Importation et exportation des données

Malick SENE

2025-04-13

2. Importation et exportation des données

Les données peuvent provenir de sources diverses et se présenter sous forme brute, comme des fichiers texte, des feuilles Excel,des fichiers stata, spss ou encore des bases de données distantes... Le **tidyverse**, à travers ses packages spécialisés, met à disposition des outils pour importer différents types de fichiers. Nous allons voir les plus courants.

2.1.Import de fichiers texte

Les fichiers texte, notamment les formats CSV (Comma-Separated Values) et TSV (Tab-Separated Values), sont largement utilisés pour stocker des données sous forme de tableaux.Les fichiers CSV utilisent des virgules pour séparer les valeurs des colonnes, tandis que les fichiers TSV utilisent des tabulations pour effectuer cette séparation. Pour importer ces fichiers dans R, les fonctions read_csv() et read_tsv() du package readr sont utilisés.

```
# Exemple d'importation d'un fichier CSV
data_cvs <- read_csv("fichier.csv")

#Si le fichier vient d'Excel, avec des valeurs séparées par des points virgule, on utilise la fonction

data_csv2 <- read_csv2("fichier.csv")

# Exemple d'importation de fichiers TSV
data_tsv <- read_tsv("Données/ehcvm_individu_bfa2021.tsv")
```

Dans la même famille de fonctions que `read_csv()`, on trouve également : `read_delim()` qui permet d'importer des fichiers **délimités par un séparateur personnalisé**, précisé via l'argument `delim`.

Les principaux arguments communs aux fonctions `readr` sont :

- `col_names` : logique indiquant si la **première ligne contient les noms des colonnes** (valeur par défaut : `TRUE`).
- `col_types` : permet de **spécifier manuellement le type des colonnes**.

Il peut arriver, notamment sous Windows que l'importation de fichiers créés sur un autre système d'exploitation génère des problèmes d'encodage, notamment pour les **caractères accentués**. Dans ce cas, on peut utiliser l'argument `locale()` pour **spécifier manuellement l'encodage du fichier**.

```r
data <- read_csv("fichier.csv", locale = locale(encoding = "ISO-8859-1"))
```

Il peut arriver, notamment sous Windows, que l'**encodage des caractères accentués** (comme les lettres accentuées ou les caractères spéciaux)
ne soit pas correctement interprété lors de l'importation de fichiers texte dans R.

Ce problème survient souvent lorsque le fichier a été créé sur un système avec un **encodage différent** de celui du système actuel.
Par exemple, si vous êtes sous Mac ou Linux et que le fichier a été créé sous Windows, il est possible qu'il soit encodé en **ISO-8859-1** (aussi appelé "latin1").

Dans ce cas, vous pouvez **spécifier manuellement l'encodage** lors de l'importation à l'aide de l'argument `locale()`.

```r
data <- read_csv("fichier.csv", locale = locale(encoding = "ISO-8859-1"))
```

À l'inverse, si vous travaillez **sous Windows** et que les **accents ne s'affichent pas correctement** à l'importation,
il est probable que le fichier source soit encodé en **UTF-8** — un encodage courant sur Mac, Linux, ou généré par des plateformes web.

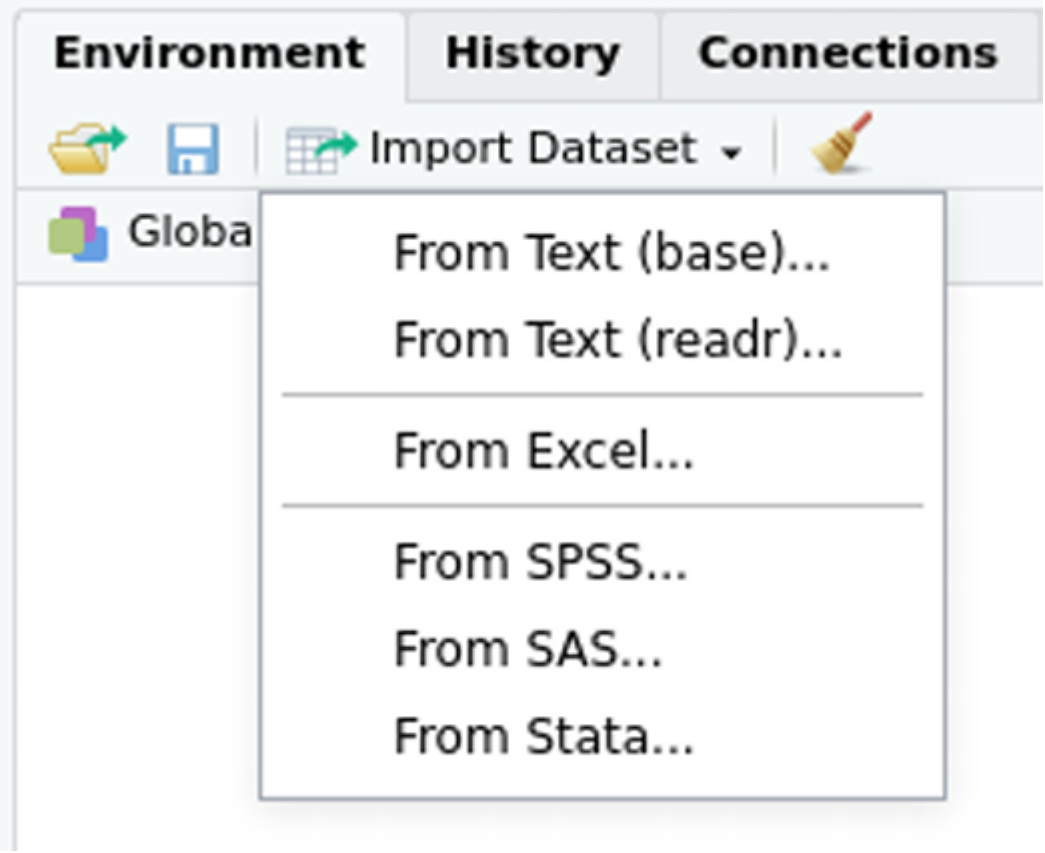
Dans ce cas, il est recommandé de **forcer l'encodage UTF-8** lors de l'importation à l'aide de l'argument `locale()`.

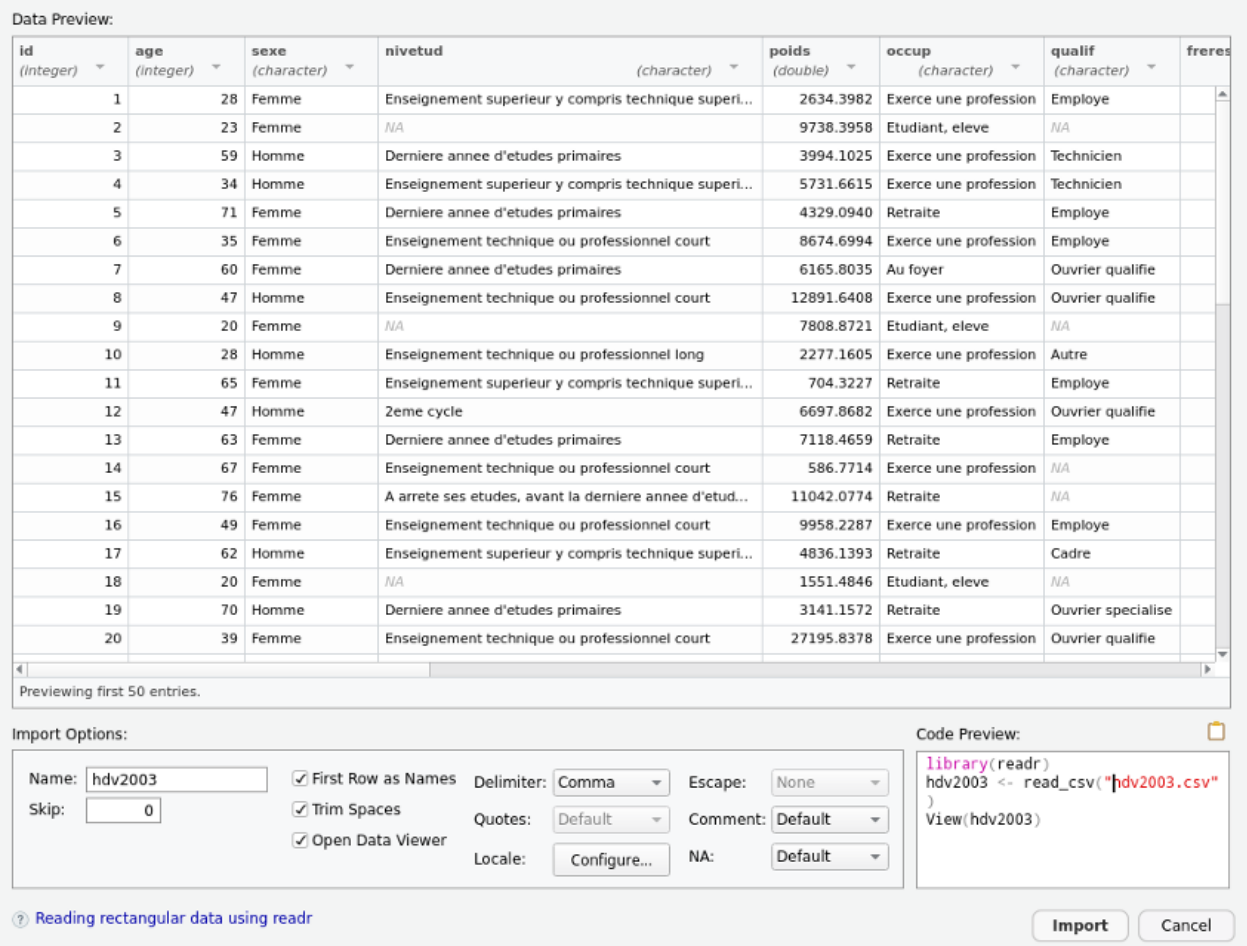
```r
data <- read_csv("fichier.csv", locale = locale(encoding = "UTF-8"))
```

2.1.1 Interface interactive d'import de fichiers

RStudio propose une **interface graphique conviviale** permettant d'importer un fichier de données. Pour cela :

1. Allez dans l'**onglet "Environment"** situé en haut à droite de la fenêtre RStudio.
2. Cliquez sur le bouton **"Import Dataset"**.
3. Sélectionnez le type de fichier à importer (CSV, Excel, SPSS, Stata, etc.).
4. Parcourez vos fichiers pour sélectionner le document à charger.
5. Un aperçu du fichier s'affiche. Vous pouvez alors :
    - Spécifier le séparateur,
    - Choisir l'encodage,
    - Indiquer si les noms de colonnes sont dans la première ligne,
    - Voir immédiatement un **résumé des données**.


Environment
History
Connections
Import Dataset
Globa
From Text (base)...
From Text (readr)...
From Excel...
From SPSS...
From SAS...
From Stata...



Important : une fois que l'import semble correct, ne cliquez pas sur le bouton Import. À la place,sélectionnez le code généré et copiez-le (ou cliquez sur l'icône en forme de presse papier) et choisissez Cancel.
Ensuite, collez le code dans votre script et exécutez-le (vous pouvez supprimer la ligne commençant par View).
Cette manière de faire permet "d'automatiser" l'importation des données, puisqu'à la prochaine ouverture du script, vous aurez juste à exécuter le code en question, sans repasser par l'interface d'import.

2.2 Import depuis un fichier Excel

Les fichiers Excel sont largement utilisés pour stocker des données, parfois réparties sur plusieurs feuilles avec des formats variés.
Le package **readxl** facilite leur importation dans R, en prenant en charge aussi bien les fichiers `.xls` que `.xlsx`,ce qui le rend particulièrement utile pour manipuler des données issues d'Excel de manière.

Une fois le package chargé, vous pouvez facilement lire une feuille spécifique d'un fichier Excel en utilisant la fonction read_excel(). Exemple:

```
library(readxl)
# Lire un fichier excel
data <- read_excel("fichier.xlsx")
# Lire une feuille spécifique d'un fichier Excel,si ce parametre n'est pas précisé, R importera la prem
data <- read_excel("fichier.xlsx", sheet = "Feuille2", range = "C1:F124")
```

Si vous ne connaissez pas les noms des feuilles dans un fichier Excel, vous pouvez obtenir une liste de toutes les feuilles d'un fichier à l'aide de la fonction excel_sheets() :

```
# Lister toutes les feuilles d'un fichier Excel
excel_sheets("fichier.xlsx")#Cela vous permet de savoir exactement quelles feuilles sont disponibles da
```

2.3 Import de fichiers SAS, SPSS et Stata

L'importation de fichiers de données issus de logiciels statistiques comme SAS, SPSS ou Stata peut être réalisée à l'aide du package **haven** de tidyverse.

- Pour les fichiers provenant de SAS, vous pouvez utiliser les fonctions `read_sas` ou `read_xpt`
- Pour les fichiers provenant de SPSS, vous pouvez utiliser `read_sav` ou `read_por`
- Pour les fichiers provenant de Stata, utilisez `read_dta`.

```
# SPSS
data_spss <- read_sav("fichier.sav")

# Stata
data_stata <- read_dta("fichier.dta")

# SAS
data_sas <- read_sas("fichier.sas7bdat")
```

2.4 Import de fichiers dBase

Avant de pouvoir importer des fichiers dBase dans R, il faut installer et charger le package foreign. Ce package permet de lire non seulement les fichiers dBase, mais aussi d'autres formats comme les fichiers SAS, SPSS, et bien d'autres. Une fois installé, vous pouvez utiliser la fonction read.dbf() pour importer vos fichiers dBase.

```
library(foreign)
data_dbf <- read.dbf("fichier.dbf")
```

La fonction `read.dbf` n'admet qu'un seul argument, `as.is`.
Si `as.is = FALSE` (valeur par défaut), les chaînes de caractères sont automatiquement converties en `factor` à l'importation.
Si `as.is = TRUE`, elles sont conservées telles quelles.

2.5 Connexion à des bases de données

Dans le cadre de l'analyse de données, il est courant de recourir à des bases de données relationnelles telles que MySQL, PostgreSQL ou SQLite pour stocker et gérer des informations volumineuses ou complexes. R facilite l'interaction avec ces bases grâce aux packages `DBI` (interface générique) et `RSQLite` (interface pour SQLite), permettant ainsi d'exécuter des requêtes, d'importer des données et d'intégrer ces opérations dans un flux analytique reproductible.

2.5.1. Connexion à une base de données SQLite

SQLite est une base de données relationnelle légère, intégrée dans des fichiers locaux. Elle est idéale pour des projets où une installation de serveur de base de données est inutile. Par exemple, dans le cadre de l'analyse de données en local, SQLite permet de manipuler facilement des fichiers .sqlite ou .db.

Pour se connecter à une base de données SQLite dans R, on utilise le package DBI pour établir la connexion, et RSQLite pour interagir spécifiquement avec les fichiers SQLite. Voici comment procéder :

```
#Installer le package RSQLite (une seule fois)
install.packages("RSQLite")

# Charger les packages nécessaires
library(DBI)        # DBI permet une interface standard avec toutes les bases de données
library(RSQLite)    # RSQLite permet de travailler spécifiquement avec des fichiers SQLite
```

a) Installation et chargement des packages nécessaires

```
# Une fois les packages chargés, la connexion à la base de données se fait avec la fonction dbConnect()

con <- dbConnect(RSQLite::SQLite(), dbname = "Données/northwind_small.sqlite")
```

b) Connexion à la base de données SQLite

c) Vérification des tables disponibles Une fois la connexion établie, vous pouvez lister toutes les tables présentes dans la base de données avec la fonction dbListTables(con). Cette commande renvoie un vecteur de caractères contenant les noms des tables présentes dans la base de données. Cela est particulièrement utile pour identifier les tables que vous pouvez manipuler.

```
# Afficher la liste des tables disponibles dans la base
dbListTables(con)
```

2.5.2 Lecture d'une table

Une fois que vous avez identifié la table à lire, vous pouvez l'importer dans R en utilisant la fonction dbReadTable(). Cette fonction récupère les données d'une table spécifique et les charge sous forme de data frame dans R. Par exemple, pour lire la table customer de la base de données, nous utilisons le code suivant :

```
# Lire la table "customer" de la base de données
clients <- dbReadTable(con, "customer")

# Afficher les premières lignes de la table pour avoir un aperçu des données
head(clients)
```

2.6 Export de données

Il peut être nécessaire d'exporter un tableau de données depuis R vers un fichier dans différents formats, que ce soit pour le partager avec d'autres utilisateurs ou pour l'utiliser dans un autre logiciel.

La plupart des fonctions d'importation possèdent un équivalent pour l'export. On peut citer notamment :

- `write_csv()`, `write_csv2()`, `write_tsv()` : pour enregistrer un `data.frame` ou un `tibble` dans un fichier texte délimité.
- `write_sas()` : pour exporter au format SAS.
- `write_sav()` : pour exporter au format SPSS.
- `write_dta()` : pour exporter au format Stata.

En revanche, **il n'existe pas de fonction native** pour exporter directement au format Excel (`.xls` ou `.xlsx`).
Dans ce cas, on peut **passer par un export au format CSV**, puis ouvrir ce fichier dans Excel.

Ces fonctions sont particulièrement utiles pour **diffuser des données** ou **assurer l'interopérabilité** entre logiciels.

Si vous travaillez avec des données volumineuses, les formats texte peuvent devenir **lents à lire ou écrire**. Dans ce cas, le package **`feather`** peut s'avérer utile : il permet d'enregistrer un `data.frame` au format **Feather**,
un format binaire non compressé mais **extrêmement rapide** à charger et à enregistrer.

```r
library(feather)
write_feather(mon_tableau, "donnees.feather")
```

Une autre manière de sauvegarder des données dans R consiste à les enregistrer au format **`.RData`**,
un format propre à R, à la fois **compact**, **rapide** à charger, et capable de contenir **plusieurs objets de types variés** dans un seul fichier.

Pour enregistrer un ou plusieurs objets, on utilise la fonction `save()` en lui indiquant la liste des objets à sauvegarder ainsi que le nom du fichier cible :

```r
# Exemple : sauvegarder trois objets
save(objet1, objet2, tableau_donnees, file = "mes_objets.RData")
```

Pour charger des objets précédemment sauvegardés dans un fichier `.RData`, on utilise la fonction `load()` :

```r
load("fichier.RData")
```