Les objets (partie 2)

Thibault LAURENT

21 octobre 2022

Contents

Ce document a été généré directement depuis **RStudio** en utilisant l'outil Markdown. La version .pdf se trouve ici.

Rappel :

Le répertoire de travail utilisé dans cette session (à vous de le modifier selon votre ordinateur):

```
setwd("Z:/m2_foad/cours_r/chapitre2")
```

Vous devrez également installer les packages suivants :

```
install.packages(c("jsonlite",    # pour importer des fichiers JSON
                   "readxl",      # pour importer des fichiers EXCEL
                   "sas7bdat"     # pour importer des fichiers SAS
                   ))
```

1 Gestion des objets : Entrée - Sortie

Cette section est dédiée à la gestion des objets dans **R**. Son apprentissage doit vous permettre de vous familiariser avec les outils permettant de sauvegarder et de charger des objets. Par ailleurs, **R** étant avant tout un logiciel de traitement de données, un paragraphe sera consacré à la gestion des jeux de données (importation, exportation).

1.1 Enregistrement d'un fichier de codes *.R*

Un fichier de codes doit porter l'extension `.R` et ne contenir que du code **R** et des commentaires (avec le symbole #). On présente ici un extrait du fichier `chapitre2_partie1.R` téléchargeable ici :

```
# Ce fichier contient les lignes de commandes présentées dans le chapitre 2
# du cours d'introduction à R

# chargement des données
load(file("http://www.thibault.laurent.free.fr/cours/Ressource/diamants.RData"))
```

```
# Codes présentés dans la première section
# 1.1. Les vecteurs
# option 1
a.numeric <- c(1.2, 3.5, 5.4, 6.2, 8.1)
is.numeric(a.numeric)
# option 2
a.numeric <- numeric(5) # on crée un vecteur de numeric de taille 5
a.numeric[1] <- 1.2 # on affecte la valeur 1.2 au 1er élément du vecteur
a.numeric[2] <- 3.5 # etc.
a.numeric[3] <- 5.4
a.numeric[4] <- 6.2

# etc.
```

Lorsque vous sauvegardez ce fichier depuis **RStudio**, il vous suffit de faire `"File < Save as < chapitre2_partie1.R"`. Penser à le faire régulièrement au cours d'une session.

Dès que vous ouvrirez une nouvelle session de **RStudio**, aucun des objets que vous aurez créé dans une session précédente n'aura été conservé. Une façon d'exécuter tout le code contenu dans `chapitre2_partie1.R` est d'utiliser la fonction `source()` qui exécute ligne par ligne les commandes du fichier qui ne comprend que du code écrit en **R** :

```
source("Z:/m2_foad/cours_r/chapitre2/chapitre2_partie1.R")
```

ou alors directement :

```
source("chapitre2_partie1.R")
```

si vous avez au préalable défini votre répertoire courant comme étant `Z:/m2_foad/cours_r/chapitre2`.

Remarque : depuis **RStudio**, on peut cliquer sur le boutton "Source" depuis le menu.

Chaque utilisateur a sa façon de travailler. Plutôt que d'éxécuter toutes les instructions du fichier `chapitre2_partie1.R`, on peut vouloir ne récupérer que certains objets et pour cela on va procéder différemment.

1.2 Enregistrement et chargement des objets

L'enregistrement d'un objet dans **R** est très simple. Il se fait à l'aide de la fonction `save()` en précisant comme premier(s) argument(s), l'objet (ou les objets) à sauvegarder et avec l'argument `file=` le nom du fichier (avec une extension `.RData`). Lorsque vous redémarrerez une session, il suffira alors d'utiliser la fonction `load()` en précisant l'adresse complète du fichier à charger.

Reprenons l'objet `don` dans la 1ère partie du chapitre 2. Pour le créer, on avait du exécuter toutes les commandes suivantes :

```
age <- c(20, 21, 20, 25, 29, 22)
taille <- c(165, 155, 150, 170, 175, 180)
sexe <- c("F", "F", "F", "M", "M", "M")
don <- data.frame(age, taille, sexe)
row.names(don) <- c("sonia", "maud", "iris", "mathieu", "amin", "gregory")
don$diplome <- c("DU", "M2", "M2", "DU", "DU", "M2")
don <- cbind(don, pays = c("FR", "FR", "SNG", "CAM", "HAI", "BF"))
don <- rbind(don, c(21, 180, "F", "DU", "FR"))
don2 <- data.frame(age = c(20, 21), taille = c(180, 175), sexe = c("M", "F"),
                   diplome = c("DU", "DU"), pays = c("FR", "ESP"))
row.names(don2) <- c("pierre", "sonia")
don <- rbind(don, don2)
```

```
don3 <- data.frame(note_algebre = c(18, 10, 8, 15, 20, 5, 17, 12, 8),
                   nom = c("sonia1", "pierre", "7", "gregory", "amin",
                           "mathieu", "iris", "maud", "sonia"))
don <- merge(don, don3, by.x = "row.names", by.y = "nom")
```

Si dans une nouvelle session, on souhaite effectuer des statistiques sur l'objet final `don`, les étapes intermédiaires ne sont plus nécessaires. Aussi, plutôt que d'exécuter à chaque fois l'ensemble de ces commandes, on aimerait pourvoir charger directement l'objet `don`. Pour cela, il faut d'abord le sauvegarder.

La sauvegarde de cet objet s'effectue par l'instruction :

```
save(don, file = "donnees_don.RData")
```

En l'état, le fichier sera enregistré dans le répertoire de travail courant. Si vous souhaitez l'enregistrer autre part, précisez un nouveau répertoire de travail avant la sauvegarde (avec la fonction `setwd()`) ou indiquez le chemin d'accès avant le nom du fichier. Par exemple :

```
save(don, file="C:/autre_repertoire/donnees_don.RData").
```

Lors d'une nouvelle session, l'instruction :

```
setwd("Z:/m2_foad/cours_r/chapitre2")
load("donnees_don.RData") # ou alors
# load("Z:/m2_foad/cours_r/chapitre2/donnees_don.RData")
```

chargera l'objet `don`.

Une autre manoeuvre consiste à sauvegarder la totalité des objets créés lors d'une session par la commande :

```
save.image(file = "chapitre2_objets.RData")
```

et de charger cette dernière lors d'une nouvelle session :

```
setwd("Z:/m2_foad/cours_r/chapitre2")
load("chapitre2_objets.RData")
```

Ce procédé peut être utile lorsque l'on souhaite sauvegarder un nombre important d'objets. On notera que lorsque vous quittez une session **R**, vous avez une fenêtre qui s'affiche en disant : `Save workspace image`. Si vous cliquez sur "oui", cela revient à exécuter la commande `save.image()`. Il peut cependant avoir l'inconvénient (dans le cas où l'environnement de travail est celui par défaut), de charger à l'ouverture de la nouvelle session plusieurs objets `parasites` inutiles pour l'utilisateur, d'où l'intérêt d'utiliser parfois la fonction `ls()` pour afficher les objets créés dans l'espace de travail et la fonction `rm()` (`remove`) qui permet de supprimer les objets qu'on n'utilise plus. Par exemple, on souhaite créer ici deux vecteurs `x` et `y`. La fonction `rnorm()` permet de simuler `n` valeurs distribuées selon une loi normale de moyenne `mean` et d'écart-type `sd`.

```
eps <- rnorm(n = 7, mean = 0, sd = 1)
a <- 3
b <- 4
x <- c(18, 17, 19, 20, 15, 19, 20)
y <- a * x + b + eps
rm(a, b, eps)
ls()
```

```
## [1] "age"    "don"    "don2"   "don3"   "sexe"   "taille" "x"      "y"
```

Ici, on a supprimé les objets `a`, `b` et `eps` (objets intermédiaires utilisés pour simuler le vecteur `y`) une fois qu'on n'en avait plus besoin. Pour supprimer tous les objets d'un seul coup, on peut utiliser la commande suivante :

```
rm(list = ls())
```

Remarque : lorsqu'au début du cours, on vous a demandé d'éxécuter la commande suivante :

```
load(file("http://www.thibault.laurent.free.fr/cours/Ressource/diamants.RData"))
```

vous avez ainsi chargé un fichier `".RData"` dans votre environnement courant. Nous avons utilisé la fonction `file()` pour préciser qu'il fallait aller cherché ce fichier sur internet. Si pour une raison ou une autre, vous risquez de travailler sans accès à internet, il est donc conseillé de sauvegarder le jeu de données `diamants` dans un répertoire courant de votre machine :

```
load(file("http://www.thibault.laurent.free.fr/cours/Ressource/diamants.RData"))
save(diamants, file = "donnees_diamants.RData")
```

Un fichier au format `.RData` n'est donc pas un fichier contenant des données à proprement dites (comme des fichiers `.txt`). Il s'agit d'un format propre à **R** et si vous essayez de l'ouvrir avec un bloc-note, vous ne pourrez pas lire son contenu. La section suivante a pour objectif de montrer comment on importe des données issues de fichiers de différents formats.

2 Importation et Exportation de fichiers de données

Il est très rare, lors d'une analyse statistique, que le jeu de données sur lequel est basée cette étude soit directement utilisable. Dans la grande majorité des cas, un "nettoyage" et/ou une fusion de plusieurs tables s'imposent. **R** peut très bien remplir cette tâche et pour des utilisateurs familiers avec d'autres outils, on retrouvera sous **R** des commandes identiques à **Perl** ou **SQL**. Toutefois, l'utilisation de ces outils est non triviale pour des non-experts en SGBD (système de gestion de bases de données).

Dans ce paragraphe, on va s'intéresser aux outils permettant d'importer des fichiers de données autres que ceux dont le format est reconnu par le logiciel. D'autre part, toutes les personnes ne travaillant pas avec le même logiciel, il peut être utile de connaître les fonctions permettant de "rapatrier" des données enregistrées depuis d'autres logiciels statistiques. C'est ce que nous verrons dans un deuxième temps.

2.1 Importation/Exportation de fichier-texte

Les fonctions permettant de lire des données dans un fichier texte sont nombreuses et efficaces dans **R**. Ce type de fichiers porte en général l'extension ".txt" et ".csv". On notera que les fichiers provenant de **Excel** (".xls" ou ".xlxs") ne rentrent pas dans ce type de fichiers (en effet, ils ne peuvent pas s'ouvrir directement depuis un éditeur de texte autre que ceux d'Office) et pour les importer dans **R**, il faudra utiliser des librairies spécifiques

Considérons les fichiers "dontxt_correct.txt" et "dontxt_problem.txt" : enregistrer-les dans votre répertoire de travail. Dans mon cas, je les ai enregistrés dans le sous répertoire de travail "Ressource". Pour y arriver, si vous utilisez Firefox, il suffit de faire un clique-droit et "Enregistrez la cible du lien sous". Cela doit être à peu près la même chose avec Internet Explorer et Chrome. Ces deux fichiers ont les mêmes données, mais celles-ci se présentent avec un système de codage différent.

2.1.1 La fonction `readLines()`

Avant d'importer un fichier de données sous **R**, il est nécessaire d'avoir une petite idée de la structure du fichier initial de données. Pour cela, on peut afficher ligne par ligne les éléments de n'importe quel fichier texte en utilisant la commande `readLines()`. Par exemple, pour savoir ce qu'il y a dans les deux premières lignes (option `n = 3`) du fichier texte `"dontxt_problem.txt"`, on procède ainsi :

```
readLines("Ressource/dontxt_correct.txt", n = 3)
```

```
## [1] "\"VER\"\t\"JA1\"\t\"RO1\"\t\"TYJ\"\t\"TrH\"\t\"IMP\""
## [2] "\"1\"\t 565\t 4\t113.00000\t\"Sem\"\t\"7h30-8h30\"\t\"0\""
```

```
## [3] "\"2\"\t 401\t 6\t 57.28571\t\"Sem\"\t\"7h30-8h30\"\t\"2\""
```

Ceci nous permet de voir que :

- la première ligne du fichier contient le nom des variables
- la première colonne contient le nom (ici le numéro) des lignes
- les chaînes de caractères sont délimitées par le symbole " "
- les colonnes sont espacées par une tabulation
- le point délimite les nombres décimaux.

On applique de nouveau cette fonction sur le second jeu de données :

```
readLines("Ressource/dontxt_problem.txt", n = 2)
```

```
## [1] "RowNames\tVER\tJA1\tRO1\tTYJ\tTrH\tIMP"
## [2] "1\t565,00\t4,00\t113,00\tSem\t7h30-8h30\t0"
```

Ceci nous permet de voir que la structure du fichier est un peu différente de la précédente : les colonnes sont espacées par une tabulation et que la virgule délimite les nombres décimaux. On ne pourra donc pas utiliser les mêmes options d'importation pour les deux fichiers.

Ainsi, la fonction `readLines()` permet d'avoir une idée de la façon dont a été construit le fichier de données.

Remarque: à ce stade si vous avez un message d'erreur, il y a de fortes chances pour cela soit du au fait que le chemin d'accès est incomplet ou alors mal recopié.

2.1.2 La fonction `read.table()`

Si vous disposez d'un fichier-texte (format `.txt`), vous pouvez le lire avec la fonction `read.table()`. Cette fonction (qui renvoie un objet de classe `data.frame`) admet un seul argument obligatoire : l'emplacement du fichier-texte à importer. Les autres arguments optionnels servent à préciser certaines caractéristiques du fichier (symbole utilisé pour séparer deux cellules, symbole utilisé pour les décimales, identification des valeurs manquantes, etc.). Voici les valeurs par défaut utilisés :

- `header = FALSE` : la première ligne du fichier à importer ne contient pas le nom des variables.
- `sep = ""` : le séparateur entre deux cellules est la tabulation TAB.
- `dec = "."` : pour séparer la partie entière de la partie décimale, le symbole utilisé est le point.

Bien évidemment, tous les fichiers-texte ne respectent pas nécessairement le même encodage et c'est pourquoi on est souvent à amener à jouer sur ces paramètres.

Commençons par importer le premier jeu de données `"dontxt_correct.txt"` sans modifier les arguments d'entrée :

```
dontxt_correct <- read.table("Ressource/dontxt_correct.txt")
```

Pour vérifier que l'importation s'est correctement effectuée, on peut utiliser la fonction `str()` (ainsi que la fonction `head()` pour afficher les premières lignes du fichier de données):

```
str(dontxt_correct)
```

```
## 'data.frame':    84 obs. of  6 variables:
##  $ VER: int  565 401 417 449 816 301 571 566 428 421 ...
##  $ JA1: int  4 6 7 3 8 7 6 6 4 9 ...
##  $ RO1: num  113 57.3 52.1 112.2 90.7 ...
##  $ TYJ: chr  "Sem" "Sem" "Sem" "Sem" ...
##  $ TrH: chr  "7h30-8h30" "7h30-8h30" "7h30-8h30" "7h30-8h30" ...
##  $ IMP: int  0 2 0 2 2 1 1 2 2 0 ...
```

```
head(dontxt_correct)
```

```
##   VER JA1       RO1 TYJ       TrH IMP
## 1 565   4 113.00000 Sem 7h30-8h30   0
## 2 401   6  57.28571 Sem 7h30-8h30   2
## 3 417   7  52.12500 Sem 7h30-8h30   0
## 4 449   3 112.25000 Sem 7h30-8h30   2
## 5 816   8  90.66667 Sem 7h30-8h30   2
## 6 301   7  37.62500 Sem 7h30-8h30   1
```

On vérifie qu'on ait le nombre de variables et d'observations attendu et aussi que les variables ont été codées dans le bon type, ce qui semble être le cas ici.

A présent, importons le deuxième fichier de données, toujours sans modifier les arguments d'entrée :

```
dontxt_problem <- read.table("Ressource/dontxt_problem.txt")
```

On utilise la fonction `str()` pour regarder le type des données et aussi la fonction `head()` pour afficher les premières lignes :

```
str(dontxt_problem)
```

```
## 'data.frame':    85 obs. of  7 variables:
##  $ V1: chr  "RowNames" "1" "2" "3" ...
##  $ V2: chr  "VER" "565,00" "401,00" "417,00" ...
##  $ V3: chr  "JA1" "4,00" "6,00" "7,00" ...
##  $ V4: chr  "RO1" "113,00" "57,29" "52,13" ...
##  $ V5: chr  "TYJ" "Sem" "Sem" "Sem" ...
##  $ V6: chr  "TrH" "7h30-8h30" "7h30-8h30" "7h30-8h30" ...
##  $ V7: chr  "IMP" "0" "2" "0" ...
```

```
head(dontxt_problem, 2)
```

```
##         V1     V2   V3     V4  V5        V6  V7
## 1 RowNames    VER  JA1    RO1 TYJ       TrH IMP
## 2        1 565,00 4,00 113,00 Sem 7h30-8h30   0
```

On se rend compte immédiatement que l'objet `dontxt_problem` n'est pas conforme. En effet, toutes les colonnes sont codées en `character` alors qu'il devrait y avoir des colonnes dont le type est `numeric`. On en déduit donc que les paramètres par défaut de la fonction `read.table()` ne sont pas adéquats pour ce jeu de données. Parmi les choses qui ne vont pas, la première ligne contenant le nom des variables est considérée comme une observation. Pour éviter cela, il faut indiquer que cette ligne contient le nom des variables. Ceci peut être fait par l'ajout de l'argument `header=TRUE`.

Le symbole qui permet de séparer la partie entière de la partie décimale est la virgule, qui n'est pas celui par défaut. Il est donc nécessaire de l'indiquer en utilisant l'option `dec=","`.

Finalement, pour importer correctement ce jeu de données, il suffisait de faire :

```
dontxt_clean <- read.table("Ressource/dontxt_problem.txt",
                           header = TRUE, dec = ",")
str(dontxt_clean)
```

```
## 'data.frame':    84 obs. of  7 variables:
##  $ RowNames: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ VER     : num  565 401 417 449 816 301 571 566 428 421 ...
##  $ JA1     : num  4 6 7 3 8 7 6 6 4 9 ...
##  $ RO1     : num  113 57.3 52.1 112.2 90.7 ...
##  $ TYJ     : chr  "Sem" "Sem" "Sem" "Sem" ...
```

```
##  $ TrH     : chr  "7h30-8h30" "7h30-8h30" "7h30-8h30" "7h30-8h30" ...
##  $ IMP     : int  0 2 0 2 2 1 1 2 2 0 ...
```

Remarque : pour la lecture du premier jeu de données, il n'était pas nécessaire d'ajouter l'option `header=TRUE`. En effet, la première ligne du fichier `"dontxt_correct.txt"` contient seulement une valeur de moins que le nombre de colonnes dans le fichier. Aussi, **R** a compris directement que la première ligne contenait le nom des variables et que la première colonne n'était pas une variable, mais correspondait au nom des individus. Ainsi, la première colonne du fichier n'apparaît pas sous forme d'une variable, mais comme étant l'identifiant des individus (l'attribut `row.names`). Pour afficher les identifiants des individus, on fait :

```
row.names(dontxt_correct)
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14" "15"
## [16] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28" "29" "30"
## [31] "31" "32" "33" "34" "35" "36" "37" "38" "39" "40" "41" "42" "43" "44" "45"
## [46] "46" "47" "48" "49" "50" "51" "52" "53" "54" "55" "56" "57" "58" "59" "60"
## [61] "61" "62" "63" "64" "65" "66" "67" "68" "69" "70" "71" "72" "73" "74" "75"
## [76] "76" "77" "78" "79" "80" "81" "82" "83" "84"
```

Dans le deuxième example, la colonne qui donne le nom des observations a été codée dans une variable à part entière. L'utilisateur a ainsi le choix de :

- donner un attribut `rownames` à chaque observation
- utiliser une variable comme identifiant

2.1.2.1 Autres arguments d'entrée

Parmi les autres arguments d'entrée de la fonction `read.table()` qu'il peut être important de connaître, il y a :

- `na.strings`, un `character` qui renseigne comment sont codées les valeurs manquantes dans le fichier de données (par défaut `na.strings = "NA"`),
- `nrows` qui indique combien de lignes il faut importer (par défaut `nrows = -1` ce qui signifie que toutes les lignes doivent être importées),
- `colClasses` qui un vecteur de `character` qui indique quel est le type des colonnes à importer. Dans l'exemple précédent, on aurait pu mettre `colClasses = c("integer", "numeric", "numeric", "numeric", "character", "character", "integer")`,
- `skip`, le nombre de lignes à sauter dans le fichier avant de commencer l'importation. Cette commande est à utiliser lorsque l'en-tête du fichier contient des informations de type métadonnées.
- `quote`, précise quel est le caractère utilisé pour encadrer une chaîne de caractères. Par défaut `quote = "\"'"`.

2.1.2.2 `Factor` ou `character` ?

La fonction `read.table()` contient un argument d'entrée qui permet à l'utilisateur de décider si les chaînes de caractère seront codées en `factor` ou `character`. Depuis la dernière grande mise à jour de **R** (passage à la version 4.0.0) les chaînes de caractère sont codées en `character`. Pour choisir un codage des chaînes de caractère en `factor`, on utilise l'option `stringsAsFactors = TRUE`.

2.1.3 Autres fonctions utiles

Il existe d'autres fonctions permettant de lire des données dans un fichier-texte. Elles sont toutes plus ou moins proches de `read.table()` car elles ne diffèrent que par le changement d'un ou quelques paramètres d'entrée. Parmi elles, on peut citer `scan()`, `read.csv()`, `read.delim()`, `read.fwf()`. Il est bon de les connaître (si leur utilisation est vraiment nécessaire), mais dans la grande majorité des cas, la fonction `read.table()` est suffisante pour arriver à ses fins.

2.1.3.1 Fonction `read.csv2()`

La fonction `read.csv2()` permet d'importer fichier qui a été enregistré depuis au format `".csv"` (en utilisant le "séparateur point-virgule' '). Beaucoup de fichiers de données ont été sauvegardées avec cette extension. C'est le cas de cette base de donnée, initiallement produite par Toulouse Metropole et accessible depuis ma page web.

```
link <- "http://www.thibault.laurent.free.fr/cours/R_intro/Ressource/"
don_csv <- read.csv2(file = paste0(link,
                "communes-de-toulouse-metropole.csv"))
head(don_csv)
```

```
##                Ville Deliberation.en.faveur.de.l.OpenData
## 1          Colomiers                                   OK
## 2 Gagnac-sur-Garonne
## 3            L'Union                                   OK
## 4        Saint-Alban
## 5         Saint-Jean
## 6       Aigrefeuille
##
## 1 https://data.toulouse-metropole.fr/api/v2/catalog/datasets/communes-de-toulouse-metropole/files/935
## 2 https://data.toulouse-metropole.fr/api/v2/catalog/datasets/communes-de-toulouse-metropole/files/2b9
## 3 https://data.toulouse-metropole.fr/api/v2/catalog/datasets/communes-de-toulouse-metropole/files/48a
## 4 https://data.toulouse-metropole.fr/api/v2/catalog/datasets/communes-de-toulouse-metropole/files/120
## 5 https://data.toulouse-metropole.fr/api/v2/catalog/datasets/communes-de-toulouse-metropole/files/b09
## 6 https://data.toulouse-metropole.fr/api/v2/catalog/datasets/communes-de-toulouse-metropole/files/6c3
##                            Adresse
## 1    http://www.ville-colomiers.fr/
## 2 http://www.gagnac-sur-garonne.fr/
## 3      http://www.mairie-lunion.fr/
## 4      http://www.saint-alban31.fr/
## 5   http://www.mairie-saintjean.fr/
## 6        http://www.aigrefeuille.fr
```

2.1.3.2 Fonction `fromJSON()`

Les fichiers de données sont de plus en plus sauvegardées dans le format **JSON** qui est aussi un format texte. C'est notamment le cas des données issues de l'Open Data. Pour importer ce type de données, on pourra faire appel à la fonction `fromJSON()` de la librairie **jsonlite**.

```
require("jsonlite")
```

```
## Le chargement a nécessité le package : jsonlite
```

```
elec <- "https://www.data.gouv.fr/fr/datasets/r/cae2bd1b-e682-4866-9eff-bd18ecb548da"
don.json <- fromJSON(elec)
```

2.1.4 Exportation de données

L'exportation de données vers un fichier-texte se fait au moyen de la fonction `write.table()`. Les deux arguments obligatoires sont le nom de l'objet à exporter et le nom du fichier à créer. Les autres arguments sont les mêmes que ceux de la fonction `read.table()`, i.e. qu'ils servent à définir différents codes. Pour continuer avec l'exemple ci-dessus, on enregistre la table `.txt` en donnant l'instruction :

```
write.table(dontxt_correct, "fichier-sortie.txt").
```

Il existe d'autre fonctions qui pemettent d'exporter un `data.frame` vers d'autres formats de données :

- `write.csv2()` pour exporter vers un fichier au format .csv
- `toJSON()` pour exporter vers un fichier au format JSON.

2.2 Importation/Exportation de fichiers issus d'autres logiciels

2.2.1 Fichiers Excel (".xls" ou ".xlsx")

Pour importer un fichier au format **Excel**, il y a deux possibilités. La première est de convertir les fichiers `.xls` et `.xlsx` au format `.csv` (avec le délimiteur point virgule) depuis Office et de les importer en utilisant les fonctions vues ci-dessus. Le logiciel **Excel** étant un logiciel propriétaire, il existe une alternative provenant du monde du libre LibreOffice qui permet de faire ce type de conversion.

La deuxième solution est d'utiliser une librairie spécialisée. Il en existe plusieurs et on recommande pour l'instant la librairie **readxl**. La fonction `read_xls()` permet ainsi d'importer directement des fichiers de données au format **Excel**

```
library("readxl")
f <- "https://www.insee.fr/fr/statistiques/fichier/3292622/dep31.xls"
download.file(f, destfile = paste0(getwd(), "/dep31.xls"))
```

Dans l'exemple ci-dessus, la fonction `download.file()` a permis de télécharger depuis un site extérieur un fichier de données sur le disque local. L'argument `destfile` indique le chemin où est sauvegardé le fichier.

Dans un deuxième temps, si on importe le fichiers de données sans préciser aucun argument, on va rencontrer un problème qui s'explique par le fait que les premières lignes du fichier **Excel** contiennent des Métadonnée:

```
don_xls <- readxl::read_xls("dep31.xls")
```

```
## New names:
## * `` -> `...2`
## * `` -> `...3`
## * `` -> `...4`
## * `` -> `...5`
## * `` -> `...6`
## * `` -> `...7`
## * `` -> `...8`
## * `` -> `...9`
```

Pour pallier ce problème, on ajoute l'argument `skip` qui permet de sauter les lignes du fichier qu'on ne souhaite pas importer :

```
don_xls <- readxl::read_xls("dep31.xls", skip = 7)
```

2.2.2 Importation de données d'un autre logiciel statistique (SAS/STATA/SPSS)

Un module a été spécialement conçu pour pouvoir importer des fichiers de données produit par d'autres logiciels statistiques. Il s'agit de la librairie **foreign**, librairie intégrée à la version de base de **R**. Noter toutefois qu'il faudra la charger avant chaque utilisation par :

```
library("foreign")
```

Plusieurs systèmes statistiques sont pris en compte (**Minitab**, **SPSS**, **Stata**, ...), mais nous n'en verrons ici que quelques uns. Avant de commencer, il faut savoir qu'étant donné leur coût, il est très rare que plusieurs logiciels statistiques cohabitent. Cependant, il peut arriver que ce module soit utile si un travail est effectué à plusieurs dans différents environnements.

De plus, certaines revues statistiques comme CSBIGS ou Journal of Statistical Software éditent des articles de statistique appliquée. Pour pouvoir vérifier les résultats obtenus dans ces études, la publication des jeux de données est obligatoire et leur format dépend du logiciel qui a été utilisé par les chercheurs . Dans ce cas, la bibilothèque **foreign** peut être fort utile.

2.2.2.1 SAS Un fichier de données SAS porte l'extension `.sas7bdat`. Le package **sas7bdat** permet l'importation d'un jeu de données au format `.sas7bdat` à l'aide de la fonction `read.sas7bdat()`.

Si l'on prend la table de données baseball.sas7bdat et qu'on la copie dans le répertoire "Ressource", cela donne :

```
require("sas7bdat")
```

```
## Le chargement a nécessité le package : sas7bdat
```

```
don.sas <- read.sas7bdat("Ressource/baseball.sas7bdat")
head(don.sas)
```

```
##             name         team no_atbat no_hits no_home no_runs no_rbi no_bb
## 1  Aldrete, Mike SanFrancisco      216      54       2      27     25    33
## 2 Allanson, Andy    Cleveland      293      66       1      30     29    14
## 3    Almon, Bill   Pittsburgh      196      43       7      29     27    30
## 4 Anderson, Dave   LosAngeles      216      53       1      31     15    22
## 5    Armas, Tony       Boston      425     112      11      40     58    24
## 6    Ashby, Alan      Houston      315      81       7      24     38    39
##   yr_major cr_atbat cr_hits cr_home cr_runs cr_rbi cr_bb   league division
## 1        1      216      54       2      27     25    33 National     West
## 2        1      293      66       1      30     29    14 American     East
## 3       13     3231     825      36     376    290   238 National     East
## 4        4      926     210       9     118     69   114 National     West
## 5       11     4513    1134     224     542    727   230 American     East
## 6       14     3449     835      69     321    414   375 National     West
##   position no_outs no_assts no_error salary
## 1       1O     317       36        1     75
## 2        C     446       33       20    NaN
## 3       UT      80       45        8    240
## 4       3S      73      152       11    225
## 5       CF     247        4        8    NaN
## 6        C     632       43       10    475
```

2.2.2.2 STATA Si l'on considère les données contenues dans le fichier automiss.dta (extrait du site de Stata), l'importation se réalise avec la fonction `read.dta()` comme suit :

```
automiss <- read.dta("Ressource/automiss.dta")
head(automiss)
```

```
##            make price mpg rep78 headroom trunk weight length turn displacement
## 1   AMC Concord  4099  22    NA      2.5    11   2930    186   40          121
## 2     AMC Pacer  4749  17    NA      3.0    NA   3350    173   40          258
## 3    AMC Spirit  3799  22    NA      3.0    12   2640    168   35          121
## 4 Buick Century  4816  20     3      4.5    16   3250    196   40           NA
## 5 Buick Electra  7827  15     4      4.0    20   4080    222   NA          350
## 6 Buick LeSabre  5788  18     3      4.0    21   3670    218   43           NA
##   gear_ratio  foreign
## 1       3.58 Domestic
## 2       2.53 Domestic
## 3         NA Domestic
## 4       2.93 Domestic
## 5       2.41 Domestic
## 6       2.73 Domestic
```

2.2.2.3 SPSS SPSS, comme Stata, ne pose aucun problème d'importation. Pour la table donspss.sav, le code est :

```r
don.spss <- read.spss("Ressource/donspss.sav", to.data.frame = T)
head(don.spss)
```

```
##   ROWNAMES VER JA1       RO1 TYJ       TRH IMP
## 1        1  565   4 113.00000 Sem 7h30-8h30   0
## 2        2  401   6  57.28571 Sem 7h30-8h30   2
## 3        3  417   7  52.12500 Sem 7h30-8h30   0
## 4        4  449   3 112.25000 Sem 7h30-8h30   2
## 5        5  816   8  90.66667 Sem 7h30-8h30   2
## 6        6  301   7  37.62500 Sem 7h30-8h30   1
```