Lecture et écriture dans des fichiers externes à partir de R

Sophie Baillargeon, Université Laval

2018-01-15

Table des matières

Maintenant que les objets R pouvant servir de structure de données ont été vus, voyons comment lire et écrire dans des fichiers externes à partir de R. Dans un cadre d'analyse de données, lire et écrire dans des fichiers permet d'importer et d'exporter des données.

Répertoire courant

Avant toute chose, revenons sur le répertoire courant (ou répertoire de travail, en anglais *working directory*). Afin d'accéder à des données en R, il faut souvent aller les lire dans un fichier. Le plus souvent, ce fichier se situe sur l'ordinateur local, dans un certain répertoire. Il faut donc aller lire son contenu à cet endroit.

Dans toute commande R effectuant une lecture ou de l'écriture dans un fichier, il faut identifier le fichier. Il peut être identifié en utilisant son chemin d'accès complet, ou encore en utilisant un chemin d'accès relatif à un emplacement courant. Peu importe l'option choisie, le chemin doit se terminer par le nom complet du fichier, incluant son extension.

Par exemple, supposons que le fichier `dataEx.txt` se trouve dans le répertoire `C:\coursR` de l'ordinateur d'un utilisateur de R sous Windows. Le chemin d'accès complet de ce fichier est `C:\coursR\dataEx.txt`. Le chemin d'accès du fichier relatif au répertoire `C:\coursR` est pour sa part simplement le nom du fichier, soit `dataEx.txt`.

Remarque importante : Bien que le caractère utilisé pour séparer les composantes dans un chemin d'accès sous Windows soit `\`, ce caractère est réservé en R pour les "séquences d'échappement". Il s'agit de séquences ayant une signification particulière dans une chaîne de caractères (par exemple `"\t"` représente une tabulation, `"\n"` représente un retour de chariot, etc.). Dans un chemin d'accès en R, il faut plutôt séparer les composantes par le caractère `/`, ou encore par `\\`.

Lorsqu'un fichier est identifié par un chemin relatif dans une instruction, R doit faire une supposition quant au répertoire à partir duquel chercher le fichier. C'est justement l'utilité du répertoire courant. Il s'agit de l'emplacement de base, celui à partir duquel débuter un chemin d'accès à un fichier lorsque le début du chemin n'est pas spécifié par l'utilisateur.

Pour connaître le répertoire courant d'une session R, il suffit de soumettre la commande suivante :

```
getwd()
```

Pour accéder au fichier `dataEx.txt` en utilisant seulement son nom, il faut d'abord s'assurer que le répertoire courant de la session R correspond au répertoire contenant le fichier. La fonction `setwd` permet de modifier le répertoire courant. Par exemple, la commande suivante change le répertoire courant pour `C:\coursR`.

```
setwd("C:/coursR")
```

Dans une commande R, un chemin d'accès doit toujours être encadré de guillemets.

Si le répertoire courant avait été fixé à `"C:"`, le chemin d'accès relatif du fichier aurait été `"/coursR/dataEx.txt"` (le `"/"` au début du chemin est nécessaire dans une commande R).

Accéder au système de fichiers de notre ordinateur à partir de R

Ainsi, R à accès au système de fichiers de l'ordinateur à partir duquel la session a été démarrée. Par défaut, l'accès se fait à partir du répertoire courant. Directement dans la console R, il est possible de voir le contenu du répertoire courant, créer des fichiers et des répertoires, effacer des fichiers, etc. Par exemple, la fonction `dir` permet d'afficher le contenu d'un répertoire (par défaut du répertoire courant).

```
dir()
```

```
## [1] "dataEx.csv"  "dataEx.json" "dataEx.txt"  "dataEx.xlsx"
```

Pour plus d'information à ce propos, la fiche d'aide nommée `files` (ouverte avec la commande `help(files)`) est un bon point de départ.

Aussi, un bon truc pour trouver facilement le nom complet d'un fichier, qui respecte la syntaxe de R, est d'utiliser la commande suivante :

```
file.choose()
```

Cette commande ouvre une fenêtre pour naviguer dans le système de fichiers et sélectionner un fichier. Une fois la sélection du fichier complétée, la commande `file.choose()` retourne une chaîne de caractères contenant le chemin d'accès complet du fichier sélectionné.

Projets RStudio

Il est souvent judicieux de rassembler sous un même répertoire un programme R et tous les fichiers qu'il utilise. Les fichiers de données peuvent même être placés dans un sous-répertoire, surtout s'ils sont nombreux.

RStudio offre l'option de transformer un répertoire qui rassemble un ou des programmes R et leurs fichiers associés en « projet ». Des explications à propos de l'utilisation de projets RStudio se trouvent sur la page web suivante : https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects.

Les projets RStudio facilitent la gestion du répertoire courant. En ouvrant un projet RStudio, une nouvelle session R est démarrée ayant pour répertoire courant le répertoire de base du projet.

Lecture de fichiers

Une analyse de données en R débute typiquement par l'importation de données. L'importation consiste à lire le contenu d'un fichier externe et à enregistrer ce contenu dans un objet R.

Les fonctions de base en R pour lire dans des fichiers externes sont `scan` et `readLines`. Nous ne verrons pas comment utiliser ces fonctions, car il existe des fonctions mieux adapter à la lecture de fichiers contenant des données à analyser.

Notons que les données à importer en R peuvent provenir de toutes sortes de sources. Elles peuvent se trouver dans des fichiers sous un certain format, dans des bases de données, ou provenir directement du web. En outre, il existe plusieurs formats de fichiers pour stocker des données : texte (dont CSV), JSON, EXCEL, formats propres à un logiciel statistique particulier, etc. Nous aborderons ici uniquement les formats texte, CSV, JSON, EXCEL, ainsi que les formats propres à R (vus plus loin). La lecture de table HTML sur le web sera aussi introduite. Pour en savoir plus sur les autres possibilités, des références sont fournies en fin de document.

Les données du data frame suivant seront utilisées dans les exemples. Il s'agit de données créées dans les notes sur les structures de données en R, dans lesquelles deux valeurs ont été remplacées par des valeurs manquantes.

```
de1 <- c(2,3,4,1,2,3,5,6,5,4)
de2 <- c(1,4,2,3,5,4,6,2,5,3)
lanceur <- rep(c("Luc", "Kim"), each = 5)
dataEx <- data.frame(de1, de2, lanceur)
# Introduction de valeurs manquantes
dataEx$de2[7] <- NA
dataEx$lanceur[3] <- NA
# Affichage du date frame
dataEx
```

```
##    de1 de2 lanceur
## 1    2   1     Luc
## 2    3   4     Luc
## 3    4   2    <NA>
## 4    1   3     Luc
## 5    2   5     Luc
## 6    3   4     Kim
## 7    5  NA     Kim
## 8    6   2     Kim
## 9    5   5     Kim
## 10   4   3     Kim
```

Ces données se rapportent à une expérience fictive de lancers de dés.

Importation de données à partir d'un fichier texte

La lecture de données dans un fichier texte se fait principalement avec la fonction R `read.table`. Par exemple, allons lire les données contenues dans le fichier `dataEx.txt`, qui a l'allure suivante.

```
de1 de2 lanceur
2   1   Luc
3   4   Luc
4   2   .
1   3   Luc
2   5   Luc
3   4   Kim
5   .   Kim
6   2   Kim
5   5   Kim
4   3   Kim
```

Remarquons que les valeurs manquantes dans ce fichier sont représentées par des points. Importons ces données avec `read.table`.

```
dataEx_txt <- read.table("C:/coursR/dataEx.txt")
```

Le seul argument obligatoire dans la fonction `read.table` est le chemin d'accès au fichier à lire. Ici, il n'aurait pas été nécessaire de spécifier le chemin d'accès complet du fichier puisque ce fichier se trouve dans le répertoire courant de la session R. Dans tous les exemples suivants, seul le nom du fichier sera fourni.

Les autres arguments de la fonction `read.table` doivent être adaptés selon le formatage du fichier à lire (voir la fiche d'aide de la fonction). Dans l'exemple ci-dessus, est-ce que toutes les valeurs par défaut des arguments convenaient vraiment ? Jetons un coup d'oeil à l'objet obtenu.

```
dataEx_txt
```

```
##     V1  V2      V3
## 1  de1 de2 lanceur
## 2    2   1     Luc
## 3    3   4     Luc
## 4    4   2       .
## 5    1   3     Luc
## 6    2   5     Luc
## 7    3   4     Kim
## 8    5   .     Kim
## 9    6   2     Kim
## 10   5   5     Kim
## 11   4   3     Kim
```

```
str(dataEx_txt)
```

```
## 'data.frame':    11 obs. of  3 variables:
##  $ V1: Factor w/ 7 levels "1","2","3","4",..: 7 2 3 4 1 2 3 5 6 5 ...
##  $ V2: Factor w/ 7 levels ".","1","2","3",..: 7 2 5 3 4 6 5 1 3 6 ...
##  $ V3: Factor w/ 4 levels ".","Kim","lanceur",..: 3 4 4 1 4 4 2 2 2 2 ...
```

Il s'agit d'un data frame. La fonction `read.table` retourne toujours un objet de ce type.

Nous pouvons tout de suite remarquer un problème avec ce data frame. Le nom des variables a été interprété comme une observation. Il faudrait utiliser l'argument `header` pour indiquer que la première ligne du fichier contient les noms des variables.

```
dataEx_txt <- read.table("dataEx.txt", header = TRUE)
str(dataEx_txt)
```

```
## 'data.frame':    10 obs. of  3 variables:
##  $ de1    : int  2 3 4 1 2 3 5 6 5 4
##  $ de2    : Factor w/ 6 levels ".","1","2","3",..: 2 5 3 4 6 5 1 3 6 4
```

```
##  $ lanceur: Factor w/ 3 levels ".","Kim","Luc": 3 3 1 3 3 2 2 2 2 2
```

La première variable, `de1`, a été correctement lu, mais il y a un problème avec la deuxième variable, `de2`. Elle est un facteur. Cette variable est en réalité numérique et non catégorique (il s'agit du résultat d'un lancer de dé). Afin de pouvoir faire des calculs numériques sur cette variable, elle doit être stockée dans un vecteur (= colonne du data frame) de valeurs numériques. R a cru que les valeurs dans cette colonne étaient de type caractère à cause de la valeur manquante représentée par un point. L'argument `na.strings` de `read.table` indique à R les chaînes de caractères à interpréter comme des valeurs manquantes. Par défaut, `na.strings` prend la valeur `"NA"`. Il faut donc changer la valeur de cet argument comme suit.

```
dataEx_txt <- read.table("dataEx.txt", header = TRUE, na.strings = ".")
str(dataEx_txt)
```

```
## 'data.frame':    10 obs. of  3 variables:
##  $ de1    : int  2 3 4 1 2 3 5 6 5 4
##  $ de2    : int  1 4 2 3 5 4 NA 2 5 3
##  $ lanceur: Factor w/ 2 levels "Kim","Luc": 2 2 NA 2 2 1 1 1 1 1
```

Nous avons réglé le problème de lecture de la deuxième colonne. Attardons-nous maintenant à la dernière colonne. Elle est enregistrée dans un facteur à 2 niveaux. C'est potentiellement ce que nous voulons. Par défaut, `read.table` transforme toutes les colonnes contenant des valeurs caractères en facteur (comme le fait la fonction `data.frame`). Pour obtenir plutôt un vecteur de chaînes de caractères, il faut modifier la valeur de l'argument `stringsAsFactors` comme suit.

```
dataEx_txt <- read.table("dataEx.txt", header = TRUE, na.strings = ".",
                         stringsAsFactors = FALSE)
str(dataEx_txt)
```

```
## 'data.frame':    10 obs. of  3 variables:
##  $ de1    : int  2 3 4 1 2 3 5 6 5 4
##  $ de2    : int  1 4 2 3 5 4 NA 2 5 3
##  $ lanceur: chr  "Luc" "Luc" NA "Luc" ...
```

Nous avons illustré ici trois arguments de la fonction `read.table`. Elle en possède plusieurs autres, à adapter selon le fichier à lire.

Résumé des arguments les plus utiles de la fonction `read.table`

Chaque argument de la fonction `read.table` est en lien avec une question que l'utilisateur doit se poser lors de la lecture d'un fichier de données.

- Est-ce que les noms des variables sont sur la première ligne ?
  - Si oui, utiliser `header = TRUE`, sinon `header = FALSE`.
  - Par défaut `header = FALSE`, sauf si la première ligne contient un élément de moins que les lignes suivantes. Dans ce cas, R considère que la première ligne contient les noms des variables et que les lignes suivantes contiennent les observations précédées d'un nom de ligne.
- Quel caractère sépare les valeurs ?
  - Fournir ce caractère comme valeur à l'argument `sep`.
  - Par défaut, `sep = ""`, ce qui signifie que le séparateur est un « blanc », soit un espace ou plus, une tabulation, une fin de ligne ou un retour de chariot.
- Quel est le symbole décimal ?
  - Fournir ce symbole sous forme de chaîne de caractères à l'argument `dec`.
  - Par défaut `dec = "."`
- Comment sont représentées les valeurs manquantes ?
  - Fournir cette représentation sous forme de chaîne de caractères à l'argument `na.strings`.
  - Par défaut, `na.strings = "NA"`.
- Est-ce que certaines valeurs sont encadrées ? Si oui, par quel symbole ?

  - S'il y a des valeurs encadrées, fournir à l'argument `quote` une chaîne de caractère contenant une concaténation de tous les caractères utilisés pour encadrer ces valeurs.
  - Par défaut `quote = "\"'"`, ce qui signifie qu'un guillemet double ou simple indique le début ou la fin d'une valeur. La barre oblique inversée (\) devant le deuxième `"` dans `"\"'"` est nécessaire afin que ce `"` ne soit pas interprété comme la fin de la chaîne de caractères fournie en valeur d'entrée à `quote`.
- Est-ce qu'il y a des caractères spéciaux, par exemple des accents ? Si oui, quel encodage est utilisé ?
  - Un des encodages les plus courants est `UTF-8`. En cas de problèmes avec des accents mal lus, essayer l'argument `encoding = "UTF-8"`. Ça pourrait régler le problème.
  - Par défaut, l'encodage est supposé inconnu.
- Est-ce que les données ou les noms des variables débutent à la ligne 1 ?
  - Si ce n'est pas le cas, fournir à l'argument `skip` le nombre de lignes à ne pas lire au début du fichier.
  - Par défaut, `skip = 0`, donc aucune ligne n'est sautée.
- Est-ce que les données se terminent à la dernière ligne non vide ?
  - Si ce n'est pas le cas, fournir à l'argument `nrows` le nombre de lignes à lire.
  - Par défaut, `nrows` prend une valeur ignorée, donc la lecture des lignes se termine à la fin du fichier.
- Est-ce que le fichier contient des commentaires ? Si oui, quel signe précède ces lignes ?
  - Fournir à `comment.char` un seul caractère, soit celui indiquant que le reste d'une ligne contient un commentaire. Pour empêcher la détection de commentaires, utiliser `comment.char = ""`.
  - Par défaut, `comment.char = "#"`.
- Est-ce que les colonnes contenant des chaînes de caractères doivent être converties en facteurs ?
  - Si oui, utiliser `stringsAsFactors = TRUE`, sinon `stringsAsFactors = TRUE`.
  - Par défaut `stringsAsFactors = TRUE`.

Tous les détails ainsi que la description des autres arguments de la fonction `read.table` sont dans la fiche d'aide de la fonction `read.table`.

Importation de données en format CSV

Le format CSV est un format texte de données assez usuel. CSV signifie Comma Separated Values. En fait, dans un fichier CSV, les valeurs sont séparées par des virgules ou des points-virgules. Le fichier `dataEx.csv` est un exemple de fichier sous ce format. Il a l'allure suivante.

```
"de1";"de2";"lanceur"
2;1;"Luc"
3;4;"Luc"
4;2;
1;3;"Luc"
2;5;"Luc"
3;4;"Kim"
5;;"Kim"
6;2;"Kim"
5;5;"Kim"
4;3;"Kim"
```

La fonction `read.csv2` est tout indiquée pour importer ce fichier puisque la valeur par défaut de son argument `sep` est `";"`. Cette fonction appelle en fait la fonction `read.table`, mais avec des valeurs par défaut différentes pour les arguments. Ce type de fonction est appelé fonction enveloppe (en anglais *wrapper*). La fonction `read.table` possède 4 fonctions enveloppe : `read.csv`, `read.csv2`, `read.delim` et `read.delim2`.

Les arguments de la fonction `read.csv2` ont-ils toutes les bonnes valeurs par défaut pour importer le fichier `dataEx.csv` ?

```
dataEx_csv <- read.csv2("dataEx.csv")
str(dataEx_csv)
```

```
## 'data.frame':    10 obs. of  3 variables:
##  $ de1    : int  2 3 4 1 2 3 5 6 5 4
##  $ de2    : int  1 4 2 3 5 4 NA 2 5 3
##  $ lanceur: Factor w/ 3 levels "","Kim","Luc": 3 3 1 3 3 2 2 2 2 2
```

Ici, la première ligne a correctement été interprétée comme le nom des variables, parce que l'argument `header` de `read.csv2` prend par défaut la valeur `TRUE`. Aussi, la valeur manquante a été bien interprétée pour la variable numérique, mais pas pour la variable catégorique. Utilisons donc l'argument `na.strings` comme suit pour faire correctement l'importation.

```
dataEx_csv <- read.csv2("dataEx.csv", na.strings = "")
str(dataEx_csv)
```

```
## 'data.frame':    10 obs. of  3 variables:
##  $ de1    : int  2 3 4 1 2 3 5 6 5 4
##  $ de2    : int  1 4 2 3 5 4 NA 2 5 3
##  $ lanceur: Factor w/ 2 levels "Kim","Luc": 2 2 NA 2 2 1 1 1 1 1
```

La leçon à retenir de ces exemples est qu'il faut toujours vérifier que l'objet R dans lequel nous avons importé des données contient les bonnes données et sous le bon format. Si ce n'est pas le cas, la commande d'importation de données doit être corrigée. Plusieurs tentatives sont parfois nécessaires afin d'arriver à effectuer correctement l'importation.

Importation de données en format JSON

Le format de données JSON semble gagner en popularité ces dernières années. Ce ne sont pas toutes les données qui conviennent au format "tableau de données" avec des observations en ligne et des variables en colonnes. Le format JSON n'est pas contraint à des données de ce type. Un fichier JSON est en fait un fichier texte, contenant des paires attributs - valeurs. Par exemple, les données utilisées dans les exemples précédents ont l'allure suivante en format JSON.

```
[
    {
      "de1": 2,
      "de2": 1,
      "lanceur": "Luc"
    },
    {
      "de1": 3,
      "de2": 4,
      "lanceur": "Luc"
    },
    {
      "de1": 4,
      "de2": 2,
      "lanceur": null
    },
    {
      "de1": 1,
      "de2": 3,
      "lanceur": "Luc"
    },
    {
      "de1": 2,
      "de2": 5,
```

```
    "lanceur": "Luc"
  },
  {
    "de1": 3,
    "de2": 4,
    "lanceur": "Kim"
  },
  {
    "de1": 5,
    "de2": null,
    "lanceur": "Kim"
  },
  {
    "de1": 6,
    "de2": 2,
    "lanceur": "Kim"
  },
  {
    "de1": 5,
    "de2": 5,
    "lanceur": "Kim"
  },
  {
    "de1": 4,
    "de2": 3,
    "lanceur": "Kim"
  }
]
```


Le fichier comporte un élément pour chacune des 10 observations. Pour chaque observation, la valeur prise pour chacune des variables est spécifiée. Ce format est plus souple qu'un tableau de données, car la valeur d'une variable pour une observation pourrait être de n'importe quelle dimension, plutôt que d'être une seule donnée. Remarquons que dans le format JSON, une donnée manquante est représentée par `null` plutôt que `NA`.

Aucune fonction de l'installation de base de R ne permet de lire ce format de fichier. Cependant, quelques packages offrent des fonctions pour ce faire. C'est le cas de la fonction `fromJSON` du package `jsonlite`. Voici un exemple d'utilisation de cette fonction.

```
library(jsonlite)
dataEx_json <- fromJSON("dataEx.json")
str(dataEx_json)
```

```
## 'data.frame':    10 obs. of  3 variables:
##  $ de1    : int  2 3 4 1 2 3 5 6 5 4
##  $ de2    : int  1 4 2 3 5 4 NA 2 5 3
##  $ lanceur: chr  "Luc" "Luc" NA "Luc" ...
```

La fonction `fromJSON` a converti le contenu du fichier `dataEx.json` en data frame.

Importation de données à partir d'un fichier EXCEL

Afin de lire en R des données provenant d'un fichier EXCEL, une première option est d'enregistrer le fichier dans un format texte, puis de l'importer en R avec les moyens vus précédemment. Cette procédure n'est cependant pas automatique. Il existe maintenant quelques packages R offrant des fonctions pour lire

	A	B	C
1	de1	de2	lanceur
2	2	1	Luc
3	3	4	Luc
4	4	2	
5	1	3	Luc
6	2	5	Luc
7	3	4	Kim
8	5		Kim
9	6	2	Kim
10	5	5	Kim
11	4	3	Kim

FIGURE 1 –

directement dans un fichier EXCEL. Un de ces packages est `readxl`. Supposons que nous voulons importer en R le fichier EXCEL `dataEx.xlsx` ayant l'allure suivante.

Nous pourrions procéder comme suit.

```
library(readxl)
dataEx_xlsx <- read_excel("dataEx.xlsx")
str(dataEx_xlsx)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    10 obs. of  3 variables:
##  $ de1    : num  2 3 4 1 2 3 5 6 5 4
##  $ de2    : num  1 4 2 3 5 4 NA 2 5 3
##  $ lanceur: chr  "Luc" "Luc" NA "Luc" ...
```

Remarquons que la colonne contenant des valeurs caractères n'a pas été transformée en un facteur comme le fait `read.table` par défaut.

Le package `XLConnect` offre aussi des fonctions pour lire des fichiers EXCEL. Il est plus puissant que `readxl`. Il permet de lire dans des cases précises d'un fichier EXCEL, ainsi que d'écrire dans un fichier EXCEL. Cependant, ce package dépend d'un autre logiciel : Java. Il faut donc avoir une installation du logiciel Java sur son ordinateur pour utiliser ce package. Il faut en outre avoir la version de Java avec un nombre de bits correspondant au nombre de bits de la version de R utilisé. Afin d'optimiser la mémoire accessible au cours d'une session R, je conseille d'utiliser R 64-bits à ceux qui travaillent sous Windows 64-bits. Par contre, il est courant d'avoir seulement la version 32-bits de Java sur son ordinateur, même si le système d'exploitation de celui-ci fonctionne en 64-bits. C'est dû au fait que le logiciel Java est utilisé principalement par les navigateurs web, qui fonctionnent souvent en 32-bits, même sous un système d'exploitation 64-bits. Pour installer la version 64-bits de Java en plus de la version 32-bits, il faut télécharger un installateur pour cette version de Java sur la page web suivante :
https://www.java.com/fr/download/manual.jsp

Le package `readxl` est moins puissant, mais il a l'avantage de ne pas dépendre d'un autre logiciel.

Importation de données publiées sur internet

Le web est rempli de données de toutes sortes. Pour lire des données stockées dans un fichier téléchargeable sur une page web, il suffit de d'abord télécharger le fichier, puis le lire avec un des moyens présentés précédemment.

Il est aussi possible de donner comme argument `file` à une fonction de la famille `read.table` directement une URL (adresse web).

De plus, il est possible de lire directement dans une table HTML. La majorité des sites web utilisent le langage HTML. Ce truc donne donc facilement accès à un très grand nombre de données. C'est une forme de ce qui est appelé en anglais le « *web harvesting* » ou « *web scraping* ».

Une façon d'importer en R les données d'une table HTML serait de copier/coller le contenu de la table dans un fichier (texte ou autre), puis importer ce fichier en R. Cependant, cette procédure n'est pas automatique. Certains packages R permettent d'éviter l'étape intermédiaire du copier/coller. Voici comment lire des données directement à partir d'une table HTML avec le package `rvest`.

```
library(rvest)
url <- "http://pro.boxoffice.com/numbers/all_time"
doc <- read_html(url)
tables <- html_table(doc)
boxoffice <- tables[[1]]
str(boxoffice)
```

```
## 'data.frame':    1000 obs. of  4 variables:
##  $ X1: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ X2: chr  "Star Wars: The Force Awakens (Disney)" "Avatar (Fox)" "Titanic (Paramount)" "Jurassic W
##  $ X3: chr  "Dec 18, 2015" "Dec 18, 2009" "Dec 19, 1997" "Jun 12, 2015" ...
##  $ X4: chr  "$936,662,225" "$760,507,625" "$658,672,302" "$652,270,625" ...
```

La fonction `read_html` lit tout le code HTML d'une page web. La fonction `html_table` en extrait les tables et les converties en data frames. La plupart du temps, les données publiées sur internet sont sous forme de tableaux et se trouvent dans des tables HTML. Pour finir, il ne reste plus qu'à extraire la bonne table parmi toutes les tables lues.

Le data frame obtenu dans le code précédent, nommé `boxoffice`, contient les recettes des films ayant générés le plus de revenus dans les cinémas américains, telles que présentées sur la page web http://pro.boxoffice.com/numbers/all_time. Ce data frame pourrait être amélioré en renommant ses colonnes.

Mentionnons finalement qu'il existe de plus en plus de packages R pour aller chercher rapidement, à partir de R, des données ouvertes publiées sur des sites web. Par exemple, le package `twitteR` permet d'importer en R des messages publiés sur Twitter. Ce genre de possibilité ne sera pas illustrée ici, mais la page web suivante énumère des packages utiles pour aller chercher des données sur divers sites web :
https://cran.r-project.org/web/views/WebTechnologies.html.

Écriture dans des fichiers

En cours d'analyse, il n'est pas rare d'obtenir des résultats dont il est préférable de conserver une copie dans un fichier externe. Ce fichier peut facilement être partagé avec d'autres ou importé dans un autre logiciel pour la poursuite des analyses selon les besoins. Il faut alors demander à R de créer un fichier et d'écrire dans celui-ci.

Pour les exemples à venir, reprenons le data frame `dataEx`, auquel une colonne est ajoutée contenant la somme des 2 dés.

```
dataExS <- cbind(dataEx, Somme = dataEx$de1 + dataEx$de2)
dataExS
```

```
##    de1 de2 lanceur Somme
## 1    2   1     Luc     3
## 2    3   4     Luc     7
## 3    4   2    <NA>     6
```

```
## 4    1   3     Luc     4
## 5    2   5     Luc     7
## 6    3   4     Kim     7
## 7    5  NA     Kim    NA
## 8    6   2     Kim     8
## 9    5   5     Kim    10
## 10   4   3     Kim     7
```

Nommons ce nouveau jeu de données `dataExS` et voyons comment exporter son contenu.

Exportation de données dans un fichier texte

Le format de fichier le plus universel pour exporter des données est le format texte. Un grand nombre de logiciels peuvent lire ce format. Les fonctions de base en R pour écrire dans un fichier texte sont `write` et `write.table`. Nous verrons ici seulement comment utiliser `write.table`, qui est plus adaptée à l'exportation de jeux de données. Par exemple, exportons le contenu du data frame `dataExS` dans le fichier texte `dataExS.txt`

```
write.table(dataExS, file = "dataExS.txt")
```

Il faut spécifier à la fonction l'objet dans lequel se trouvent les données à exporter (typiquement une matrice ou un data frame), ainsi que le fichier dans lequel écrire les données. Tout comme son équivalent pour lire des données `read.table`, la fonction `write.table` possède plusieurs arguments permettant de contrôler notamment le symbole de décimale (`dec`), le symbole représentant les valeurs manquantes (`na`), le séparateur entre les différentes colonnes (`sep`), la présence ou non des noms des lignes ou des colonnes (`row.names` et `col.names`), etc.

Il existe aussi deux fonctions enveloppe à la fonction `write.table` pour les formats CSV, nommées `write.csv` et `write.csv2`.

Exportation de données dans un fichier JSON

Pour exporter les données dans un data frame en format JSON avec le package `jsonlite`, il faut d'abord transformer les données au format JSON avec la fonction `toJSON`. Ensuite, l'objet obtenu doit être écrit dans un fichier externe avec la fonction `write` du R de base.

```
dataExS_json <- toJSON(dataExS, pretty = TRUE)
write(dataExS_json, "dataExS.json")
```

Enregistrement d'objet(s) R dans un fichier externe

R propose aussi ses formats de fichiers pour enregistrer à la fois les données contenues dans un objet R et la structure de l'objet, donc des objets entiers.

Formats binaires

Une image de session est l'ensemble des objets dans l'environnement de travail d'une session R. Lors de la fermeture de R, il nous est proposé par défaut d'enregistrer cette image. La fonction `save.image` peut aussi être utilisée en tout temps pour enregistrer cette image.

```
save.image(file = "image.RData")
```

La fonction `save` permet pour sa part d'enregistrer un ou des objets en particulier, par exemple :

```
save(dataExS, de1, file = "deuxObj.rda")
save(dataExS, file = "dataExS.rda")
```

R offre aussi un format binaire alternatif pour enregistrer un seul objet, sans inclure le nom de l'objet dans le fichier (contrairement aux fonctions précédentes). Il s'agit du format .rds, obtenu avec la fonction `saveRDS`.

```
saveRDS(dataExS, file = "dataExS.rds")
```

L'utilisateur peut en réalité donner les extensions de son choix aux fichiers. Il est cependant recommandé d'utiliser les extensions suivantes :

- .RData ou .rda pour un objet créé avec les fonctions `save.image` ou `save`,
- .rds pour un fichier créé avec la fonction `saveRDS`.

Ainsi, il est plus facile de se souvenir du format exact du fichier et de la fonction à utiliser pour charger le fichier lors de sessions futures (voir plus loin).

Formats texte

Il est aussi possible d'enregistrer des objets R sous un format texte avec `dump` ou `dput`. La fonction `dump` permet d'enregistrer plus d'un objet et les noms des objets sont inclus dans le fichier (comme pour `save` et `save.image`). La fonction `dput` ne permet d'enregistrer qu'un seul objet et son nom n'est pas inclus dans l'objet (comme pour `saveRDS`). Cependant, selon la documentation de R, l'utilisation des formats binaires est plus efficace et fiable.

Écriture de sorties R dans un fichier externe

Finalement, il est possible d'envoyer les résultats de nos commandes dans un fichier texte externe plutôt que dans la console grâce aux fonctions `capture.output` et `sink`.

```
capture.output(dataExS, file = "dataExS.out")
```

```
sink("dataExS.out")
dataExS
colMeans(dataExS[, 1:2], na.rm = TRUE)
sink()
```

Chargement d'objet(s) R provenant d'un fichier externe

Formats binaires

Pour charger en R des objets R stockés dans un fichier au format .RData, .rda ou .rds, il faut utiliser la fonction

- `load` : pour les fichiers produits avec la fonction `save` ou `save.image` (format .RData ou .rda, peut contenir plus d'un objet, les noms des objets sont inclus dans le fichier) ;
- `readRDS` : pour les fichiers produits avec la fonction `saveRDS` (format .rds, ne peut contenir qu'un seul objet, le nom de l'objet n'est pas inclus dans le fichier).

Voici deux exemples d'appel de ces fonctions.

```
load("deuxObj.rda")
```

```
dataEx_copie <- readRDS("dataExS.rds")
```

La fonction `load` ajoute des objets dans l'environnement de travail. Si des objets portant les mêmes noms que ceux à importer sont déjà présents dans l'environnement de travail, ils sont remplacés sans même qu'un avertissement ne soit émis.

La fonction `readRDS` doit être utilisée avec une assignation pour stocker le résultat de l'importation dans un objet .

Rappel : Si le répertoire courant par défaut de R contient un fichier nommé `.RData`, les objets dans ce fichier seront automatiquement chargés en R lors d'une ouverture de session. Pour empêcher ce chargement automatique, il faut effacer le fichier `.RData` .

Références pertinentes

Documentation officielle de R :

- http://cran.r-project.org/doc/manuals/r-release/R-data.html

Bonnes sources d'information sur le web :

- https://www.datacamp.com/community/tutorials/r-data-import-tutorial
- https://en.wikibooks.org/wiki/R_Programming/Importing_and_exporting_data

Livre dédié à la manipulation de données en R :

- Spector, P. (2008). Data Manipulation with R. Springer, New York.

Pour aller plus loin en importation et exportation de données

- packages utiles pour aller chercher des données ouvertes publiées sur le web :
  - https://cran.r-project.org/web/views/WebTechnologies.html
- package `foreign`, `Hmisc` ou `haven` pour des jeux de données dans un format propre à un autre logiciel statistique (ex. formats SAS, SPSS, etc.) et autres formats :
  - http://www.statmethods.net/input/importingdata.html
  - http://www.statmethods.net/input/exportingdata.html
  - https://cran.r-project.org/web/packages/haven/index.html
- communication avec des bases de données relationnelles (ex. : utilisant SQL) :
  - http://www.statmethods.net/input/dbinterface.html
- utilisation de connexions pour accéder à des fichiers :
  - Chapitre 10 de Matloff, N. (2011). The Art of R Programming : A Tour of Statistical Software Design. No Starch Press.