Analyse des données

Philippe Vanden Eeckaut

2024-08-15

1 Importation/Exportation (IO)

1.1 Principes

La lecture de données se réalise de diverses manières:

- Données internes de R (importation)
  - Depuis les packages.
- Données externes de R (importation/exportation)
  - Données compressées
    - Format propriétaire de R (.rda/.rds)
    - Format ouvert (.feather/.parquet)
  - Données au format texte
  - Fichiers textes formatés (.csv)
  - Données au format d'autres applications
  - Fichiers de tableurs (.xls,.xlsx)
  - fichiers statistiques (sas,spss,....=)

1.2 Packages

On va à partir de cette session utiliser des ***packages***. Pour utiliser un package deux opérations sont nécessaires:

- *Installer le package*:
  - La première opération consiste à récupérer le package sur un serveur. Un serveur est toujours utilisé par défaut dans Rstudio.
  - Pour vérifier le serveur utilisé il faut aller voir la section package dans les préférence de Rstudio.
  - L'installation d'un package se fait soit par l'intetface de Rstudio soit dans la fenêtre ‹ sud est › de Rstudio et l'onglet package avec Install soit par l'instruction `install.packages()`. Il sera alors présent sur la machine (il est disponible dans la liste de la fenêtre package de Rstudio)
- *Charger le package*:
  - Pour utiliser le package durant une session de Rstudio il est nécessaire de charger le package par la commande `library()`.
  - Dans la fenêtre `sud-est` à l'onglet `Packages` on dispose de la liste des packages disponible sur la machine et ceux chargés seront cochés. (Certains packages sont toujoutrs disponibles: base,datasets,datasets,grDevices,methods,utils)

Pour simplifier l'ensemble du processus, il est plus facile réaliser les deux opérations en une fois avec les instructions ci-dessous

```
Mes_packages = c('rio',
                 'openxlsx','readxl',
                 'feather','arrow',
                 'data.table',
                 'here',
                 'rbenchmark','fakir')
# Liste des packages nécessaires
for(p in Mes_packages){
  if(!require(p,character.only = TRUE))
    install.packages(p,repos = "http://cran.us.r-project.org")
  # si le package est absent on l'installe
  library(p,character.only = TRUE)
  # On charge ensuite tous les packages
}
```

1.3 Données internes

La manière la plus simple est de lire les données incluses directement dans un package. Au départ une ensemble de packages est déjà chargé dans Rstudio. Ainsi par exemple, nous avons le package `datasets`qui comporte de nombreuses données qui sont visible avec la fonction `data()'

```
dataset <- data(package="datasets")  # Lecture de la liste
head(dataset$results[,3:4])          # Affichage des 6 premiers
```

```
>      Item                      Title
> [1,] "AirPassengers"         "Monthly Airline Passenger Numbers 1949-1960"
> [2,] "BJsales"                "Sales Data with Leading Indicator"
> [3,] "BJsales.lead (BJsales)" "Sales Data with Leading Indicator"
> [4,] "BOD"                    "Biochemical Oxygen Demand"
> [5,] "CO2"                    "Carbon Dioxide Uptake in Grass Plants"
> [6,] "ChickWeight"          "Weight versus age of chicks on different diets"
```

Si nous avons besoin de données présentes dans un package précis, il est nécessaire de le charger au préalable.

```
install.packages("plm",repos = "http://cran.us.r-project.org")
```

```
>
> The downloaded binary packages are in
>   /var/folders/gw/b5_9kkkx3k7949g_gnhs0nkr0000gn/T//RtmpcAnLby/
downloaded_packages
```

Par la même commande `data(package=<package>)` il est possible de visualiser les données de ce package. Les données sont visibles uniquement dans une fenêtre de Rstudio.

```r
data(package="plm")  # Liste des données disponibles
```

```
Data sets in package 'plm':

Cigar                 Cigarette Consumption
Crime                 Crime in North Carolina
EmplUK                Employment and Wages in the United Kingdom
Gasoline              Gasoline Consumption
Grunfeld              Grunfeld's Investment Data
Hedonic               Hedonic Prices of Census Tracts in the
                      Boston Area
LaborSupply           Wages and Hours Worked
Males                 Wages and Education of Young Males
Parity                Purchasing Power Parity and other parity
                      relationships
Produc                US States Production
RiceFarms             Production of Rice in Indonesia
Snmesp                Employment and Wages in Spain
SumHes                The Penn World Table, v. 5
Wages                 Panel Data of Individual Wages
```

Pour nos besoins nous utiliserons l'ensemble de données « Wages ». Ces données regroupent des informations sur une cohorte de 595 personnes qui sont collectées aux Etats Unis annuellement entre 1976 et 1982 soit 4165 observations.

```r
data(Wages,package="plm") # Lecture du fichier "Wages"
head(Wages,n=7)           # Affichage des 6 premières lignes
```

```
>   exp wks bluecol ind south smsa married  sex union ed black   lwage
> 1   3  32      no   0   yes   no     yes male    no  9    no 5.56068
> 2   4  43      no   0   yes   no     yes male    no  9    no 5.72031
> 3   5  40      no   0   yes   no     yes male    no  9    no 5.99645
> 4   6  39      no   0   yes   no     yes male    no  9    no 5.99645
> 5   7  42      no   1   yes   no     yes male    no  9    no 6.06146
> 6   8  35      no   1   yes   no     yes male    no  9    no 6.17379
> 7   9  32      no   1   yes   no     yes male    no  9    no 6.24417
```

Une fois les données lues, il est utile de connaitre la description de ces données. Une possibilité est d'aller dans l'aide de R et d'afficher les informations disponibles:

```
Wages {plm} R Documentation
Panel Data of Individual Wages
Description
A panel of 595 individuals from 1976 to 1982, taken from the Panel Study of
Income Dynamics (PSID).

The data are organized as a stacked time series/balanced panel, see Examples
on how to convert to a pdata.frame.
```

```
Format
A data frame containing:

exp       years of full-time work experience.
wks       weeks worked.
bluecol   blue collar?
ind       works in a manufacturing industry?
south     resides in the south?
smsa      resides in a standard metropolitan statistical area?
married   married?
sex       a factor with levels "male" and "female"
union     individual's wage set by a union contract?
ed        years of education.
black     is the individual black?
lwage     logarithm of wage.

Details
total number of observations : 4165

observation : individuals

country : United States

Source
Online complements to Baltagi (2001):

https://www.wiley.com/legacy/wileychi/baltagi/

Online complements to Baltagi (2013):

https://bcs.wiley.com/he-bcs/Books?action=resource&bcsId=4338&itemId=
1118672321&resourceId=13452

References
Baltagi BH (2001). Econometric Analysis of Panel Data, 3rd edition. John
Wiley and Sons ltd.

Baltagi BH (2013). Econometric Analysis of Panel Data, 5th edition. John
Wiley and Sons ltd.

Cornwell C, Rupert P (1988). "Efficient Estimation With Panel Data: an
Empirical Comparison of Instrumental Variables Estimators." Journal of Applied
Econometrics, 3, 149–155.

Examples
Run examples

# data set 'Wages' is organized as a stacked time series/balanced panel
data("Wages", package = "plm")
Wag <- pdata.frame(Wages, index=595)
```

Il est ensuite possible de régarder la description de ces données par l'instruction `str()`

```r
str(Wages)            # Affichage de la description
```

```
> 'data.frame': 4165 obs. of  12 variables:
>  $ exp    : int  3 4 5 6 7 8 9 30 31 32 ...
>  $ wks    : int  32 43 40 39 42 35 32 34 27 33 ...
>  $ bluecol: Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 2 2 2 ...
>  $ ind    : int  0 0 0 0 1 1 1 0 0 1 ...
>  $ south  : Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 1 1 1 ...
>  $ smsa   : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 ...
>  $ married: Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 ...
>  $ sex    : Factor w/ 2 levels "male","female": 1 1 1 1 1 1 1 1 1 ...
>  $ union  : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 2 ...
>  $ ed     : int  9 9 9 9 9 9 11 11 11 ...
>  $ black  : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 ...
>  $ lwage  : num  5.56 5.72 6 6 6.06 ...
```

Nous allons regarder plus spécifiquement les variables suivantes:

Variable	type	Définition
exp	int	années d'expériences (plein temps)
bluecol	Factor	ouvrier ? (yes,no)
sex	Factor	sexe ? (male,female)
ed	int	années d'étude
lwage	num	log du salaire

1.3.a Exercice

Nous allons sélectionner un sous-ensemble des 100 ouvriers avec plus de 10 ans pour l'expérience et moins de 10 ans pour les études en 1980 et classer le résultat selon le salaire décroissant.La table contiendra les variables suivante: id,sex,exp,ed,lwage.

```r
# 1. Obtenir les informations manquantes
#    ---------------------------------
# Technique manuelle basée sur rep()
head(Wages,7)
```

```
>   exp wks bluecol ind south smsa married  sex union ed black   lwage
> 1   3  32      no   0   yes   no     yes male    no  9    no 5.56068
> 2   4  43      no   0   yes   no     yes male    no  9    no 5.72031
> 3   5  40      no   0   yes   no     yes male    no  9    no 5.99645
> 4   6  39      no   0   yes   no     yes male    no  9    no 5.99645
> 5   7  42      no   1   yes   no     yes male    no  9    no 6.06146
> 6   8  35      no   1   yes   no     yes male    no  9    no 6.17379
> 7   9  32      no   1   yes   no     yes male    no  9    no 6.24417
```

```r
Wages_p <- Wages
Wages_p$time <- rep(seq(1:7),595)
Wages_p$id <- rep(1:595,each=7)
# 2. Réaliser le filtre
#    -----------------
Wages_s <- subset(Wages_p,
                  bluecol == "yes" & exp >=10 & ed < 10 & time==5 ,
                  select = c(id,sex,exp,ed,lwage))
Wages_s <- head(Wages_s[order(Wages_s$lwage,decreasing=TRUE),],25)
str(Wages_s)
```

```
> 'data.frame': 25 obs. of  5 variables:
>  $ id   : int  496 224 486 55 269 123 65 435 367 91 ...
>  $ sex  : Factor w/ 2 levels "male","female": 1 1 1 1 1 1 1 1 1 1 ...
>  $ exp  : int  37 38 36 36 25 31 34 17 29 14 ...
>  $ ed   : int  8 5 9 8 9 8 8 9 9 7 ...
>  $ lwage: num  7.39 7.12 7.09 7.04 7 ...
```

```r
Wages_s
```

```
>       id  sex exp ed   lwage
> 3470 496 male  37  8 7.39080
> 1566 224 male  38  5 7.12287
> 3400 486 male  36  9 7.09008
> 383   55 male  36  8 7.04141
> 1881 269 male  25  9 7.00307
> 859  123 male  31  8 6.93925
> 453   65 male  34  8 6.93828
> 3043 435 male  17  9 6.93245
> 2567 367 male  29  9 6.88244
> 635   91 male  14  7 6.85646
> 824  118 male  29  9 6.85646
> 915  131 male  34  8 6.85646
> 1468 210 male  38  8 6.85646
> 1426 204 male  36  9 6.80239
> 1937 277 male  37  8 6.80239
> 2791 399 male  35  6 6.77194
> 4135 591 male  38  8 6.77079
> 2651 379 male  26  7 6.73934
> 1328 190 male  10  9 6.71901
> 2455 351 male  27  8 6.69703
> 460   66 male  24  8 6.68711
> 1370 196 male  11  9 6.68461
> 1713 245 male  40  7 6.66568
> 2238 320 male  24  6 6.59578
> 2070 296 male  34  8 6.59167
```

1.4 Données externes

L'activité la plus régulière en cas d'analyse de données sera de lire et d'écrire des données de ou vers des ressources extérieures à R. Il existe de très nombreux formats possibles. Dans le cadre de ce cours nous allons insister sur les catégories suivantes

Format	Caractéristiques	Exemple
Compressées R	I/O uniquement par R	.RData .rda
Compressées	I/O universel	.parquer .feather
Textes	I/O universel	.csv,...
Spécifiques	I/O spécifique	.xlsx,...

De nombreux packages sont nécessaires pour convertir les différents formats possibles. Un package `rio` regroupe tous ces packages sous une interface commune

1.4.a Données Compressés

Afin de gagner en efficacité, les données peuvent etre compressés selon un algorithme choisi (zip,...). Nous allons distinguer deux possibiliés, soit les données sont lisibles uniquement par le créateur des données (dans notre cas R), soit les données sont lisibles de manière universelle par un format reconnu nativement au m

1.4.a.a Fichiers propriétaires R: .RData,.rda

Nous examinerons les fichiers suivants:

Format	Extension	Importation (Package)	Exportation (Package)	rio ?
R objects (global)	.RData, .rda	**base**	**base**	Présent
R objects (unique)	.rds	**base**	**base**	Présent

Il est possible de sauvegarder l'ensemble ou une partie des données définies dans la session de R. Cette sauvegarde se réalise avec les instructions `save()` pour un ensemble d'objets et `save.image()`pour sauvegarder tous les objets.

La lecture se réalise par l'instruction `load()` et se réalise sous la forme d'un fichier binaire. Le format des fichiers est de type binaire et est pas conséquent un fichier propriétaire de R (compressé par R et lisible uniquement par R).

Nous allons expérimenter en créant trois objets en plus des objets existants

```
ls()  # Objets présents
```

```
> [1] "dataset"      "Mes_packages" "p"            "Wages"        "Wages_p"
> [6] "Wages_s"
```

```
a <- c("Université de Lille")
b <- TRUE
c <- c(10,20,30) # Création de 3 objets
ls()    # Liste des objets
```

```
> [1] "a"           "b"           "c"           "dataset"     "Mes_packages"
> [6] "p"             "Wages"       "Wages_p"     "Wages_s"
```

Ensuite nous sauvegardons uniquement les objets a,b et c par l'instruction `save()`. En ensuite nous sauvgardons l'ensemble des objets par l'instruction `save.image()`. Nous vérifions ensuite les tailles des fichiers respectifs par l'instruction `file.info()$size`.

```
save(a,b, file=here("data/Lille.RData"))
# Sauvegarde de deux objets sur un fichier .Rdata
save.image(file=here("data/Lille_image.RData"))
# Sauvegarde totale sur un fichier .Rdata
file.info(here("data/Lille.RData"))$size
```

```
> [1] 100
```

```
# Vérification du fichier
file.info(here("data/Lille_image.RData"))$size
```

```
> [1] 204088
```

```
# Vérification du fichier
```

On vérifie l'opération en retirant les objets et en les rechargeant ensuite.

```
rm(list = c("a","b","c"))               # Suppression de a et b
ls()                                    # Liste de objets vide
```

```
> [1] "dataset"     "Mes_packages" "p"           "Wages"       "Wages_p"
> [6] "Wages_s"
```

```
load(file=here("data/Lille.RData")); ls()
```

```
> [1] "a"           "b"           "dataset"     "Mes_packages" "p"
> [6] "Wages"       "Wages_p"     "Wages_s"
```

```
load(file=here("data/Lille_image.RData"));ls()
```

```
> [1] "a"           "b"           "c"           "dataset"     "Mes_packages"
> [6] "p"             "Wages"       "Wages_p"     "Wages_s"
```

L'avantage unique de ce type de fichier est la capacité de sauvegarder dans un seul fichier l'ensemble des objets peut importe le type.

1.4.a.b Fichiers .rds

Le type de fichier .rds est plus limité et seul un objet sera sauvegardé (dataframe, matrice, liste,...). L'écriture se réalise avec `saveRDS()`, l'extension du fichier est .rds. La lecture se réalise avec l'instruction `readRDS`.

```
data(Wages,package="plm")
saveRDS(Wages,file=here("data/Wages_binaire.rds"))    # Ecriture fichier
binaire
file.info(here("data/Wages_binaire.rds"))$size       # Taille du fichier
```

```
> [1] 34403
```

```
Wages_RDS <- readRDS(here("data/Wages_binaire.rds")) # Lecture du fichier
all.equal(Wages_RDS,Wages)                          # Fichiers égaux ?
```

```
> [1] TRUE
```

```
rm(Wages_RDS)                                       # Effacement données
```

Nous disposeront également d'une possibilité plus simple de travailler si nous utilisont le package rio, il consiste qui a pour objet d'uniformiser la syntaxe pour les différents formats. Ainsi il suffira simplement d'utiliser la fonction rio::import(nom.extension) pour une importation et rio::export(nom.extension) pour une exportation. L'extension définira la méthode à utiliser. Le package choisit habituellement la fonction existante la plus performante pour réaliser la tâche.

Par exemple, voici une importation et exportation du fichier Wages comme précedemment avec rio:

```
data(Wages,package="plm")
export(Wages,file=here("data/Wages_rio.rds"))   # Ecriture fichier compressé
file.info(here("data/Wages_rio.rds"))$size      # Taille du fichier
```

```
> [1] 34403
```

```
Wages_rio <- import(here("data/Wages_rio.rds")) # Lecture du fichier
all.equal(Wages_rio,Wages)                     # Fichiers égaux ?
```

```
> [1] TRUE
```

```
rm(Wages_rio)                                    # Effacement données
```

1.4.b Fichiers compressés universels

Dans le cadre de fichiers portables, nous allons nous intéressé à deux formats de fichiers de type colonnes, qui ont l'avantage d'être supportés par de nombreuses plateformes:

- Le format Feather qui va contenir soit des tables Arrow ou des data frames de R ou Python. Son usage principal est de permettre un stockage de données commun à Python et R.
- Le format Parquet qui est un format de type colonne principalement utilisé pour les problème utilisant les « big data »

https://www.cetic.be/Apache-Parquet-pour-le-stockage-de-donnees-volumineuses

ID	Name	Age	Department
10	Bean	27	Production
11	Willis	61	Finance
12	Ferguson	58	Production

Figure 1. – Exemple de table classique

La structure typique d'un fichier csv sera orientée vers les lignes:

Figure 2. – Structure en lignes

a structure typique d'un fichier csv sera orientée vers les colonnes:

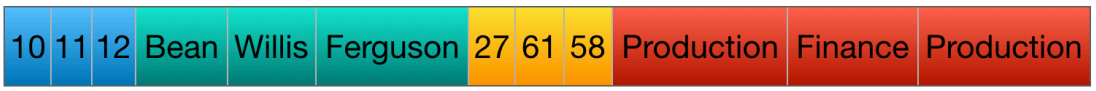

Figure 3. – Structure en colonnes

Format	Extension	Importation (Package)	Exportation (Package)	rio ?
Feather	.feather	**arrow**	**arrow**	oui
Parquet	.parquet	**arrow**	**arrow**	oui

Lecture/Ecriture à partir du format Feather

```
# Ecriture
arrow::write_feather(Wages,here("data/Wages.feather")) # Ecriture des
données (.feather)
Wages_feather <- arrow::read_feather(file=here("data/Wages.feather"),
                 col_select=c("exp","bluecol","sex","ed","lwage"))
# Lecture des données dans r
head(Wages_feather)   # Affichage des données
```

```
> # A tibble: 6 × 5
>     exp bluecol sex      ed lwage
>   <int> <fct>   <fct> <int> <dbl>
> 1     3 no      male      9  5.56
> 2     4 no      male      9  5.72
> 3     5 no      male      9  6.00
> 4     6 no      male      9  6.00
> 5     7 no      male      9  6.06
> 6     8 no      male      9  6.17
```

Le même exercice sera réalisé à partir de rio

```
rio::export(Wages,here("data/Wages_rio.feather"))
Wages_rio <-  rio::import(here("data/Wages_rio.feather"))
head(Wages_rio)
```

```
>   exp wks bluecol ind south smsa married  sex union ed black   lwage
> 1   3  32      no   0   yes   no     yes male    no  9    no 5.56068
> 2   4  43      no   0   yes   no     yes male    no  9    no 5.72031
> 3   5  40      no   0   yes   no     yes male    no  9    no 5.99645
> 4   6  39      no   0   yes   no     yes male    no  9    no 5.99645
> 5   7  42      no   1   yes   no     yes male    no  9    no 6.06146
> 6   8  35      no   1   yes   no     yes male    no  9    no 6.17379
```

```
str(Wages_rio,list.lenght=3)
```

```
> 'data.frame': 4165 obs. of  12 variables:
>  $ exp    : int  3 4 5 6 7 8 9 30 31 32 ...
>  $ wks    : int  32 43 40 39 42 35 32 34 27 33 ...
>  $ bluecol: Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 2 2 2 ...
>  $ ind    : int  0 0 0 0 1 1 1 0 0 1 ...
```

```
>  $ south  : Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 1 1 1 ...
>  $ smsa   : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
>  $ married: Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...
>  $ sex    : Factor w/ 2 levels "male","female": 1 1 1 1 1 1 1 1 1 1 ...
>  $ union  : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 2 ...
>  $ ed     : int  9 9 9 9 9 9 9 11 11 11 ...
>  $ black  : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
>  $ lwage  : num  5.56 5.72 6 6 6.06 ...
```

Lecture/Ecriture à partir du format Parquet

```
# Ecriture
arrow::write_parquet(Wages,here("data/Wages.parquet"))
# Ecriture des données (.parquet)
Wages_parquet <- arrow::read_parquet(
  file=here("data/Wages.parquet"),
  col_select=c("exp","bluecol","sex","ed","lwage"))
# Lecture des données dans r
head(Wages_parquet)  # Affichage des données
```

```
> # A tibble: 6 × 5
>     exp bluecol sex      ed lwage
>   <int> <fct>   <fct> <int> <dbl>
> 1     3 no      male      9  5.56
> 2     4 no      male      9  5.72
> 3     5 no      male      9  6.00
> 4     6 no      male      9  6.00
> 5     7 no      male      9  6.06
> 6     8 no      male      9  6.17
```

Le même exercice sera réalisé à partir de rio

```
rio::export(Wages,here("data/Wages_rio.parquet"))
Wages_rio <-  rio::import(here("data/Wages_rio.parquet"),
              col_select=c("exp","bluecol","sex","ed","lwage"))
head(Wages_rio)
```

```
>   exp bluecol  sex ed   lwage
> 1   3      no male  9 5.56068
> 2   4      no male  9 5.72031
> 3   5      no male  9 5.99645
> 4   6      no male  9 5.99645
> 5   7      no male  9 6.06146
> 6   8      no male  9 6.17379
```

```
str(Wages_rio,list.lenght=3)
```

```
> 'data.frame': 4165 obs. of  5 variables:
>  $ exp    : int  3 4 5 6 7 8 9 30 31 32 ...
>  $ bluecol: Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 2 2 2 ...
>  $ sex    : Factor w/ 2 levels "male","female": 1 1 1 1 1 1 1 1 1 1 ...
>  $ ed     : int  9 9 9 9 9 9 9 11 11 11 ...
>  $ lwage  : num  5.56 5.72 6 6 6.06 ...
```

1.4.c Fichiers textes

Les fichiers textes sont simplement des caractères lisible avec les colonnes séparées par un délimiteur. Quatre options sont possibles selon le format du fichier à lire:

Types	décimal	délimiteur
delim	locale()	delim
csv	.	,
csv2	,	;
tsv	locale()	tab

De nombreuses librairies permettent de lire des données en format texte.

Nous utilisons le fichier binaire “"bigfile.rds". Il s’agit d’un fichier réels de 17872 lignes et 373 variable qui contient du texte et des nombres. Le contenu de ce fichier sera explicité dans la partie consacrée à SAS (nyts).

```
# Lecture sous dataframe du fichier
bf <- fake_ticket_client(vol=50000,n = 2000,seed = 2811,local = c( "fr_FR"))
str(bf)
```

```
> tibble [50,000 × 25] (S3: tbl_df/tbl/data.frame)
>  $ ref              : chr [1:50000] "DOSS-KJSH-009518" "DOSS-ERWS-035357"
"DOSS-EGBF-040728" "DOSS-QVDE-032652" ...
>  $ num_client        : chr [1:50000] "1" "1" "1" "1" ...
>  $ prenom          : chr [1:50000] "Gérard" "Gérard" "Gérard" "Gérard" ...
>  $ nom                : chr [1:50000] "Fontainé" "Fontainé" "Fontainé"
"Fontainé" ...
>  $ job            : chr [1:50000] "Esthéticien" "Esthéticien" "Esthéticien"
"Esthéticien" ...
>  $ age               : num [1:50000] 62 62 62 62 62 62 63 63 63 NA ...
>  $ region            : chr [1:50000] "Aquitaine" "Aquitaine" "Aquitaine"
"Aquitaine" ...
>  $ id_dpt            : chr [1:50000] "47" "47" "47" "47" ...
>  $ departement        : chr [1:50000] "Lot-et-Garonne" "Lot-et-Garonne"
"Lot-et-Garonne" "Lot-et-Garonne" ...
>  $ gestionnaire_cb   : chr [1:50000] "VISA 13 digit" "VISA 13 digit" "VISA
13 digit" "VISA 13 digit" ...
>  $ nom_complet       : chr [1:50000] "Gérard Fontainé" "Gérard Fontainé"
"Gérard Fontainé" "Gérard Fontainé" ...
>  $ entry_date         : POSIXct[1:50000], format: "2017-03-18 19:16:00"
"2017-03-18 19:16:00" ...
```

```
>  $ points_fidelite    : num [1:50000] 2773 2773 2773 2773 2773 ...
>  $ priorite_encodee   : num [1:50000] 3 3 3 3 3 3 3 3 3 2 ...
>  $ priorite           : Factor w/ 4 levels "Bronze","Silver",..: 3 3 3 3 3
3 3 3 3 2 ...
>  $ timestamp         : Date[1:50000], format: "2017-04-05" "2017-05-23" ...
>  $ annee              : num [1:50000] 2017 2017 2017 2017 2017 ...
>  $ mois               : num [1:50000] 4 5 5 10 11 12 6 7 7 7 ...
>  $ jour               : int [1:50000] 5 23 29 30 7 28 25 12 17 22 ...
>  $ pris_en_charge     : chr [1:50000] "Non" "Non" "Oui" "Oui" ...
>  $ pris_en_charge_code: int [1:50000] 0 0 1 1 1 1 1 1 1 1 ...
>  $ type               : chr [1:50000] "Ligne" "Ligne" "Ligne" "Ligne" ...
>  $ type_encoded       : int [1:50000] 3 3 3 3 3 2 2 1 1 3 ...
>  $ etat              : Factor w/ 5 levels "En cours","Attente confirmation
client",..: 5 5 2 4 3 2 2 5 3 2 ...
>  $ source_appel       : Factor w/ 4 levels "Local","France",..: 2 1 4 4 4
2 4 3 4 4 ...
```

```
length(bf)
```

```
> [1] 25
```

On sauvegarde ce fichier sous forme d'un fichier compressé qui nous sera utile poure le suite.

```
# Lecture sous dataframe du fichierdu fichier
save(bf,file=here("data/bf0.rds"))
file.info(here("data/bf0.rds"))$size
```

```
> [1] 1881025
```

On remarque la taille fichier qui est de `file.info(here("data/bigfile.rds"))$size`. Nous allons écrire ce fichier sous la forme d'un fichier texte. Nous avons le choix entre des fonctions de base et des fonctions proposées par les packages

Package	Extension	Importation	Exportation	rio ?
Base	.csv	read.csv2	write.csv2	Non
readr	.csv	read_csv2	write_csv2	Non
data.table	.csv	fread	fwrite	Oui
vroom	.csv	read.csv2	write.csv2	Non

Ecriture avec les fonctions de base de R soit `write.csv2` et `read.csv2`. Nous adoptons la version francophone de CSV soit CSV2 avec des colonnes séparées par des points virgules et le délimiteur décimal est la virgule

```
benchmark(write.csv2=write.csv2(bf,here("data/bf1.csv")),
          readr_write_csv=readr::write_csv2(bf,here("data/bf2.csv")),
```

```
          data.table_fwrite=data.table::fwrite(bf,here("data/bf3.csv")),
          rio_export=rio::export(bf,here("data/bf4.csv")),
          vroom_vroom_write=vroom::vroom_write(bf,here("data/bf5.csv")),
          replications=1,
                       columns=c('test','user.self','sys.self',  'elapsed',
'replications'),
              order=NULL
)
```

```
>               test user.self sys.self elapsed replications
> 1        write.csv2     0.309    0.013   0.322            1
> 2   readr_write_csv     0.161    0.031   0.128            1
> 3 data.table_fwrite     0.027    0.002   0.030            1
> 4        rio_export     0.029    0.002   0.032            1
> 5 vroom_vroom_write     0.176    0.124   0.086            1
```

```
fichiers_liste <- paste0("data/bf", 1:5, ".csv")
bench_size <- data.frame(Fichiers=fichiers_liste)
bench_size$Fonction <- c("base csv","readr","data.table","rio","vroom")
bench_size$Taille <- file.size(bench_size$Fichiers)
bench_size
```

```
>       Fichiers   Fonction Taille
> 1 data/bf1.csv   base csv     NA
> 2 data/bf2.csv      readr     NA
> 3 data/bf3.csv data.table     NA
> 4 data/bf4.csv        rio     NA
> 5 data/bf5.csv      vroom     NA
```

La taille des fichiers est relativement similaire avec les différents formats de fichiers texte. Cependant le temps d'écriture et de lecture est fort variable. Nous utilisons la fonction `system.time()` pour mesurer le temps écoulé.

Pour la lecture:

les performance sont variables avec la fonction vroom() qui est toujours la plus rapide data.-table est le plus rapide

```
benchmark(read.csv2=read.csv2(here("data/bf1.csv")),
         readr_read_csv=readr::read_csv(here("data/bf2.csv"),show_col_types
= FALSE),
          data.table_fread=data.table::fread(here("data/bf3.csv")),
          rio_import=rio::import(here("data/bf4.csv")),
             vroom_vroom=vroom::vroom(here("data/bf5.csv"),show_col_types =
FALSE),
          replications=1,
                       columns=c('test','user.self','sys.self',  'elapsed',
'replications'),
```

```
              order=NULL
)
```

```
>               test user.self sys.self elapsed replications
> 1        read.csv2     0.175    0.005   0.180            1
> 2   readr_read_csv     0.099    0.002   0.032            1
> 3 data.table_fread     0.044    0.001   0.046            1
> 4       rio_import     0.044    0.001   0.046            1
> 5      vroom_vroom     0.086    0.003   0.013            1
```

1.4.d Comparaison fichiers textes et compressés

Nous allon comparer le format csv le plus performant (vromm) avec les formats compressés (rds, feather et parquet)

```
benchmark(csv=vroom::vroom_write(bf,here("data/bf.csv")) ,
          rds=export(bf,here("data/bf.rds")) ,
          feather=export(bf,here("data/bf.feather")) ,
          parquet=export(bf,here("data/bf.parquet")),
          replications=1,
              columns=c('test','user.self','sys.self', 'elapsed',
'replications'),
              order=NULL
)
```

```
>      test user.self sys.self elapsed replications
> 1     csv     0.171    0.112   0.086            1
> 2     rds     0.318    0.002   0.319            1
> 3 feather     0.033    0.002   0.019            1
> 4 parquet     0.044    0.002   0.041            1
```

On constate en écriture que le fichier **rds** est le plus lent suivi du **csv** , les autres formats sont proches, le plus rapide est clairement **feather**

```
bench_size <- data.frame(Fichiers=c("data/bf.csv",
                "data/bf.feather",
                "data/bf.rds",
                "data/bf.parquet"))
bench_size$Fonction <- c("vroom","arrow feather","rds","arrow parquet")
bench_size$Taille <- file.size(bench_size$Fichiers)
bench_size
```

```
>          Fichiers      Fonction Taille
> 1     data/bf.csv         vroom     NA
> 2 data/bf.feather arrow feather     NA
> 3     data/bf.rds           rds     NA
> 4 data/bf.parquet arrow parquet     NA
```

On constate pour les tailles que le fichier **csv** est le plus imposant suivi du format **feather**, les autres format sont nettement plus compressés, le plus léger est **parquet**

```
benchmark(vroom_csv=import(here("data/bf.csv")) ,
          rds=import(here("data/bf.rds")) ,
          feather=import(here("data/bf.feather")) ,
          parquet=import(here("data/bf.parquet")),
          replications=1,
              columns=c('test','user.self','sys.self', 'elapsed',
'replications'),
              order=NULL
)
```

```
>        test user.self sys.self elapsed replications
> 1 vroom_csv     0.043    0.002   0.045            1
> 2       rds     0.110    0.001   0.111            1
> 3   feather     0.006    0.003   0.004            1
> 4   parquet     0.015    0.002   0.004            1
```

On constate en lecture que le fichier **rds** est le plus lent suivi du **csv** , les autres formats sont les plus rapides.

La différence entre le binaire et le csv sera la taille et la vitesse de lecture

- Si on travaille sur de petits fichiers cela n'a pas d'importance.
- Sur de gros fichiers, les avantages en terme de taille et de vitesse sont importants pour les fichiers compressés. Les fichiers universels sont plus efficaces en taille et en vitesse. Le format feather est à privilégier en vitesse et le format parquet est efficace en taille.

1.4.e Fichiers de tableurs (.xls,.xlsx)

Le format tableur est souvent utile pour transmettre ou recevoir des données. Deux formats sont possibles soit le format ancien (.xls) ou le format actual (.xlsx). Pour lire un format ancien, `readxl` est le plus adapté.. Si nous travaillons avec des formats plus modernes, le package `openxlsx` est mieux adapté et beaucoup plus flexible pour le formatage des données. Le package `rio` simplifie l'usage de ce package au prix d'une flexibilité plus faible

Format	Extension	Importation (Package)	Exportation (Package)	rio ?
Excel	.xls	**readxl**	non	non
Excel	.xlsx	**openxlsx**	**openxlsx**	Oui

Nous reprenons les données sur les salaires pour illustrer les opérations: L'objectif sera de séparer la base de données en deux groupes (hommes et femmes) et d'écrire les bases dans deux feuilles séparées d'un classeur

On commence par séparer les deux groupes :

```
data(Wages)
```

```
> Warning in data(Wages): data set 'Wages' not found
```

```
Wages_m <- Wages[Wages$sex=="male",]            # Sélection des hommes
Wages_f <- Wages[Wages$sex=="female",]          # Sélection des femmes
```

> **!** Important
>
> L'écriture `Wages[sex=="male",]` n'est pas valable car le nom `sex` n'est pas défini auparavant

Par exemple nous écrivons dans le nouveau format avec

```
# Exriture des données
wb <- openxlsx::createWorkbook(title="Wages_split")             # Création
du tableur vide
openxlsx::addWorksheet(wb,"Wage_M_s")
openxlsx::addWorksheet(wb,"Wage_F_s")
openxlsx::writeData(wb, "Wage_M_s", Wages_m,
  rowNames = TRUE, startCol = "C", startRow = 3,
  borders = "surrounding", borderColour = "black")
openxlsx::writeData(wb, "Wage_F_s", Wages_f,
  rowNames = TRUE, startCol = "C", startRow = 3,
  borders = "surrounding", borderColour = "red")
openxlsx::saveWorkbook(wb, file = "data/Wages.xlsx",
             overwrite = TRUE)
# Lecture des données
Wage_M_s     <-     openxlsx::readWorkbook(xlsxFile     =     here("data/
Wages.xlsx"),sheet="Wage_M_s")
Wage_F_s     <-     openxlsx::readWorkbook(xlsxFile     =     here("data/
Wages.xlsx"),sheet="Wage_F_s")
str(Wage_M_s)
```

```
> 'data.frame': 3696 obs. of  12 variables:
>  $ exp    : num  3 4 5 6 7 8 9 30 31 32 ...
>  $ wks    : num  32 43 40 39 42 35 32 34 27 33 ...
>  $ bluecol: chr  "no" "no" "no" "no" ...
>  $ ind    : num  0 0 0 0 1 1 1 0 0 1 ...
>  $ south  : chr  "yes" "yes" "yes" "yes" ...
>  $ smsa   : chr  "no" "no" "no" "no" ...
>  $ married: chr  "yes" "yes" "yes" "yes" ...
>  $ sex    : chr  "male" "male" "male" "male" ...
>  $ union  : chr  "no" "no" "no" "no" ...
>  $ ed     : num  9 9 9 9 9 9 11 11 11 ...
>  $ black  : chr  "no" "no" "no" "no" ...
>  $ lwage  : num  5.56 5.72 6 6 6.06 ...
```

```
str(Wage_F_s)
```

```
> 'data.frame': 469 obs. of  12 variables:
>  $ exp    : num  31 32 33 34 35 36 37 9 10 11 ...
>  $ wks    : num  52 46 46 49 44 52 46 50 46 48 ...
>  $ bluecol: chr  "yes" "yes" "yes" "yes" ...
>  $ ind    : num  0 0 0 0 0 0 0 0 0 0 ...
>  $ south  : chr  "no" "no" "no" "no" ...
>  $ smsa   : chr  "yes" "yes" "yes" "yes" ...
>  $ married: chr  "no" "no" "no" "no" ...
>  $ sex    : chr  "female" "female" "female" "female" ...
>  $ union  : chr  "no" "no" "no" "no" ...
>  $ ed     : num  10 10 10 10 10 10 10 14 14 14 ...
>  $ black  : chr  "yes" "yes" "yes" "yes" ...
>  $ lwage  : num  6.16 6.24 6.3 6.36 6.47 ...
```

La même opération avec `rio`, beaucoup plus compact mais moins flexible

```
# Ecriture
rio::export(list(Wage_M_s = Wages_m, Wage_F_s = Wages_f), file="data/
Wages.xlsx",overwrite=TRUE)
# Lecture
Wages_i <- rio::import_list(which=c("Wage_M_s","Wage_F_s"),file="data/
Wages.xlsx")
str(Wages_i$Wage_M_s)
```

```
> 'data.frame': 3696 obs. of  12 variables:
>  $ exp    : num  3 4 5 6 7 8 9 30 31 32 ...
>  $ wks    : num  32 43 40 39 42 35 32 34 27 33 ...
>  $ bluecol: chr  "no" "no" "no" "no" ...
>  $ ind    : num  0 0 0 0 1 1 1 0 0 1 ...
>  $ south  : chr  "yes" "yes" "yes" "yes" ...
>  $ smsa   : chr  "no" "no" "no" "no" ...
>  $ married: chr  "yes" "yes" "yes" "yes" ...
>  $ sex    : chr  "male" "male" "male" "male" ...
>  $ union  : chr  "no" "no" "no" "no" ...
>  $ ed     : num  9 9 9 9 9 9 11 11 11 ...
>  $ black  : chr  "no" "no" "no" "no" ...
>  $ lwage  : num  5.56 5.72 6 6 6.06 ...
```

```
str(Wages_i$Wage_F_s)
```

```
> 'data.frame': 469 obs. of  12 variables:
>  $ exp    : num  31 32 33 34 35 36 37 9 10 11 ...
>  $ wks    : num  52 46 46 49 44 52 46 50 46 48 ...
>  $ bluecol: chr  "yes" "yes" "yes" "yes" ...
>  $ ind    : num  0 0 0 0 0 0 0 0 0 0 ...
>  $ south  : chr  "no" "no" "no" "no" ...
>  $ smsa   : chr  "yes" "yes" "yes" "yes" ...
>  $ married: chr  "no" "no" "no" "no" ...
>  $ sex    : chr  "female" "female" "female" "female" ...
```

```
>  $ union  : chr  "no" "no" "no" "no" ...
>  $ ed     : num  10 10 10 10 10 10 10 14 14 14 ...
>  $ black  : chr  "yes" "yes" "yes" "yes" ...
>  $ lwage  : num  6.16 6.24 6.3 6.36 6.47 ...
```

Utilisation du format ancien

Opération inverse de récupération des données:

```
# Lecture
Wages_m <- readxl::read_xls(here("data/Wages.xls"),sheet="Wage_M_s")
Wages_f <- readxl::read_xls(here("data/Wages.xls"),sheet="Wage_F_s")
Wages_xls <- rbind(Wages_m,Wages_f)                # recréation des données
str(Wages)                                    # Vérification
```

```
> 'data.frame': 4165 obs. of  12 variables:
>  $ exp    : int  3 4 5 6 7 8 9 30 31 32 ...
>  $ wks    : int  32 43 40 39 42 35 32 34 27 33 ...
>  $ bluecol: Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 2 2 2 ...
>  $ ind    : int  0 0 0 0 1 1 1 0 0 1 ...
>  $ south  : Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 1 1 1 ...
>  $ smsa   : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
>  $ married: Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...
>  $ sex    : Factor w/ 2 levels "male","female": 1 1 1 1 1 1 1 1 1 1 ...
>  $ union  : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 2 ...
>  $ ed     : int  9 9 9 9 9 9 9 11 11 11 ...
>  $ black  : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
>  $ lwage  : num  5.56 5.72 6 6 6.06 ...
```

```
str(Wages_xls)                                    # Vérification
```

```
> tibble [4,165 × 12] (S3: tbl_df/tbl/data.frame)
>  $ exp    : num [1:4165] 3 4 5 6 7 8 9 30 31 32 ...
>  $ wks    : num [1:4165] 32 43 40 39 42 35 32 34 27 33 ...
>  $ bluecol: chr [1:4165] "no" "no" "no" "no" ...
>  $ ind    : num [1:4165] 0 0 0 0 1 1 1 0 0 1 ...
>  $ south  : chr [1:4165] "yes" "yes" "yes" "yes" ...
>  $ smsa   : chr [1:4165] "no" "no" "no" "no" ...
>  $ married: chr [1:4165] "yes" "yes" "yes" "yes" ...
>  $ sex    : chr [1:4165] "male" "male" "male" "male" ...
>  $ union  : chr [1:4165] "no" "no" "no" "no" ...
>  $ ed     : num [1:4165] 9 9 9 9 9 9 9 11 11 11 ...
>  $ black  : chr [1:4165] "no" "no" "no" "no" ...
>  $ lwage  : num [1:4165] 5.56 5.72 6 6 6.06 ...
```

1.4.f Compatibilité SAS,SPSS et Stata

Pour ces formats particuliers , nous avons deux options soit utiliser le package ***heaven*** soit le package ***rio***

Format	Extension	Importation	Exportation	rio ?
SAS	.sas7bdat	**read_sas**	**write_sas**	Oui
SAS XPORT	.xpt	**read_xpt**	**write_xpt**	Oui
SPSS	.sav	**read_sav**	**write_sav**	Oui
SPSS (compressed)	.zsav	**read_sav**	**write_sav**	Oui
SPSS Portable	.por	**read_por**		Oui
Stata	.dta	**read_dta**	**write_dta**	Oui

On dispose sur le www de nombreux fichiers au format sas qui comportent souvent des informations fort détaillées sur les données. Les fichiers sur les enquêtes ou sondages sont souvent de cette nature. Nous prenons comme exemple l'enquête nyts (National Youth Tabacco Survey) du CDC (Center for Diseases control and prevention) voir: https://www.cdc.gov/tobacco/data_statistics/surveys/nyts/index.html. Base de donnée de sondage typique avec 17872 personnes interrogées répondant à 373 questions.

Les fichiers se composent de deux parties, une partie pour les données (extension .sas7bdat) et une partie pour le format (extension .sas7bcat). Le package `haven' nous permet de lire facilement ces données et de récupérer l'information (certaines opérations complémentaires seront parfois nécessaires).

```
nyts <- haven::read_sas(data_file    = here("data/nyts2017.sas7bdat"),
                        catalog_file = here("data/formats.sas7bcat"))
nyts[1:10,1:3]
```

```
> # A tibble: 10 × 3
>    Q1                 Q2         Q3
>    <chr+lbl>          <chr+lbl>  <chr+lbl>
>  1 05 [13 years old] 2 [Female] 2 [7th]
>  2 04 [12 years old] 2 [Female] 2 [7th]
>  3 04 [12 years old] 2 [Female] 2 [7th]
>  4 04 [12 years old] 2 [Female] 2 [7th]
>  5 05 [13 years old] 2 [Female] 2 [7th]
>  6 04 [12 years old] 2 [Female] 1 [6th]
>  7 03 [11 years old] 2 [Female] 1 [6th]
>  8 03 [11 years old] 2 [Female] 1 [6th]
>  9 03 [11 years old] 2 [Female] 1 [6th]
> 10 04 [12 years old] 2 [Female] 1 [6th]
```

```
rio::export(nyts,here("data/nyts.rds"))
```

Il est possible d'utiliser `rio` pour lire les données (pas le catalogue)

```
nyts <- rio::import(file    = here("data/nyts2017.sas7bdat"))
nyts[1:10,1:3]
```

```
>    Q1 Q2 Q3
> 1  05  2  2
> 2  04  2  2
> 3  04  2  2
> 4  04  2  2
> 5  05  2  2
> 6  04  2  1
> 7  03  2  1
> 8  03  2  1
> 9  03  2  1
> 10 04  2  1
```