. Discuter le résultat obtenu.

Flux d'iris

Nous considérons le jeu de données `iris`. L'objectif de cet exercice est de découvrir les fonctions `stream_in` et `stream_out` du package `jsonlite` qui permettent de gérer des flux de documents au format JSON. Ces fonctions se révéleront particulièrement utiles avec MongoDB.

Répondre aux questions suivantes en utilisant les fonctions du package `jsonlite` :

1. Lire la page d'aide des fonctions `stream_in` et `stream_out`. En particulier, remarquer dans les exemples comment l'argument `con` est utilisé pour travailler avec un fichier.
2. Utiliser la fonction `stream_out` pour afficher les données au format NDJSON. Exporter le résultat dans un fichier `iris.json`.
3. Importer les données de `iris.json` dans un objet **R** avec la fonction `stream_in`. Quelle différence y a-t-il entre cet objet et `iris`? Vous pouvez utiliser des fonctions comme `all.equal` ou `str` pour répondre.
4. Définir la fonction suivante :

   ```
   dummy_handler <- function(df) {
     cat("--- APPEL DE LA FONCTION HANDLER ---\n")
     stream_out(df)
   }
   ```

   Expliquer la sortie de `stream_in(file("iris.json"), handler=dummy_handler)`.
5. Comparer la sortie précédente avec celle de

   ```
   stream_in(file("iris.json"), pagesize=10, handler=dummy_handler)
   ```

   Quel est le rôle de `pagesize` dans la gestion d'un flux?

Star Wars API

Le projet SWAPI est une source d'informations sur l'univers Star Wars. L'API disponible fournit plusieurs bases de données concernant les planètes, les vaisseaux, les véhicules, les personnages, les films et les espèces de la saga venue d'une galaxie très, très lointaine.

1. Commencer par importer des données relatives aux planètes avec la commande :

   ```
   df_planet1 <- fromJSON("https://swapi.dev/api/planets/?format=json")
   ```

   Combien de planètes sont stockées dans `df_planet1` ?

2. À quoi correspondent `df_planet1[["count"]]` et `df_planet1[["next"]]` ?

3. Écrire une boucle pour récupérer les informations de toutes les planètes disponibles dans l'API et stocker le résultat dans un objet `df_planet`. La fonction `rbind_pages` peut être utile ici.

4. Sauvegarder le résultat de la question précédente dans un fichier au format NDJSON.

Web Scraping

Peter Jackson

Nous nous intéressons à la page Wikipedia du réalisateur Peter Jackson :

```
url_jackson <- "https://fr.wikipedia.org/wiki/Peter_Jackson"
```

1. Récupérer les données au format HTML de cette page.

2. Extraire les nœuds `h2` associés au titres de niveau 2.

3. Proposer un sélecteur CSS pour ne récupérer que les titres de niveau 2 des sections du sommaire. Pour information, un sélecteur de classe s'écrit avec un point `.` comme dans `p.ma-classe` pour un paragraphe `<p class="ma-classe">...</p>`.

4. Récupérer les textes des titres avec `html_text`. Comparer avec le résultat obtenu par `html_attrs`.

5. Construire un sélecteur CSS pour récupérer la liste des films de Peter Jackson en tant que réalisateur et les URL des pages Wikipedia associées.

6. Obtenir le même résultat avec XPath.

7. Construire un `tibble` contenant les titres des films réalisés par Peter Jackson ainsi que leur année de sortie et leur durée en minutes.

8. Utiliser les fonctions de `dplyr` pour trouver les 3 films les plus longs réalisés par Peter Jackson.

9. Exporter le `tibble` dans un fichier au format NDJSON.

Trampoline

Le trampoline est un sport olympique depuis les jeux de Sydney en 2000. La page suivante donne accès à la liste de tous les médaillés de cette discipline :

https://fr.wikipedia.org/wiki/Liste_des_m%C3%A9daill%C3%A9s_olympiques_au_trampoline

1. Utiliser la fonction `html_table` pour récupérer le tableau des médaillées féminines dans un data frame.

2. À partir de ce tableau, créer un nouveau data frame contenant, pour chaque pays, le nombre de médailles d'or, d'argent et de bronze obtenues lors des différentes olympiades.

3. Classer ce data frame dans l'ordre usuel en fonction d'abord du nombre de médailles d'or obtenues puis, pour départager les ex-æquo, en fonction du nombre de médailles d'argent et enfin du nombre de médailles de bronze.

4. Mêmes questions pour le tableau masculin et enfin pour le tableau mixte. Le résultat pourra être comparé avec la page : https://fr.wikipedia.org/wiki/Trampoline_aux_Jeux_olympiques

MongoDB

Planètes de Star Wars

Nous reprenons ici les données exportées au format NDJSON à la fin de l'exercice *Star Wars API*.

1. Se connecter à une collection `planet` sur un serveur MongoDB et s'assurer que la collection est vide.
2. Importer les données au format NDJSON dans la collection.
3. Rechercher les planètes dont la période de rotation est égale à 25. Combien y en a-t-il?
4. Même question mais en limitant la réponse aux clés `name`, `rotation_period`, `orbital_period` et `diameter`.
5. Trier les planètes du résultat précédent par diamètre décroissant. Quel est le problème ? Stocker le résultat de la recherche dans un objet **R** et utiliser `str` pour justifier votre réponse
6. Vider la collection et importer de nouveau les données en utilisant la méthode par flux décrite en cours. Utiliser la fonction `handler` pour nettoyer les données :

- convertir les valeurs qui doivent l'être en nombres (ignorer les warnings avec `suppressWarnings`),
- transformer `climate` et `terrain` en tableaux de chaînes de caractères,
- supprimer les colonnes `films`, `gravity`, `residents`, `created` et `edited`.

7. Reprendre la question 5 et vérifier que le résultat est maintenant correct.
8. Extraire les planètes dont le nom commence par `T`.
9. Extraire les planètes dont le diamètre est strictement supérieur à `10000` et où se trouve des montagnes.
10. Rechercher puis supprimer la planète dont le nom est `unknown`.

Agrégation

Planètes de Star Wars (Fin)

Nous continuons avec la collection `planet` créée dans l'exercice précédent.

1. Écrire un agrégateur qui calcule le nombre de planètes dans la base avec le pipeline d'agrégation de MongoDB. Verifier le résultat avec la méthode `count`.
2. Écrire un agrégateur pour calculer le diamètre moyen et la somme des populations des planètes contenant des glaciers avec le pipeline d'agrégation de MongoDB.

Exercices de synthèse

Choisir une des sources d'informations parmi les propositions suivantes (ou en prendre une de votre choix mais après discussion avec le formateur) pour mettre en place un pipeline complet :

- récupération des données depuis une source,
- filtrage et nettoyage du flus de données,
- insertion dans une collection MongoDB,
- mise en place de quelques agrégateurs pertinents (statistique, graphique, ...).

Web Scraping

- Wikipedia (https://fr.wikipedia.org/)

- Sites Fandom de votre choix (https://www.fandom.com/)
- Vélos Specialized (https://www.specialized.com)
- National Weather Service (https://www.weather.gov/)
- Internet Movie Database (https://www.imdb.com/)
- TheTVDB.com (https://www.thetvdb.com/)

Pour Éric : Ski Info (https://www.skiinfo.fr/alpes-du-nord/statistiques.html)

Sources API

- SWAPI (https://swapi.dev/)
- Sites Fandom de votre choix (https://www.fandom.com/)
- Nature OpenSearch (https://www.nature.com/opensearch/)
- OpenLibrary (https://openlibrary.org/dev/docs/json_api)
- Recipe Puppy (http://www.recipepuppy.com/about/api/)

Il existe aussi des moteurs de recherche d'API : https://www.programmableweb.com/apis/directory