Lecture et écriture dans des fichiers externes à partir de R

Sophie Baillargeon, Université Laval

2021-03-15

Table des matières

Note préliminaire : Lors de leur dernière mise à jour, ces notes ont été révisées en utilisant R version 4.0.3, le package ***`readr`*** *version 1.4.0, le package* ***`data.table`*** *version 1.13.6, le package* ***`jsonlite`*** *version 1.7.2, le package* ***`readxl`*** *version 1.3.1, le package* ***`rvest`*** *version 0.3.6 et le package* ***`openxlsx`*** *version 4.2.3. Pour d'autres versions, les informations peuvent différer.*

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. Nous traiterons ici seulement du cas pour lequel l'importation ou l'exportation consiste à lire ou écrire dans un fichier externe de format texte, JSON, EXCEL, HTML ou dans un format propre à R.

Pour apprendre comment lire ou écrire des données stockées dans une base de données, le lecteur est référé à la documentation d'un package permettant de communiquer avec la base de données qu'il utilise. La page web suivante répertorie plusieurs de ces packages : https://CRAN.R-project.org/view=Databases.

1 Accéder au système de fichiers

Afin d'accéder à des données à partir de R, il faut souvent aller les lire là où elles se trouvent. Si les données sont dans un fichier situé sur l'ordinateur local, dans un certain répertoire, il faut aller lire son contenu à cet endroit.

Dans toute commande R effectuant de la lecture ou de l'écriture dans un fichier, il faut identifier ce fichier. Il peut être identifié en utilisant son chemin d'accès complet (en anglais *full* ou *absolute path*), ou encore en utilisant un chemin d'accès relatif à un emplacement courant (en anglais *relative path*). 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 `data_ex.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\data_ex.txt`. Le chemin d'accès du fichier relatif au répertoire `C:\coursR` est pour sa part simplement le nom du fichier, soit `data_ex.txt`.

Remarques importantes

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 une nouvelle ligne, etc.). Dans un chemin d'accès en R, il faut plutôt séparer les composantes par le caractère `/`, ou encore par `\\`.

De plus, dans une commande R, un chemin d'accès doit toujours être présenté sous forme de chaîne de caractères, donc être encadré de guillemets.

Ainsi, dans une commande R, le chemin d'accès `C:\coursR\data_ex.txt` devrait être écrit ainsi : `"C:/coursR/data_ex.txt"`. Ce chemin pourrait aussi être produit avec la fonction `file.path` comme suit.

```
file.path("C:", "coursR", "data_ex.txt")
```

```
## [1] "C:/coursR/data_ex.txt"
```

1.1 Répertoire de travail

Lorsqu'un fichier est identifié par un chemin d'accès relatif dans une commande, R doit faire une supposition quant au répertoire à partir duquel chercher le fichier. C'est justement l'utilité du répertoire de travail (ou répertoire courant, en anglais *working directory*). 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 de travail d'une session R, il suffit de soumettre la commande suivante :

```
getwd()
```

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

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

Si le répertoire de travail avait été fixé à `"C:"`, le chemin d'accès relatif du fichier aurait été `"coursR/data_ex.txt"`.

Symboles reconnus dans les chemins d'accès

Dans les chemins d'accès à des fichiers en R, les symboles `.`, `..` et `~` ont les significations usuelles suivantes :

- `.` : répertoire de travail courant,
- `..` : répertoire parent du répertoire de travail courant,
- `~` : répertoire de travail par défaut (en anglais *home directory*).

Ainsi, si le répertoire de travail était `"C:/coursR"`, les chemins d'accès `"C:/coursR/data_ex.txt"`, `"../coursR/data_ex.txt"`, `"./data_ex.txt"` et `"data_ex.txt"` seraient tous équivalents.

1.2 Explorer le contenu du système de fichiers

R a donc accès au système de fichiers de l'ordinateur à partir duquel la session a été démarrée. Directement dans la console R, il est possible de voir le contenu d'un répertoire, de créer des fichiers et des répertoires, d'effacer des fichiers, etc. Pour de l'information à ce propos, la fiche d'aide nommée `files` (ouverte avec la commande `help(files)`) est un bon point de départ. Contentons-nous ici d'illustrer l'utilisation de la fonction `dir`, qui permet d'énumérer le contenu d'un répertoire (par défaut le répertoire de travail).

```
dir()
```

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

Note : Tous les fichiers de la liste ci-dessus sont disponibles dans le fichier `lecture_ecriture_r_2020.zip`. Il s'agit des fichiers utilisés à la section 2 dans les exemples de lecture de données.

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.

1.3 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 de travail. En ouvrant un projet RStudio, une nouvelle session R est démarrée ayant pour répertoire de travail le répertoire de base du projet.

---

2 Lecture de fichiers

Une analyse de données en R débute typiquement par l'importation de données. 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. Ces données se rapportent à une expérience fictive de lancers de dés.

```
de_1 <- c(2, 3, 4, 1, 2, 3, 5, 6, 5, 4)
de_2 <- c(1, 4, 2, 3, 5, 4, 6, 2, 5, 3)
lanceur <- rep(c("Luc", "Kim"), each = 5)
data_ex <- data.frame(de1 = de_1, de2 = de_2, lanceur = lanceur)
# Introduction de valeurs manquantes
data_ex$de2[7] <- NA
data_ex$lanceur[3] <- NA
# Affichage du date frame
data_ex
```

```
##    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
```

Avant de lire en R un fichier de données, il est utile de d'abord le visualiser afin de connaître la façon dont les données sont formatées. Cependant, il est important de ne pas modifier le fichier en le visualisant. Pour être absolument certain(e) de ne pas modifier le fichier par inadvertance, une bonne pratique est de faire une copie du fichier et de visualiser cette copie plutôt que le fichier lu en R.

2.1 Fichier texte

Le fichier `data_ex.txt` (partagé dans le fichier `lecture_ecriture_r_2021.zip` au haut de cette page afin que vous puissiez reproduire les exemples proposés) contient les données exemple sous un format texte. Le contenu du fichier 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.

Les fonctions les plus simplistes en R pour lire dans des fichiers texte externes sont `readLines` et `scan`. Cependant, ces fonctions ne sont pas les meilleurs outils pour la lecture de tableaux de données.

Comme son nom l'indique, la fonction `readLines` lit un fichier texte ligne par ligne. Par exemple, voici ce que nous obtenons en lisant le contenu du fichier `data_ex.txt` avec la fonction `readLines`.

```
lecture_readLines <- readLines("C:/coursR/data_ex.txt")
lecture_readLines
```

```
##  [1] "de1\tde2\tlanceur" "2\t1\tLuc"         "3\t4\tLuc"         "4\t2\t."
##  [5] "1\t3\tLuc"         "2\t5\tLuc"         "3\t4\tKim"         "5\t.\tKim"
##  [9] "6\t2\tKim"         "5\t5\tKim"         "4\t3\tKim"
```

Le résultat est un vecteur contenant un élément par ligne du fichier lu. Pour chaque ligne, tout son contenu a été laissé dans une seule chaîne de caractère. Nous pouvons constater que le caractère séparant les champs dans le fichier `data_ex.txt` est `\t`, ce qui représente une tabulation.

La fonction `scan` permet quant à elle de lire des données en les considérant toutes du même type et stocke les données lues dans un vecteur ou une liste. Voici le résultat de la lecture du fichier `data_ex.txt` avec `scan`.

```
lecture_scan <- scan("C:/coursR/data_ex.txt", what = "character")
lecture_scan
```

```
##  [1] "de1"     "de2"     "lanceur" "2"       "1"       "Luc"     "3"       "4"
##  [9] "Luc"     "4"       "2"       "."       "1"       "3"       "Luc"     "2"
## [17] "5"       "Luc"     "3"       "4"       "Kim"     "5"       "."       "Kim"
## [25] "6"       "2"       "Kim"     "5"       "5"       "Kim"     "4"       "3"
## [33] "Kim"
```

Par défaut, `scan` suppose que le caractère séparant les champs dans le fichier lu est un « blanc », soit un ou plusieurs espaces, tabulations, fins de ligne ou retours de chariot. Ce comportement par défaut a bien convenu au fichier `data_ex.txt`. Il resterait cependant encore du travail à faire pour mettre ces données sous un format facilitant leur utilisation en R, par exemple un data frame utilisant les 3 premières valeurs comme noms de variables.

Pour l'importation d'un tableau de données provenant d'un fichier texte, la meilleure fonction offerte dans le R de base est plutôt `read.table` du package `utils` (chargé par défaut à l'ouverture d'une session R). Par exemple, lisons les données contenues dans le fichier `data_ex.txt` avec `read.table`.

```
data_ex_txt <- read.table("C:/coursR/data_ex.txt")
```

Le seul argument obligatoire dans la fonction `read.table` est le chemin d'accès au fichier à lire. Ici (comme dans les exemples précédents), il n'aurait pas été nécessaire de spécifier le chemin d'accès complet du fichier puisque ce fichier se trouve dans le répertoire de travail 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 `read.table`). 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.

```
data_ex_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(data_ex_txt)
```

```
## 'data.frame':    11 obs. of  3 variables:
##  $ V1: chr  "de1" "2" "3" "4" ...
##  $ V2: chr  "de2" "1" "4" "2" ...
##  $ V3: chr  "lanceur" "Luc" "Luc" "." ...
```

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.

```
data_ex_txt <- read.table("data_ex.txt", header = TRUE)
str(data_ex_txt)
```

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

La première variable, `de1`, a été correctement lu, mais il y a un problème avec la deuxième variable, `de2`. Elle contient des données de type caractères, alors qu'il s'agit d'une variable numérique (le 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.

```
data_ex_txt <- read.table("data_ex.txt", header = TRUE, na.strings = ".")
str(data_ex_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 réglé le problème de lecture de la deuxième colonne. Attardons-nous maintenant à la dernière colonne. Elle est enregistrée dans un vecteur de chaînes de caractères. Si nous avions préféré que `read.table` transforme les colonnes contenant des valeurs caractères en facteur, il aurait fallu assigner la valeur `TRUE` à l'argument `stringsAsFactors`. Avant sa version 4.0.0, R avait ce comportement par défaut. Le nouveau comportement par défaut est cependant bien ce que nous désirons ici.

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

2.1.1 Arguments de la fonction `read.table`

Voici un résumé des arguments les plus utiles de la fonction `read.table`. Chaque argument 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 champs (différentes valeurs) ?
  - Fournir ce caractère comme valeur à l'argument `sep`.

  - Par défaut `sep = ""`, ce qui signifie que le séparateur est un « blanc », soit un ou plusieurs espaces, tabulations, fins de ligne ou retours 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 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 = FALSE`.
  - Par défaut `stringsAsFactors = FALSE`.
- Est-ce que les données dans les différentes colonnes sont toutes du bon type ?
  - Si ce n'est pas le cas, les types désirés peuvent être spécifiés avec l'argument `colClasses`.
  - Par défaut R déduit des types de données en fonction du contenu des colonnes.

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`.

2.1.2 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. Deux séparateurs consécutifs signifient qu'une valeur est manquante. Le fichier `data_ex.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 `data_ex.csv` ?

```
data_ex_csv <- read.csv2("data_ex.csv")
str(data_ex_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: chr  "Luc" "Luc" "" "Luc" ...
```

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.

```
data_ex_csv <- read.csv2("data_ex.csv", na.strings = "")
str(data_ex_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: chr  "Luc" "Luc" NA "Luc" ...
```

2.1.3 Fichier texte volumineux

La fonction `read.table` (et ses fonctions enveloppe) arrive en principe à lire n'importe quel fichier texte de données. Elle peut cependant être lente pour lire des fichiers volumineux. Dans ce cas, il est plus pratique de se tourner vers une alternative plus rapide.

2.1.3.1 Package `readr`

Le package `readr` du `tidyverse` offre différentes fonctions pour lire des fichiers textes, selon leur format. Par exemple, il offre la fonction `read_csv2`, similaire à la fonction `read.csv2` du R de base. Voici un exemple d'utilisation de cette fonction.

```
library(readr)
data_ex_csv_readr <- read_csv2("data_ex.csv")
str(data_ex_csv_readr)
```

```
## spec_tbl_df [10 x 3] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ de1    : num [1:10] 2 3 4 1 2 3 5 6 5 4
##  $ de2    : num [1:10] 1 4 2 3 5 4 NA 2 5 3
##  $ lanceur: chr [1:10] "Luc" "Luc" NA "Luc" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   de1 = col_double(),
##   ..   de2 = col_double(),
##   ..   lanceur = col_character()
```

```
##   .. )
```

L'objet retourné par les fonctions de `readr` est un tibble. Remarquons que la fonction `read_csv2` a, par défaut, correctement interprété les valeurs manquantes et elle n'a pas stocké les données de type caractère dans un facteur. Des attributs ont été ajoutés au tibble pour spécifier les types des colonnes.

2.1.3.2 Package `data.table`

Bien que les fonctions du package `readr` tendent à être plus rapides que la fonction `read.table`, une autre fonction est capable d'être encore plus rapide : la fonction `fread` du package `data.table`. Voici un exemple d'utilisation de cette fonction.

```
library(data.table)
data_ex_csv_fread <- fread("data_ex.csv", na.strings = "")
str(data_ex_csv_fread)
```

```
## Classes 'data.table' and '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" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

L'objet retourné par la fonction `fread` est un data table. Tout comme la fonction `read.table`, la fonction `fread` possède un argument nommé `na.strings`, que nous avons du spécifier afin que la valeur manquante pour la variable catégorique soit bien interprétée. La fonction `fread` a, comme `read.table` et `read_csv2`, stocké par défaut dans un vecteur les données de types caractères.

2.2 Format JSON

Ce ne sont pas toutes les données qui conviennent au format « tableau de données » avec des observations en lignes 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)
data_ex_json <- fromJSON("data_ex.json")
str(data_ex_json)
```

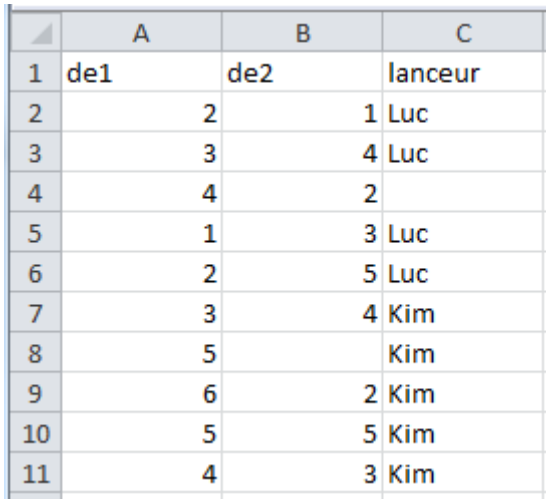
```
## 'data.frame':    10 obs. of  3 variables:
##  $ de1    : int  2 3 4 1 2 3 5 6 5 4
##  $ de2    : int  1 4 2 3 5 4 NA 2 5 3
##  $ lanceur: chr  "Luc" "Luc" NA "Luc" ...
```

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

2.3 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 quelques packages R offrant des fonctions pour lire directement dans un

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



	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

Nous pourrions procéder comme suit.

```
library(readxl)
data_ex_xlsx <- read_excel("data_ex.xlsx")
str(data_ex_xlsx)
```

```
## tibble [10 x 3] (S3: tbl_df/tbl/data.frame)
##  $ de1    : num [1:10] 2 3 4 1 2 3 5 6 5 4
##  $ de2    : num [1:10] 1 4 2 3 5 4 NA 2 5 3
##  $ lanceur: chr [1:10] "Luc" "Luc" NA "Luc" ...
```

Un des grands avantages de `readxl` comparativement à d'autres packages offrant des fonctions pour lire des fichiers EXCEL est qu'il ne dépend pas d'un autre logiciel. Plusieurs de ces packages requièrent une installation du logiciel Java.

2.4 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 de 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).

La plupart des données disponibles sur le web ne sont cependant pas distribuées dans un fichier. Souvent, elles sont simplement présentées sous forme de tableaux dans des tables HTML. 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 d'importer ce fichier en R. Si la table HTML en question provient d'une page sur un Wiki, tel que Wikipédia, le site web https://wikitable2csv.ggor.de/ facilite même cette tâche en créant des fichiers au format CSV contenant les données dans les tables HTML.

Ces façons de procéder ne sont cependant pas automatiques. Certains packages R permettent d'éviter les étapes intermédiaires manuelles et permettent de lire directement dans une table HTML. Il est ainsi facile d'accéder à un très grand nombre de données en R. C'est une forme de ce qui est appelé en anglais le « *web harvesting* » ou « *web scraping* ».

Voici un exemple de lecture de données directement à partir d'une table HTML avec le package `rvest`.

```
library(rvest)
url <- "https://en.wikipedia.org/wiki/List_of_highest-grossing_films"
```

```
page <- read_html(url)
tables <- html_table(page, fill = TRUE)
boxoffice <- tables[[1]]
str(boxoffice)
```

```
## 'data.frame':    50 obs. of  6 variables:
##  $ Rank           : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Peak           : chr  "1" "1" "1" "3" ...
##  $ Title          : chr  "Avatar" "Avengers: Endgame" "Titanic" "Star Wars: The Force Awakens" ...
##  $ Worldwide gross: chr  "$2,810,779,794" "$2,797,501,328" "$2,194,439,542" "$2,068,223,624" ...
##  $ Year           : int  2009 2019 1997 2015 2018 2015 2019 2012 2015 2019 ...
##  $ Reference(s)   : chr  "[# 1][# 2]" "[# 3][# 4]" "[# 5][# 6]" "[# 7][# 8]" ...
```

La fonction `read_html` lit tout le code HTML d'une page web. Cette lecture fonctionnera évidemment à la condition d'être connecté à internet. La fonction `html_table` en extrait les tables et les converties en data frames. Pour finir, il ne reste plus qu'à extraire la table voulue 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 https://en.wikipedia.org/wiki/List_of_highest-grossing_films le 18 janvier 2021.

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 `gtrendsR` permet d'importer en R des données provenant de Google trends. 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.

2.5 Vérification de l'importation

Une leçon importante à retenir de tous les exemples d'importation présentés dans ce document 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.

Voici une liste de vérifications qu'il est toujours bon d'effectuer après une importation en R, dans un data frame, de données sous forme de tableau avec des observations en lignes et des variables en colonnes.

- Est-ce que le nombre de variables (colonnes) dans le data frame est le même que le nombre de variables dans le fichier ?
- Est-ce que le nombre d'observations (lignes) dans le data frame est le même que le nombre d'observations dans le fichier ?
- Est-ce que, pour chacune des variables du jeu de données, le type de données dans le data frame correspond au type de données dans le fichier ?
- Si le jeu de données n'est pas de très petite taille, il est impossible de vérifier une à une toutes les observations. Cependant, pour une observation (ligne) sélectionnée au hasard (ou plus d'une si vous désirez être encore plus rigoureux), est-ce que les valeurs de cette observation dans le data frame sont identiques aux valeurs de cette observation dans le fichier ?
- Est-ce que les données manquantes sont bien représentées dans data frame par la constante `NA`, et ce, dans les colonnes de tout type ?

Si vous répondez oui à toutes ces questions, votre importation de données a de bonnes chances d'avoir été effectuée correctement.

---

3 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.

Exporter les données stockées dans un objet R peut aussi être utile simplement pour aider à les visualiser. Il est vrai que R offre la possibilité de voir le contenu de matrices ou de data frames dans un chiffrier ressemblant à une feuille EXCEL. Ce chiffrier peut être ouvert en appelant la fonction `View` ou en cliquant sur un objet dans la sous-fenêtre *Environment* de RStudio. Cependant, pour certaines structures de données non représentables sous forme de tableau, exporter les données dans un fichier externe demeure la meilleure option pour les visualiser.

Pour les exemples à venir, reprenons le data frame `data_ex` et voyons comment exporter son contenu.

3.1 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 `writeLines`, `write` et `write.table`. Les fonctions `writeLines` et `write` sont l'équivalent de `readLines` et `scan`, respectivement, pour l'écriture dans des fichiers au lieu de la lecture. Pour réaliser une tâche d'exportation de jeux de données, la fonction `write.table` est la mieux adaptée des trois. Par exemple, exportons le contenu du data frame `data_ex` dans le fichier texte `data_ex_export.txt`

```
write.table(data_ex, file = "data_ex_export.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 `read.table`, la fonction `write.table` possède plusieurs arguments permettant de contrôler le formatage des données, notamment le symbole de décimale (`dec`), le symbole représentant les valeurs manquantes (`na`), le séparateur entre les champs (`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`.

3.2 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.

```
data_ex_json <- toJSON(data_ex, pretty = TRUE)
write(data_ex_json, "data_ex_export.json")
```

3.3 Fichier EXCEL

Le package `openxlsx` offre des fonctions pour exporter des données stockées dans un objet R dans un fichier EXCEL. Par exemple, le contenu du data frame `data_ex` peut être écrit dans un fichier EXCEL avec la fonction `write.xlsx` comme suit.

```
library(openxlsx)
write.xlsx(data_ex, file = "data_ex_export.xlsx")
```

3.4 Enregistrement d'objets R dans un fichier externe

R propose aussi ses formats de fichiers pour enregistrer des objets entiers, en conservant à la fois les données contenues dans les objets et les structures des objets.

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(data_ex, de_1, file = "deux_obj.rda")
save(data_ex, file = "data_ex.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(data_ex, file = "data_ex.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.

3.5 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(summary(data_ex), file = "data_ex_summary.out")
```

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

---

4 Chargement d'objets 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) ; la fonction `readRDS` doit être utilisée avec une assignation pour stocker le résultat de l'importation dans un objet.

Voici deux exemples d'appel de ces fonctions.

```
load("deux_obj.rda")
```

```
data_ex_copie <- readRDS("data_ex.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. Avec cette fonction, il y a donc un risque d'écraser sans s'en rendre compte des objets présents dans notre environnement de travail.

Rappel : Si le répertoire de travail 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 ce fichier* `.RData` .

---

5 Résumé

Répertoire de travail et écriture de chemins d'accès à des fichiers

Répertoire de travail = emplacement par défaut dans le système de fichier de notre ordinateur utilisé dans la session R (emplacement supposé par R lorsque cette information n'est pas spécifiée)

- `getwd` : affiche le chemin d'accès au répertoire de travail,
- `setwd` : permet de changer de répertoire de travail.

Règles à respecter dans l'écriture de chemins d'accès à des fichiers :

- **pas de `\`** : seuls `/` et `\\` sont acceptés pour séparer les éléments d'un chemin d'accès ;
- toujours inclure l'**extension** du fichier ;
- encadrer le chemin de **guillemets**.

Exemple :

```
"C:/coursR/data_ex.txt"
```

Lecture (importation) dans des fichiers

Format à lire	Fonctions de lecture
texte	`read.table` (`.csv`, `.csv2`, `.delim`, `.delim2`)
	`read_table` (`_csv`, `_delim`, etc. ; package `readr`)
	`fread` (package `data.table`)
JSON	`fromJSON` (package `jsonlite`)
EXCEL	`read_excel` (package `readxl`)
table HTML	`read_html` + `html_table` (package `rvest`)

Des fonctions R existent pour importer des données sous plusieurs autres formats (voir références dans les notes). → non couvert dans le cours

À garder en tête : Il est conseillé de toujours vérifier que les données importées sont les bonnes et sous le bon format. Si ce n'est pas le cas, la commande d'importation de données doit être corrigée.

Écriture (exportation) dans des fichiers

Élément(s) à exporter	Fonction d'écriture
données et (optionnel) métadonnées	format texte (recommandé) : `write.table` (`.csv`, `.csv2`) format JSON : `toJSON` (package `jsonlite`) + `write` format EXCEL : `write.xlsx` (package `openxlsx`)
+ structure complète (objet R intact)	`save.image`, `save` (objet(s) + leur(s) nom(s)) `saveRDS` (un seul objet, sans son nom)
sorties R	`capture.output`, `sink`

Chargement d'objets R provenant d'un fichier externe

Fonction pour charger en R des objets stockés dans un fichier externe :

- fichier créé avec `save.image` ou `save` (typiquement extension .RData ou .rda) : `load` ;
- fichier créé avec `saveRDS` : `readRDS`, accompagnée d'une assignation à un nom pour enregistrer l'objet dans l'environnement de travail.

Références

Documentation officielle de R :

- R Core Team (2020). *R Data Import/Export*. R version 4.0.3. URL http://cran.r-project.org/doc/manuals/r-release/R-data.html

Documentation des packages utilisés :

- package `readr` :
  - page web du package sur le CRAN : https://CRAN.R-project.org/package=readr
  - documentation du package : https://readr.tidyverse.org/
  - chapitre 11 du livre : Grolemund, G. et Wickham, H. (2016). *R for Data Science*. O'Reilly Media, Inc. : https://r4ds.had.co.nz/data-import.html
- package `data.table` :
  - page web du package sur le CRAN : https://CRAN.R-project.org/package=data.table
  - documentation du package : https://rdatatable.gitlab.io/data.table/
- package `jsonlite` :
  - page web du package sur le CRAN : https://CRAN.R-project.org/package=jsonlite
- package `readxl` :
  - page web du package sur le CRAN : https://CRAN.R-project.org/package=readxl
  - documentation du package : https://readxl.tidyverse.org/
- package `rvest` :
  - page web du package sur le CRAN : https://CRAN.R-project.org/package=rvest
  - documentation du package : https://rvest.tidyverse.org/
- package `openxlsx` :
  - page web du package sur le CRAN : https://CRAN.R-project.org/package=openxlsx
  - documentation du package : https://ycphs.github.io/openxlsx/index.html

Autres 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
- https://themockup.blog/posts/2020-12-13-extracting-json-from-websites-and-public-apis-with-r/

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/view=WebTechnologies
- 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/package=haven
- communication avec des bases de données :
  - https://CRAN.R-project.org/view=Databases
  - https://db.rstudio.com/
  - https://datacarpentry.org/R-ecology-lesson/05-r-and-databases.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.