Importation-exportation et production de données

Togo Posit community

2025-05-06

Table of contents

1. Importation des données

1.1 Importer des données depuis un fichier texte ASCII

Si vous avez un fichier texte au format **ASCII** contenant vos données, voici comment les importer dans **R**.

1.1.1. Préparer le fichier

Si vous ne disposez pas d'un fichier texte au format ASCII qui contient vos données, vous pouvez créer un fichier texte avec **Wordpad** sous **Windows** ou **Emacs** sous **Linux**. Si votre fichier contient un grand nombre de données, **préférez un tableur** pour la gestion avant l'importation.

1.1.2. Méthodes d'importation dans R

R propose plusieurs fonctions pour lire des fichiers **texte** :

Fonction	Description
`read.table()`	Idéale pour importer des tableaux de données structurés.
`read.ftable()`	Permet de lire des tableaux de contingence.
`scan()`	Plus flexible et puissante pour lire divers types de fichiers.

1.1.3. Lecture d'un fichier texte avec `read.table()`

Voici un script permettant de **charger un fichier texte** sous R et de le stocker sous forme de `data.frame` :

```
# Sélection du fichier via une fenêtre de dialogue
donnees <- read.table(file = file.choose(), header = TRUE, sep = "\t", dec = ".", row.names =
```

Paramètre	Description

Paramètres principaux de `read.table()`

Paramètre	Description
`file`	Emplacement et nom du fichier à lire.
`header = TRUE`	Indique si la première ligne contient les noms des colonnes.
`sep = "\t"`	Délimiteur de colonnes (`"\t"` pour tabulation, `","` pour virgule, `" "` pour espace, etc.).
`dec = "."`	Séparateur décimal (`"."` ou `","` selon le format du fichier).
`row.names = 1`	Si la première colonne contient les identifiants des lignes.

Chargement manuel du fichier

Vous pouvez spécifier directement le chemin du fichier :

```
donnees <- read.table(file = "C:/MonDossier/mesdonnees.txt", header = TRUE, sep = "\t", dec =
```

Attention : Sous **Windows**, utilisez `/` ou `\\` au lieu de `\` dans le chemin du fichier.

Définir le répertoire de travail

Plutôt que de spécifier le chemin complet, vous pouvez **définir un répertoire par défaut** :

```
setwd("C:/MonDossier")
mon.fichier <- "mesdonnees.txt"
donnees <- read.table(file = mon.fichier, header = TRUE, sep = "\t", dec = ".",
                          row.names = 1)
```

1.1.4. Vérification des données

Une fois les données importées, vous pouvez les **visualiser rapidement** :

```r
head(donnees)   # Afficher les premières lignes
tail(donnees)   # Afficher les dernières lignes
```

Si votre fichier contient des **lignes vides** ou **incomplètes**, ajoutez ces paramètres pour éviter les erreurs :

```r
donnees <- read.table(file = "C:/MonDossier/mesdonnees.txt", fill = TRUE,
                              blank.lines.skip = FALSE)
```

1.2. Importation avec d'autres fonctions

R propose plusieurs alternatives adaptées à différents formats :

```r
read.csv("mon_fichier.csv")      # Séparé par des virgules
read.csv2("mon_fichier.csv")     # Séparé par des points-virgules
read.delim("mon_fichier.txt")    # Séparé par des tabulations
```

1.3. Importation avancée

Lorsque les données **ne sont pas sous forme de tableau**, `scan()` est plus adapté :

```r
nom.variable <- scan("Intima_Media.txt", skip = 5, nlines = 1, what = "")
donnee <- scan("Intima_Media.txt", skip = 9, dec = ",")
matable <- as.data.frame(matrix(donnee, ncol = 9, byrow = TRUE))
colnames(matable) <- nom.variable
```

Utilisation typique : - `skip=5` → Ignore les premières **lignes descriptives** du fichier. - `dec=","` → Utilise la **virgule comme séparateur décimal**. **Astuce** : `scan()` permet une **lecture rapide**, idéale pour les gros fichiers !

. Utilisation des packages spécialisés

- `readxl` : **Lit les fichiers `.xls/.xlsx`** sans dépendances externes
- `gdata` : **Lit `.xls`** mais requiert **Perl**
- `foreign` : **Lit des fichiers `.sav`, `.mtp`, `.xpt` et `.mat`**

1.4. Importation depuis d'autres logiciels

Logiciel	Package	Fonction	Extension	Format
SPSS	`foreign`	`read.spss()`	`.sav`	`list`
Minitab	`foreign`	`read.mtp()`	`.mtp`	`list`
SAS	`foreign`	`read.xport()`	`.xpt`	`data.frame`
Matlab	`R.matlab`	`readMat()`	`.mat`	`list`

1.5. Gestion des gros fichiers

- Définir les types de colonnes (`colClasses`) pour optimiser la lecture
- Stocker dans une base de données SQL en cas de **limite RAM**

2. Exportation des données dans R

Avant d'exporter, il faut disposer des données en R :

```r
# Création d'un exemple de jeu de données
X <- data.frame(
  Nom = c("Alice", "Bob", "Charlie"),
  Age = c(25, 30, 35),
  Taille = c(160, 175, 180)
)
```

2.1. Exporter vers un fichier texte ASCII

Utilisez la fonction **`write.table()`** pour enregistrer un **data.frame** dans un fichier texte.

```r
write.table(donnees, file = "mon-fichier.txt", sep = "\t", row.names = FALSE)
```

Explication des paramètres :
- `file = "mon-fichier.txt"` → Nom du fichier de sortie.
- `sep = "\t"` → Séparateur de colonnes (ici, tabulation).
- `row.names = FALSE` → Ne pas enregistrer les noms des lignes.

Autre méthode : La fonction `write()` permet d'exporter **vecteurs et matrices**, en précisant le nombre de colonnes :

```r
write(matrix(1:12, nrow=3), file = "data.txt", ncolumns = 4)
```

2.2. Exporter des données vers Excel avec `write.csv()` dans R

2.2.1. Exportation dans excel par copie coller

Une méthode simple consiste à **copier les données dans le presse-papiers** et à les coller dans un tableur :

```r
X <- data.frame(Poids = c(80, 90, 75), Taille = c(182, 190, 160))
write.table(X, file = "clipboard", sep = "\t", dec = ",", row.names = FALSE)
```

Collez ensuite les données dans Excel ou LibreOffice avec **CTRL + V**. L'exportation de données vers **Excel** peut être réalisée efficacement à l'aide de la fonction **`write.csv()`**, qui permet de générer un fichier **CSV** compatible avec Excel et d'autres tableurs.

2.2.2. Exportation dans un fichier CSV

La fonction **`write.csv()`** permet de **sauvegarder les données dans un fichier CSV** :

```r
write.csv(X, file = "mes_donnees.csv", row.names = FALSE)
```

Explication des paramètres :
- `file = "mes_donnees.csv"` → Nom du fichier à enregistrer.
- `row.names = FALSE` → Supprime l'ajout automatique des numéros de ligne.

Par défaut, `write.csv()` utilise `,` comme séparateur de colonnes et `.` pour les décimales.

. Spécifier un séparateur de colonnes

Si vous souhaitez **un séparateur différent**, par exemple **";"**, utilisez `write.csv2()` :

```r
write.csv2(X, file = "mes_donnees.csv", row.names = FALSE)
```

Différences entre `write.csv()` et `write.csv2()` :

Fonction	Séparateur de colonnes	Séparateur décimal
`write.csv()`	`,` (virgule)	`.` (point)
`write.csv2()`	`;` (point-virgule)	`,` (virgule)

Utilisez `write.csv2()` si votre version d'Excel s'attend à des séparateurs de type `;` (français/européen).

Sauvegarde avec encodage UTF-8 (pour éviter les problèmes de caractères)

Si votre fichier contient des accents ou caractères spéciaux, utilisez **`fileEncoding`** :

```
write.csv(X, file = "mes_donnees.csv", row.names = FALSE, fileEncoding = "UTF-8")
```

Comment ouvrir le fichier CSV exporté

Une fois votre fichier créé, ouvrez **Excel**, puis : 1. Allez dans **Fichier** , **Ouvrir**. 2. Sélectionnez **"mes_donnees.csv"**. 3. Si les colonnes ne sont pas bien séparées, utilisez l'Assistant d'importation : Sélectionnez **Délimité**. Choisissez le séparateur correct (**,** ou **;**). Vérifiez l'encodage UTF-8 si des caractères sont corrompus.

2.3. Exportation vers Excel avec le package `writexl` (alternative sans CSV)

Si vous préférez **enregistrer directement en `.xlsx`**, utilisez le package `writexl` :

```
install.packages("writexl")
library(writexl)
# Exportation directe vers Excel
write_xlsx(X, "mes_donnees.xlsx")
```

En résumé

`write.csv()` → Exportation standard en **CSV** avec `,` comme séparateur
`write.csv2()` → Format européen avec `;` comme séparateur
`fileEncoding = "UTF-8"` → Évite les problèmes de caractères spéciaux
`writexl` → Exportation directe en `.xlsx` sans passer par CSV

3. Générer des données dans R

3.1. Générer des données simples

Vecteurs avec `c()`

```r
vec <- c(1, 5, 8, 2.3)
```

Suites de valeurs avec `seq()`

```r
seq(from = 4, to = 5, by = 0.1)  # Pas de 0.1
seq(from = 4, to = 5, length = 8) # 8 valeurs réparties
```

Séquence d'entiers avec :

```r
1:12  # Génère 1, 2, ..., 12
```

3.2. Générer des données pseudo-aléatoires

Valeurs uniformes avec `runif()`

```r
runif(5, min = 2, max = 7)  # 5 valeurs entre 2 et 7
```

Valeurs normales avec `rnorm()`

```r
rnorm(5, mean = 2, sd = 3)  # Moyenne 2, écart-type 3
```

Tirage aléatoire avec `sample()`

```r
urne <- 0:9
sample(urne, 20, replace = TRUE)  # Tirage avec remise
```

3.3. Entrer manuelle des données dans R

Avec `scan()`

```
z <- scan()  # Entrée interactive
```

Tapez vos nombres et **appuyez sur "Entrée"** pour valider. Cloturer la saisie par **Entrée**.

** Avec `data.entry()` (éditeur graphique)**

```
rm(list = ls())  # Effacer les objets existants
data.entry("")
```

Les données saisies dans `data.entry("")` sont automatiquement stockées dans l'environnement R, mais elles ne sont pas sauvegardées sur le disque.

Manipuler un tableau interactif

Si vous souhaitez entrer des **données directement dans un mini-tableur**, utilisez :

```
X <- as.data.frame(de(""))
```

Modifiez les noms de variables et les types (`numeric`, `character`) en cliquant sur la **première ligne du tableau**.

4. Lecture/écriture dans les bases de données SQL

R peut interagir avec plusieurs **Systèmes de Gestion de Bases de Données (SGBD)** via des **drivers spécialisés**. Voici une procédure **générale** applicable à **MySQL, PostgreSQL et SQL Server**.

4.1. Procédure générale pour se connecter aux SGBD externes (MySQL, PostgreSQL, SQL Server) avec R

4.1.1. Installer le package adapté

Selon le SGBD utilisé, installez le package correspondant :

MySQL → `RMySQL`
PostgreSQL → `RPostgreSQL`
SQL Server → `odbc`

Installation du package :

```
install.packages("RMySQL")  # Pour MySQL
install.packages("RPostgreSQL")  # Pour PostgreSQL
install.packages("odbc")  # Pour SQL Server
```

Puis, **chargez le package** :

```
library(RMySQL)  # Exemple pour MySQL
```

4.1.2. Établir la connexion à la base de données

La connexion varie légèrement selon le SGBD, mais suit la structure générale :

Connexion à MySQL

```
con <- dbConnect(RMySQL::MySQL(),
                 dbname = "nom_base",
                 host = "localhost",
                 user = "nom_utilisateur",
                 password = "mot_de_passe")
```

Connexion à PostgreSQL

```
con <- dbConnect(RPostgreSQL::PostgreSQL(),
                 dbname = "nom_base",
                 host = "localhost",
                 user = "nom_utilisateur",
                 password = "mot_de_passe")
```

Connexion à SQL Server via ODBC

```
con <- dbConnect(odbc::odbc(),
                 Driver = "SQL Server",
                 Server = "nom_du_serveur",
                 Database = "nom_base",
                 UID = "nom_utilisateur",
                 PWD = "mot_de_passe")
```

Note : Pour **SQL Server**, vous devez installer le **driver ODBC** correspondant à votre version.

4.1.3. Exécuter une requête SQL

Une fois connecté, vous pouvez envoyer des requêtes **SQL** depuis R :

```
# Récupérer toutes les lignes d'une table
result <- dbGetQuery(con, "SELECT * FROM clients")
print(result)
```

4.1.4. Insérer des données dans la base

Ajoutez des données directement avec SQL :

```
dbExecute(con, "INSERT INTO clients (nom, age, ville) VALUES ('Alice', 30, 'Paris')")
```

Ou utilisez un **data.frame** :

```
clients_df <- data.frame(nom = c("David", "Emma"),
                         age = c(22, 29),
                         ville = c("Nice", "Toulouse"))

dbWriteTable(con, "clients", clients_df, append = TRUE, row.names = FALSE)
```

4.1.5. Fermer la connexion

Une fois les opérations terminées, **fermez la connexion** pour libérer les ressources :

```r
dbDisconnect(con)
```

4.2. Cas pratique avec SQLite

SQLite est une **base de données légère et intégrée** ne nécessitant pas de serveur externe. Cette section présente les opérations courantes permettant d'utiliser **SQLite avec R**.

4.2.1. Installation et chargement du package SQLite

Pour communiquer avec SQLite, on utilise le package **`RSQLite`**. Si ce n'est pas encore fait, installez-le :

```r
#install.packages("RSQLite")
library(RSQLite)
```

4.2.2. Création d'une base de données SQLite

La première étape consiste à **créer** une base de données SQLite ou à se connecter à une base existante.

```r
# Connexion à une base de données SQLite (création si elle n'existe pas)
con <- dbConnect(SQLite(), dbname = "ma_base.sqlite")
#Pour un stockage de la base dans la memeoire utiliser:
con <- dbConnect(SQLite(), dbname =  ":memory:")
```

Explication : - `dbConnect(SQLite(), dbname = "ma_base.sqlite")` → Crée ou ouvre un fichier **SQLite** nommé `ma_base.sqlite`.

4.2.3. Création d'une table

On peut maintenant **créer une table** dans cette base de données :

```r
dbExecute(con, "
    CREATE TABLE clients (
        id INTEGER PRIMARY KEY,
        nom TEXT,
        age INTEGER,
        ville TEXT
```

```
    )
")
```

```
[1] 0
```

Explication : - La table `clients` contient **4 colonnes** (`id`, `nom`, `age`, `ville`). - La colonne `id` est une **clé primaire**.

4.2.4. Insérer des données dans SQLite

On peut insérer des données avec **SQL** directement :

```
dbExecute(con, "
    INSERT INTO clients (nom, age, ville) VALUES
    ('Alice', 28, 'Paris'),
    ('Bob', 35, 'Lyon'),
    ('Charlie', 40, 'Marseille')
")
```

```
[1] 3
```

Alternative avec un `data.frame` ::: {.cell}

```
# Création d'un data.frame
clients_df <- data.frame(
    nom = c("David", "Emma"),
    age = c(22, 29),
    ville = c("Nice", "Toulouse")
)
# Ajout des données dans SQLite
dbWriteTable(con, "clients", clients_df, append = TRUE, row.names = FALSE)
```

:::

4.2.5. Lecture des données depuis SQLite

Pour **récupérer** les données stockées dans la base de données :

```r
res <- dbGetQuery(con, "SELECT * FROM clients")
print(res)  # Affichage des données récupérées
```

```
  id     nom age     ville
1  1   Alice  28     Paris
2  2     Bob  35      Lyon
3  3 Charlie  40 Marseille
4  4   David  22      Nice
5  5    Emma  29  Toulouse
```

Explication : - `dbGetQuery()` exécute une requête **SELECT** et retourne un **data.frame** en R.

4.2.6. Suppression et mise à jour des données

Mettre à jour une entrée

```r
dbExecute(con, "UPDATE clients SET age = 90 WHERE nom = 'Alice'")
```

```
[1] 1
```

Supprimer un enregistrement

```r
dbExecute(con, "DELETE FROM clients WHERE nom = 'Bob'")
```

```
[1] 1
```

```r
res <- dbGetQuery(con, "SELECT * FROM clients")
print(res)  # Affichage des données récupérées
```

```
  id     nom age     ville
1  1   Alice  90     Paris
2  3 Charlie  40 Marseille
3  4   David  22      Nice
4  5    Emma  29  Toulouse
```

4.2.7. Fermeture de la connexion

Une fois les opérations terminées, il est **important de fermer la connexion** :

```
dbDisconnect(con)
```

Exercices et Travaux pratiques

Exercice

1 – Quelles sont les trois fonctions principales de R permettant d'importer des données depuis un fichier texte au format ASCII ?

2 – Une fonction couramment utilisée pour la lecture de données possède les paramètres suivants : `header`, `sep`, `dec`, `row.names`, `skip`, `nrows`.
- Expliquez leur rôle et donnez un exemple de valeur que peut prendre chacun d'entre eux.

3 – À quoi sert la fonction `readLines()` ?

4 – Quelle est l'utilité de la fonction `fix()` ?

5 – Décrivez les particularités des fonctions `read.csv()`, `read.csv2()`, `read.delim()` et `read.delim2()`.

6 – À quoi sert la fonction `read.ftable()` ?

7 – En quoi les fonctions `scan()` et `read.table()` se différencient-elles ?

8 – Expliquez en détail comment importer des données provenant d'une feuille de calcul **Excel** dans R.

9 – Quel package contient plusieurs fonctions permettant d'importer des données depuis des logiciels commerciaux de statistique ?

10 – Lors de la lecture de **gros fichiers de données**, quel paramètre de la fonction `read.table()` permet d'accélérer significativement la vitesse de lecture ?

11 – Quelle fonction R permet d'écrire un jeu de données contenu dans un **data.frame** vers un fichier texte ?
- Citez une autre fonction ayant une fonction similaire.

12 – Citez **quatre fonctions de base** permettant de créer des **vecteurs** en R.

13 – Indiquez comment utiliser la fonction `seq()` pour obtenir le vecteur suivant :

```
[1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0
```

14 – Donnez l'instruction R la plus concise permettant d'obtenir le vecteur suivant :

```
[1] 1 1 2 2 3 3
```

15 – Donnez l'instruction R la plus concise permettant d'obtenir le vecteur suivant :

```
[1] 1 2 3 1 2 3
```

16 – Citez **deux fonctions R** permettant **d'entrer des données manuellement** dans le **mini-tableur** de R.

Bien sûr ! Voici le texte corrigé, en éliminant les erreurs typographiques et en améliorant la lisibilité :

Travaux Pratiques: Lecture de différents jeux de données

A- Saisie des données issues d'un format papier

A.1 Boutons de fièvre

Cinq traitements contre les boutons de fièvre, dont un placebo, ont été administrés par tirage au sort à trente patients (six patients par groupe de traitement). Le délai (en jours) entre l'apparition des boutons et la cicatrisation complète a été recueilli chez chaque patient.

Traitements

Traitements	trt1 (placebo)	trt2	trt3	trt4	trt5
Patient 1	5	4	6	7	9
Patient 2	8	6	4	4	3
Patient 3	7	6	4	6	5
Patient 4	7	3	5	6	7
Patient 5	10	5	4	3	7
Patient 6	8	6	3	5	6

La question que l'on se pose est de savoir s'il existe une différence entre les traitements. Il s'agit donc ici de comparer les moyennes des délais de cicatrisation observés dans cinq échantillons indépendants (groupes de traitement).

L'analyse statistique adéquate s'appelle l'**ANOVA**; elle sera présentée au chapitre 13. Nous allons voir ici comment entrer ces données dans **R** uniquement pour calculer la moyenne de l'échantillon de chaque traitement.

1. **Entrez directement les données dans R** à l'aide de la fonction `c()`.

2. **Utilisez la fonction `attach()` puis la fonction `mean()`** pour calculer la moyenne de chacun des traitements.

3. **Calculez simultanément les moyennes des traitements** au moyen de la fonction `colMeans()`.

4. **Enregistrez votre `data.frame` dans un fichier nommé `boutons.txt`** en utilisant la fonction `write.table()`.

5. **Vérifiez que tout s'est bien passé** en ouvrant le fichier au moyen d'un éditeur de texte.

6. **Utilisez la fonction `rm()`** pour effacer tous les objets R que vous venez de créer dans votre environnement de travail.

7. **Importez le fichier `boutons.txt`** en utilisant `read.table()` et affichez-le.

A.2 Facteurs de risque de l'athérosclérose

Lors d'une étude sur les facteurs de risque de l'athérosclérose, des données ont été recueillies et résumées dans le tableau de contingence suivant :

Sexe	Tabac	Ne boit pas	Boit occasionnellement	Boit régulièrement
H	Ne fume pas	6	19	7
H	A arrêté de fumer	0	9	0
H	Fume	1	6	5
F	Ne fume pas	12	26	2
F	A arrêté de fumer	3	5	1
F	Fume	1	6	1

Il peut être intéressant de savoir s'il y a une dépendance entre les habitudes tabagiques et alcooliques suivant le sexe. Pour entrer ce type de données dans **R**, il faut suivre plusieurs étapes.

1. **Utilisez la fonction `scan()`** afin d'obtenir une matrice `X` de taille `6×3`, qui contiendra uniquement les données.

2. **Spécifiez que `X` est une table de contingence** à l'aide de l'instruction `class(X) <- "ftable"`.

3. **Définissez les attributs de la table** en tapant les deux instructions suivantes :

```r
attributes(X)$col.vars <- list(alcool = c("ne boit pas", "boit occasionnellement", "boit régu
attributes(X)$row.vars <- list(SEXE = c("H", "F"), tabac = c("ne fume pas", "a arrêté de fume
```

4. **Affichez votre tableau de contingence ainsi créé**.

5. **Enregistrez votre tableau de contingence** dans un fichier nommé `athero.txt` en utilisant la fonction `write.ftable()`.

6. **Vérifiez que l'enregistrement s'est bien déroulé** en ouvrant le fichier avec un éditeur de texte.

7. **Utilisez la fonction `rm()`** pour effacer tous les objets `R` que vous venez de créer dans votre environnement de travail.

8. **Importez le fichier `athero.txt`** en utilisant `read.ftable()` et affichez-le.

B- Importer des données depuis un logiciel externe

Lors de l'étude de l'**IMC** (indice de masse corporelle) chez des enfants, un fichier de données a été recueilli sous plusieurs formats différents par une équipe de statisticiens. Nous allons nous entraîner à lire ces différents formats.

Il existe plusieurs fichiers portant le nom **imcenfant**, mais avec des extensions différentes.

1. **Importez le fichier `imcenfant.xls`** dans un `data.frame` nommé `imc.XLS`.

2. **Importez le fichier `imcenfant.xpt`** dans un `data.frame` nommé `imc.SAS`.

3. **Importez le fichier `imcenfant.sav`** dans un `data.frame` nommé `imc.SPSS`.

4. **Importez le fichier `imcenfant.mat`** dans un `data.frame` nommé `imc.MAT`.

Comme la procédure pour importer un fichier `.mat` est plus complexe, voici les étapes détaillées :

```r
x <- readMat("imcenfant.mat")
class(x) # x est une liste
x # les données sont dans $imc[,,1]
x <- x$imc[,,1]

# Notez que les éléments de SEXE et zep sont enregistrés dans une liste
x$SEXE
```

```
class(x$SEXE) <- "character"
x$SEXE
class(x$zep) <- "character"

imc.MAT <- as.data.frame(x)
```

5. **Vérifiez que l'importation s'est bien déroulée** en utilisant la fonction `summary()` sur tous ces `data.frames`. Celle-ci affichera des résumés numériques.

6. **Sauvegardez l'un de ces `data.frames`** dans un fichier nommé `imcenfant.txt`.

C- Importer des fichiers de données plus complexes

Dans la pratique d'un Data scientist, il arrive souvent que l'on rencontre des fichiers de données ayant un format d'enregistrement non standard. Nous allons donc nous entraîner à lire plusieurs fichiers de ce type, sur lesquels nous serons amenés à réaliser des analyses statistiques.

1. **Importez le fichier `raf98.gra`** dans la structure la plus adaptée. Pour cela, consultez le fichier associé `formatgeoide.txt`, qui contient la description du format de ce fichier.

2. **Importez le fichier `Infarct.xls`** dans un `data.frame`, en vous assurant de bien traiter les valeurs manquantes.

3. **Importez le fichier `nutriage.txt`**, qui contient **treize variables mesurées sur 226 individus**, dans un `data.frame`. *(Indice : utilisez entre autres les fonctions `as.data.frame()` et `t()`).*

4. **Importez le fichier `Poids naissance.txt`**, qui contient **dix variables mesurées sur 189 individus**, dans un `data.frame`. Celui-ci devra contenir **le nom des variables et le nom des individus** (correspondant à la colonne **Id**). Pensez à utiliser l'aide en ligne pour vous guider.

Astuce: Utiliser à chaque importation les fonctions de base : (`str()`,`head()`, `tail()`, `summary()`), pour avoir un aperçu des données.