Document Data Pipeline - Exercices (Correction)

Xavier Gendre

October 2020

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` ?

```
df1_json <- toJSON(df1, pretty = TRUE)
typeof(df1_json)
```

```
## [1] "character"
```

```
df1
```

```
##             x label
## 1 0.196029585  <NA>
## 2 0.783583747  <NA>
## 3 0.780270480     A
## 4 0.850176823  <NA>
## 5 0.536169684  <NA>
## 6 0.003150467  <NA>
## 7 0.225437056     A
## 8 0.197436132  <NA>
```

```
df1_json
```

```
## [
##   {
##     "x": 0.196
##   },
##   {
##     "x": 0.7836
##   },
##   {
##     "x": 0.7803,
##     "label": "A"
##   },
##   {
##     "x": 0.8502
##   },
```

```
##   {
##     "x": 0.5362
##   },
##   {
##     "x": 0.0032
##   },
##   {
##     "x": 0.2254,
##     "label": "A"
##   },
##   {
##     "x": 0.1974
##   }
## ]
```

2. Convertir `df1_json` en un objet **R** avec `fromJSON`. Le résultat est-il identique à l'objet initial ?

```
df1_bis <- fromJSON(df1_json)
df1_bis
```

```
##        x label
## 1 0.1960  <NA>
## 2 0.7836  <NA>
## 3 0.7803     A
## 4 0.8502  <NA>
## 5 0.5362  <NA>
## 6 0.0032  <NA>
## 7 0.2254     A
## 8 0.1974  <NA>
```

```
all.equal(df1, df1_bis)
```

```
## [1] "Component \"x\": Mean relative difference: 7.042385e-05"
```

3. Faire la même manipulation avec `df2`. Discuter le résultat obtenu.

```
df2
```

```
##      x
## 1 3.14
## 2  NaN
## 3  Inf
```

```
fromJSON(toJSON(df2))
```

```
##      x
## 1 3.14
## 2   NA
## 3   NA
```

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.

```
help(stream_in)
# con <- file("dump.json", open = "wb")
# stream_out(df, con)
# close(con)
```

2. Utiliser la fonction `stream_out` pour afficher les données au format NDJSON. Exporter le résultat dans un fichier `iris.json`.

```
# Affichage sur la sortie standard
stream_out(iris)
```

```
# Export dans un fichier
iris_path <- file.path("data", "iris.json")
con <- file(iris_path, open = "wb")
stream_out(iris, con)
```

```
## Complete! Processed total of 150 rows.
```

```
close(con)
```

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.

```
iris_bis <- stream_in(file(iris_path))
```

```
## opening file input connection.
```

```
##  Found 150 records... Imported 150 records. Simplifying...
```

```
## closing file input connection.
```

```
str(iris_bis)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : chr  "setosa" "setosa" "setosa" "setosa" ...
```

```
all.equal(iris, iris_bis)
```

```
## [1] "Component \"Species\": 'current' is not a factor"
```

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)`.

```
# Fonction dummy_handler appelée une fois
stream_in(file(iris_path), 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?

```
# Fonction dummy_handler appelée 15 fois
# Un appel par page de 10 éléments sur un total de 150
stream_in(file(iris_path), pagesize=10, handler=dummy_handler)
```

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` ?

```
df_planet1[["results"]] %>%
   summarise(n = n())
```

```
##    n
## 1 10
```

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

```
# Nombre total de planètes dans la base
df_planet1[["count"]]
```

```
## [1] 60
```

```
# Prochaine page de 10 planètes
df_planet1[["next"]]
```

```
## [1] "http://swapi.dev/api/planets/?page=2&format=json"
```

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.

```
url_next <- "https://swapi.dev/api/planets/?format=json"
pages <- list()
while(!is.null(url_next)) {
   df <- fromJSON(url_next)
   pages[[length(pages) + 1]] <- df[["results"]]
   url_next <- df[["next"]]
}
df_planet <- rbind_pages(pages)
df_planet %>%
   summarise(n = n())
```

```
##    n
## 1 60
```

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

```
planet_path <- file.path("data", "planet.json")
con <- file(planet_path, open = "wb")
stream_out(df_planet, con)
```

```
## Complete! Processed total of 60 rows.
```

```
close(con)
```

Web Scraping

Peter Jackson

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

https://fr.wikipedia.org/wiki/Peter_Jackson

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

```
url_wikipedia <- "https://fr.wikipedia.org/"
url_jackson <- "wiki/Peter_Jackson"
url <- paste0(url_wikipedia, url_jackson)
data_html <- read_html(url)
```

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

```
data_html %>% html_nodes("h2")
```

```
## {xml_nodeset (8)}
## [1] <h2>Menu de navigation</h2>
## [2] <h2 id="mw-toc-heading">Sommaire</h2>\n
## [3] <h2>\n<span class="mw-headline" id="Biographie">Biographie</span><span cl ...
## [4] <h2>\n<span class="mw-headline" id="Filmographie">Filmographie</span><spa ...
## [5] <h2>\n<span class="mw-headline" id="Distinctions">Distinctions</span><spa ...
## [6] <h2>\n<span class="mw-headline" id="Box-office">Box-office</span><span cl ...
## [7] <h2>\n<span id="Notes_et_r.C3.A9f.C3.A9rences"></span><span class="mw-hea ...
## [8] <h2>\n<span class="mw-headline" id="Voir_aussi">Voir aussi</span><span cl ...
```

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>`.

```
data_html %>% html_nodes("h2 > span.mw-headline")
```

```
## {xml_nodeset (6)}
## [1] <span class="mw-headline" id="Biographie">Biographie</span>
## [2] <span class="mw-headline" id="Filmographie">Filmographie</span>
## [3] <span class="mw-headline" id="Distinctions">Distinctions</span>
## [4] <span class="mw-headline" id="Box-office">Box-office</span>
## [5] <span class="mw-headline" id="Notes_et_références">Notes et références</s ...
## [6] <span class="mw-headline" id="Voir_aussi">Voir aussi</span>
```

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

```
data_html %>% html_nodes("h2 > span.mw-headline") %>% html_text()
```

```
## [1] "Biographie"          "Filmographie"        "Distinctions"
## [4] "Box-office"          "Notes et références" "Voir aussi"
```

```
data_html %>% html_nodes("h2 > span.mw-headline") %>% html_attrs()
```

```
## [[1]]
##         class            id
## "mw-headline"  "Biographie"
##
```

```
## [[2]]
##          class             id
##  "mw-headline" "Filmographie"
##
## [[3]]
##          class             id
##  "mw-headline" "Distinctions"
##
## [[4]]
##         class            id
## "mw-headline"  "Box-office"
##
## [[5]]
##                 class                    id
##         "mw-headline" "Notes_et_références"
##
## [[6]]
##         class            id
## "mw-headline"  "Voir_aussi"
```

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

```
css_selector <- "#mw-content-text > div > ul:nth-of-type(1) > li > i > a"
data_html %>% html_nodes(css_selector) %>% html_attrs() %>% head(2)
```

```
## [[1]]
##              href             title
## "/wiki/Bad_Taste"       "Bad Taste"
##
## [[2]]
##                href               title
## "/wiki/Les_Feebles"       "Les Feebles"
```

5. Obtenir le même résultat avec XPath.

```
xpath_str <- '//*[@id="mw-content-text"]
  /div/ul[
    preceding::h3[span/@id="En_tant_que_réalisateur"]
    and
    following::h3[span/@id="En_tant_que_scénariste"]
  ]/li/i/a'
data_html %>% html_nodes(xpath=xpath_str) %>% html_attrs() %>% head(2)
```

```
## [[1]]
##              href             title
## "/wiki/Bad_Taste"       "Bad Taste"
##
## [[2]]
##                href               title
## "/wiki/Les_Feebles"       "Les Feebles"
```

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

```
# Récupération de la liste des films
data_html <- read_html(paste0(url_wikipedia, url_jackson))
```

```
xpath_films <- '//*[@id="mw-content-text"]
  /div/ul[
    preceding::h3[span/@id="En_tant_que_réalisateur"]
    and
    following::h3[span/@id="En_tant_que_scénariste"]
  ]/li/i/a'
films <- data_html %>% html_nodes(xpath=xpath_films) %>% html_attrs()

# Récupération des informations de chaque film
films_jackson <- tibble()
xpath_duree <- '(//*[@id="mw-content-text"]
  //table/tbody/tr/td[
    preceding::th[text()="Durée"]
  ])[1]'
xpath_sortie <- '(//*[@id="mw-content-text"]
  //table/tbody/tr/td[
    preceding::th[text()="Sortie"]
  ])[1]'
for(i in seq_along(films)) {
   url_film <- films[[i]]["href"]
   data_html <- paste0(url_wikipedia, url_film) %>% read_html()
   # Extraction de la durée en minutes (hors version longue)
   film_duree <- data_html %>%
      html_nodes(xpath=xpath_duree) %>%
      html_text() %>%
      str_extract("[0-9]+")
   # Extraction de l'année de sortie
   film_sortie <- data_html %>%
      html_nodes(xpath=xpath_sortie) %>%
      html_text() %>%
      str_extract("[0-9]+")
   films_jackson <- films_jackson %>% rbind(tibble(titre = films[[i]]["title"],
                                                   duree = as.integer(film_duree),
                                                   sortie = as.integer(film_sortie)))
}

# Résultat
films_jackson
```

```
## # A tibble: 15 x 3
##    titre                                              duree sortie
##    <chr>                                              <int>  <int>
##  1 Bad Taste                                             91   1987
##  2 Les Feebles                                           94   1989
##  3 Braindead                                            104   1992
##  4 Créatures célestes                                    99   1994
##  5 Forgotten Silver                                      53   1995
##  6 Fantômes contre fantômes                             110   1996
##  7 Le Seigneur des anneaux : La Communauté de l'anneau  178   2001
##  8 Le Seigneur des anneaux : Les Deux Tours             179   2002
##  9 Le Seigneur des anneaux : Le Retour du roi           201   2003
## 10 King Kong (film, 2005)                               187   2005
## 11 Lovely Bones                                         135   2009
## 12 Le Hobbit : Un voyage inattendu                      169   2012
```

```
## 13 Le Hobbit : La Désolation de Smaug                    161   2013
## 14 Le Hobbit : La Bataille des Cinq Armées               144   2014
## 15 Pour les soldats tombés                                99   2018
```

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

```
films_jackson %>%
   arrange(desc(duree)) %>%
   head(3)
```

```
## # A tibble: 3 x 3
##   titre                                    duree sortie
##   <chr>                                    <int>  <int>
## 1 Le Seigneur des anneaux : Le Retour du roi   201   2003
## 2 King Kong (film, 2005)                       187   2005
## 3 Le Seigneur des anneaux : Les Deux Tours     179   2002
```

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

```
jackson_path <- file.path("data", "jackson.json")
con <- file(jackson_path, open = "wb")
stream_out(films_jackson, con)
```

```
## Complete! Processed total of 15 rows.
```

```
close(con)
```

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.

```
url <- "https://fr.wikipedia.org/wiki/Liste_des_m%C3%A9daill%C3%A9s_olympiques_au_trampoline"
data_html <- read_html(url)

css_selector <- "#mw-content-text > div > table:nth-of-type(2)"
df_femmes <- data_html %>% html_nodes(css_selector) %>% html_table()
df_femmes[[1]]
```

```
##   Édition           Lieu                        Or                  Argent
## 1    2000         Sydney     Irina Karavaeva (RUS) Oksana Tsyhulyeva (UKR)
## 2    2004        Athènes      Anna Dogonadze (GER)    Karen Cockburn (CAN)
## 3    2008          Pékin            He Wenna (CHN)    Karen Cockburn (CAN)
## 4    2012        Londres Rosannagh MacLennan (CAN)    Huang Shanshan (CHN)
## 5    2016 Rio de Janeiro Rosannagh MacLennan (CAN)       Bryony Page (GBR)
##                   Bronze
## 1   Karen Cockburn (CAN)
## 2   Huang Shanshan (CHN)
## 3 Ekaterina Khilko (UZB)
## 4         He Wenna (CHN)
## 5           Li Dan (CHN)
```

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.

```
medaille_femmes <- df_femmes[[1]] %>%
   mutate(MédailleOr     = str_extract(Or, "(?<=\\().*(?=\\))"),
          MédailleArgent = str_extract(Argent, "(?<=\\().*(?=\\))"),
          MédailleBronze = str_extract(Bronze, "(?<=\\().*(?=\\))")) %>%
   select(starts_with("Médaille")) %>%
   gather(key = "Médaille", value = "Pays") %>%
   group_by(Pays, Médaille) %>%
   summarise(n = n(), .groups = 'drop') %>%
   spread(Médaille, n, fill = 0) %>%
   rename(Or     = MédailleOr,
          Argent = MédailleArgent,
          Bronze = MédailleBronze) %>%
   relocate(Pays, Or, Argent, Bronze)

medaille_femmes
```

```
## # A tibble: 7 x 4
##   Pays     Or Argent Bronze
##   <chr> <dbl>  <dbl>  <dbl>
## 1 CAN       2      2      1
## 2 CHN       1      1      3
## 3 GBR       0      1      0
## 4 GER       1      0      0
## 5 RUS       1      0      0
## 6 UKR       0      1      0
## 7 UZB       0      0      1
```

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.

```
medaille_femmes %>%
   arrange(desc(Or), desc(Argent), desc(Bronze))
```

```
## # A tibble: 7 x 4
##   Pays     Or Argent Bronze
##   <chr> <dbl>  <dbl>  <dbl>
## 1 CAN       2      2      1
## 2 CHN       1      1      3
## 3 GER       1      0      0
## 4 RUS       1      0      0
## 5 GBR       0      1      0
## 6 UKR       0      1      0
## 7 UZB       0      0      1
```

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

```
# Médailles hommes
css_selector <- "#mw-content-text > div > table:nth-of-type(1)"
df_hommes <- data_html %>% html_nodes(css_selector) %>% html_table()
medaille_hommes <- df_hommes[[1]] %>%
   mutate(MédailleOr     = str_extract(Or, "(?<=\\().*(?=\\))"),
          MédailleArgent = str_extract(Argent, "(?<=\\().*(?=\\))"),
          MédailleBronze = str_extract(Bronze, "(?<=\\().*(?=\\))")) %>%
   select(starts_with("Médaille")) %>%
```

```
   gather(key = "Médaille", value = "Pays") %>%
   group_by(Pays, Médaille) %>%
   summarise(n = n(), .groups = 'drop') %>%
   spread(Médaille, n, fill = 0) %>%
   rename(Or     = MédailleOr,
          Argent = MédailleArgent,
          Bronze = MédailleBronze) %>%
   relocate(Pays, Or, Argent, Bronze)
medaille_hommes %>%
   arrange(desc(Or), desc(Argent), desc(Bronze))
```

```
## # A tibble: 7 x 4
##   Pays     Or Argent Bronze
##   <chr> <dbl>  <dbl>  <dbl>
## 1 CHN       2      1      3
## 2 RUS       1      2      0
## 3 BLR       1      0      0
## 4 UKR       1      0      0
## 5 CAN       0      1      1
## 6 AUS       0      1      0
## 7 GER       0      0      1
```

```
# Médailles mixte
medaille_femmes %>%
   full_join(medaille_hommes, by="Pays") %>%
   mutate(Or     = ifelse(is.na(Or.x), 0, Or.x) +
                   ifelse(is.na(Or.y), 0, Or.y),
          Argent = ifelse(is.na(Argent.x), 0, Argent.x) +
                   ifelse(is.na(Argent.y), 0, Argent.y),
          Bronze = ifelse(is.na(Bronze.x), 0, Bronze.x) +
                   ifelse(is.na(Bronze.y), 0, Bronze.y)) %>%
   select(Pays, Or, Argent, Bronze) %>%
   arrange(desc(Or), desc(Argent), desc(Bronze))
```

```
## # A tibble: 9 x 4
##   Pays     Or Argent Bronze
##   <chr> <dbl>  <dbl>  <dbl>
## 1 CHN       3      2      6
## 2 CAN       2      3      2
## 3 RUS       2      2      0
## 4 UKR       1      1      0
## 5 GER       1      0      1
## 6 BLR       1      0      0
## 7 GBR       0      1      0
## 8 AUS       0      1      0
## 9 UZB       0      0      1
```

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.

```
m <- mongo("planet")
if(m$count() > 0) m$drop()
m$count()
```

```
## [1] 0
```

2. Importer les données au format NDJSON dans la collection.

```
m$import(file(planet_path))
m$count()
```

```
## [1] 60
```

3. Rechercher les planètes dont la période de rotation est égale à 25. Combien y en a-t-il?

```
m$find(query = '{"rotation_period": "25"}') %>% head(2)
```

```
##             name rotation_period orbital_period diameter          climate
## 1 Cato Neimoidia              25            278        0 temperate, moist
## 2       Corellia              25            329    11000        temperate
##      gravity                                 terrain surface_water population
## 1 1 standard mountains, fields, forests, rock arches       unknown   10000000
## 2 1 standard           plains, urban, hills, forests            70 3000000000
##                                                        residents
## 1                                 http://swapi.dev/api/people/33/
## 2 http://swapi.dev/api/people/14/, http://swapi.dev/api/people/18/
##                            films                    created
## 1 http://swapi.dev/api/films/6/ 2014-12-10T13:46:28.704000Z
## 2                               2014-12-10T16:49:12.453000Z
##                         edited                              url
## 1 2014-12-20T20:58:18.449000Z http://swapi.dev/api/planets/18/
## 2 2014-12-20T20:58:18.456000Z http://swapi.dev/api/planets/22/
```

```
m$count('{"rotation_period": "25"}')
```

```
## [1] 5
```

4. Même question mais en limitant la réponse aux clés `name`, `rotation_period`, `orbital_period` et `diameter`.

```
m$find(query = '{"rotation_period": "25"}',
       fields = '{"_id": 0,
                  "name": 1,
                  "rotation_period": 1,
                  "orbital_period": 1,
                  "diameter": 1}')
```

```
##             name rotation_period orbital_period diameter
## 1 Cato Neimoidia              25            278        0
## 2       Corellia              25            329    11000
## 3      Dantooine              25            378     9830
## 4      Trandosha              25            371        0
## 5     Haruun Kal              25            383    10120
```

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

```
df <- m$find(query = '{"rotation_period": "25"}',
             fields = '{"_id": 0,
```

```
                        "name": 1,
                        "rotation_period": 1,
                        "orbital_period": 1,
                        "diameter": 1}',
             sort = '{"diameter": -1}')
print(df)
```

```
##             name rotation_period orbital_period diameter
## 1      Dantooine              25            378     9830
## 2       Corellia              25            329    11000
## 3     Haruun Kal              25            383    10120
## 4 Cato Neimoidia              25            278        0
## 5      Trandosha              25            371        0
```

```
str(df)
```

```
## 'data.frame':    5 obs. of  4 variables:
##  $ name           : chr  "Dantooine" "Corellia" "Haruun Kal" "Cato Neimoidia" ...
##  $ rotation_period: chr  "25" "25" "25" "25" ...
##  $ orbital_period : chr  "378" "329" "383" "278" ...
##  $ diameter       : chr  "9830" "11000" "10120" "0" ...
```

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`.

```
m$drop()

custom_handler <- function(df) {
  # Convertir les valeurs qui doivent l'être en nombres
  df$rotation_period <- suppressWarnings(as.double(df$rotation_period))
  df$orbital_period  <- suppressWarnings(as.double(df$orbital_period))
  df$diameter        <- suppressWarnings(as.double(df$diameter))
  df$surface_water   <- suppressWarnings(as.double(df$surface_water))
  df$population      <- suppressWarnings(as.double(df$population))

  # Transformer `climate` et `terrain` en tableaux de chaînes de caractères
  df$climate <- strsplit(df$climate, ", ")
  df$terrain <- strsplit(df$terrain, ", ")

  # Supprimer les colonnes `films`, `gravity`, `residents`, `created` et `edited`
  df$created   <- NULL
  df$edited    <- NULL
  df$films     <- NULL
  df$gravity   <- NULL
  df$residents <- NULL

  ftmp <- file(tempfile(), open="w+b")
  stream_out(df, ftmp)
  m$import(ftmp)
  close(ftmp)
}
```

```
stream_in(file(planet_path), handler = custom_handler)
```

```
## using a custom handler function.
## opening file input connection.
## Complete! Processed total of 60 rows.
##  Found 60 records...
## closing file input connection.
```

```
m$find() %>% head(2)
```

```
##       name rotation_period orbital_period diameter   climate
## 1 Tatooine              23            304    10465      arid
## 2 Alderaan              24            364    12500 temperate
##                 terrain surface_water population
## 1                desert             1      2e+05
## 2 grasslands, mountains            40      2e+09
##                               url
## 1 http://swapi.dev/api/planets/1/
## 2 http://swapi.dev/api/planets/2/
```

7. Reprendre la question 5 et vérifier que le résultat est maintenant correct.

```
m$find(query = '{"rotation_period": 25}',
       fields = '{"_id": 0,
                  "name": 1,
                  "rotation_period": 1,
                  "orbital_period": 1,
                  "diameter": 1}',
       sort = '{"diameter": -1}')
```

```
##             name rotation_period orbital_period diameter
## 1       Corellia              25            329    11000
## 2     Haruun Kal              25            383    10120
## 3      Dantooine              25            378     9830
## 4 Cato Neimoidia              25            278        0
## 5      Trandosha              25            371        0
```

8. Extraire les planètes dont le nom commence par `T`.

```
m$find(query='{"name": {"$regex": "^T", "$options" : "i"} }',
       fields='{"_id": 0, "name": 1}')
```

```
##        name
## 1  Tatooine
## 2 Trandosha
## 3  Toydaria
## 4   Troiken
## 5      Tund
## 6   Tholoth
```

9. Extraire les planètes dont le diamètre est strictement supérieur à `10000` et où se trouve des montagnes.

```
m$find(query='{"$and": [{"diameter": {"$gt": 10000}},
                        {"terrain": {"$in": ["mountains"]}}] }',
       fields='{"_id": 0, "name": 1, "diameter": 1, "terrain": 1}')
```

```
##         name diameter                                   terrain
```

```
## 1   Alderaan    12500                    grasslands, mountains
## 2      Naboo    12120 grassy hills, swamps, forests, mountains
## 3  Coruscant    12240                     cityscape, mountains
## 4    Mygeeto    10088         glaciers, mountains, ice canyons
## 5  Saleucami    14920      caves, desert, mountains, volcanoes
## 6    Sullust    12780      mountains, volcanoes, rocky deserts
## 7  Malastare    18880      swamps, deserts, jungles, mountains
## 8     Ryloth    10600      mountains, valleys, deserts, tundra
## 9 Muunilinst    13800        plains, forests, hills, mountains
```

10. Rechercher puis supprimer la planète dont le nom est `unknown`.

```
m$find(query='{"name": "unknown" }')
```

```
##      name rotation_period orbital_period diameter climate terrain
## 1 unknown               0              0        0 unknown unknown
##                               url
## 1 http://swapi.dev/api/planets/28/
```

```
m$remove(query='{"name": "unknown" }')
m$find(query='{"name": "unknown" }')
```

```
## data frame with 0 columns and 0 rows
```

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`.

```
# Pipeline d'agrégation
m$aggregate('[
  { "$group": { "_id": null, "count": { "$sum": 1 } } }
]')
```

```
##   _id count
## 1  NA    59
```

```
# Vérification
m$count()
```

```
## [1] 59
```

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.

```
m$aggregate('[
  { "$match": {"terrain": {"$in": ["glaciers"] } } },
  { "$group": { "_id": null,
                "diameter": { "$avg": "$diameter" },
                "population": { "$sum": "$population" } } }
]')
```

```
##   _id diameter population
## 1  NA    10088  519000000
```

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