S02: Import de données et indexation

Analyse de données quantitatives avec R

Samuel Coavoux

Importer des données

Localiser ses données

Notion de working directory

Le working directory est le répertoire de l'ordinateur considéré par la session courante de R comme sa "base". C'est là qu'il va aller chercher les fichiers lorsqu'on lui demande d'importer des données. C'est par rapport à ce répertoire qu'il définit les chemins relatifs.

`getwd()` renvoie le working directory actuel. `setwd("PATH")` permet de fixer le working directory.

ATTENTION: `setwd()` ne doit pas être utilisé avec Rstudio. En effet, le working directory est fixé, avec Rstudio, à la racine du **projet**.

```
getwd()
```

```
## [1] "/home/scoavoux/Dropbox/Cours/R@ENS"
```

Chemins relatifs et absolus

En informatique, on appelle:

- **absolu** le chemin vers un fichier qui part de la racine de l'ordinateur. Par exemple:
  - unix (linux ou mac): /home/user/Documents/R/data.csv
  - windows: c:\\Users\user\documents\R\data.csv
- **relatif** un chemin vers un fichier qui part du **répertoire actuel** (en l'occurence du working directory):
  - ./data/data.csv (ici, . signifie répertoire actuel)

Les chemins relatifs sont **toujours préférables** parce qu'ils sont plus pérennes: si vous copiez le dossier sur un autre ordinateur ou à un autre endroit, ils fonctionneront tant que la structure interne du répertoire ne change pas.

Chemins: bonnes pratiques

Nous allons faire en sorte de:

- toujours travailler dans un projet Rstudio (de sorte que le working directory est fixe)
- toujours localiser nos fichier par rapport à ce working directory

Personnellement, je crée toujours un repertoire `data` dans le répertoire du projet Rstudio, où j'enregistre tous les sets de données que je vais employer dans l'analyse.

Repérer le format de données

En gros, il y a deux grandes familles de format de données, que l'on repère principalement à leur extension:

- les données en texte brut (généralement .txt, .csv, .dlm)
- les données dans un format binaire, généralement propres à un logiciel:
  - R : .RData
  - SAS : .sasb7dat
  - STATA : .dta
  - SPSS : .sav, .por

Que faut-il utiliser

.RData => fonction `load()`

Texte brut => famille de fonctions `read.*()`

Autre format => regarder dans les packages `foreign()` (R-base) et `haven()` (hadleyverse)

La famille read

Import de données en R-base

La famille `read.*()` est un ensemble de fonctions pour lire les données au format texte. La fonction de base est `read.table()`. Toutes les autres fonctions sont simplement des variations, avec des arguments par défauts qui diffèrent quelque peu.

```
read.table(file, header=FALSE, sep = "", quote = "\"'",
           dec = ".")
```

fonction	header	sep	dec
read.csv	TRUE	,	.
read.csv2	TRUE	;	,
read.delim	TRUE	\t	.

Repérer le format précis des données

Si les données sont au format texte, il faut commencer par repérer à quoi elles ressemblent. On peut les ouvrir dans un éditeur de texte ; ou, si elles sont trop grandes, ne lire que le début du fichier.

Par exemple, sous unix (Mac/Linux):

```
head my_data.csv
```

Ce qu'il faut repérer

- est-ce que la première ligne contient le nom des variables ? Le plus souvent, oui. => argument `header`
- quel est le séparateur (le caractère qui sépare deux variables) ? Le plus souvent l'un de: `,` `;` `\t` (`tabulation`) => adapter argument `sep` de `read.table()` ou utiliser `read.csv()`, `read.csv2()` ou `read.delim()`
- s'il n'y a pas de séparateur visible, les données peuvent être au fixed width format => cf. `read.fwf()`
- quel est le signe de citation? Le plus souvent " => argument `quote`
- quel est le signe des décimales? Le plus souvent `.` => argument `dec`

Cas le plus fréquent: `read.csv()`

```
d <- read.csv("./data/data.csv",
              stringsAsFactors=FALSE)
```

Il vaut mieux toujours ajouter `stringsAsFactors=FALSE` et retransformer par la suite en factor les variables catégorielles.

Cas de fixed-width

Le format à largeur fixe est un format de donnée dans lesquelles chaque variable occupe un nombre de colonnes définie par avance.

```
V1V2 V3
01452478
01123236
02457124
```

Dans ce cas, on a besoin d'un vecteur indiquant la taille de chacune des colonnes. Ici:

```
d <- read.fwf("./data/data.fwf", widths = c(2, 3, 3))
```

Autres formats

Foreign

```
library(foreign)
```

- read.dta() stata
- read.spss() SPSS
- read.xport() SAS

Haven

Haven est habituellement plus performant que foreign, en particulier avec SAS.

```
library(haven)
```

- `read_stata()` STATA
- `read_spss()` SPSS
- `read_sas()` SAS

Exploration d'une base de données

- `str()`, `summary()`
- `dim()`, `length()`, `nrow()`, `ncol()`
- `head()`, `tail()`: afficher les cinq premières/cinq dernières lignes

Exercices

- Importer les bases de données de la série import présent dans le dossier data du répertoire github ;
- explorer les données: classes, structure, nombre de ligne, nombre de colonnes.

Indexation

Principe

Indexation

On appelle indexation l'opération qui consiste à sélectionner un sous-ensemble restreint des valeurs d'un vecteurs:

- seulement certaines valeurs d'un vecteur unidimensionnel ;
- seulement certains lignes ou certaines colonnes d'un vecteur à deux dimensions.

Il y a trois opérateurs d'indexation, dont deux que nous connaissons:

- `$`
- `[[`
- `[`

Charger les données

```
library(questionr)
data("hdv2003")
```

$ et [[

Dollar et crochet double permettent d'accéder aux objets stockés dans une liste (et par conséquent aux variables d'un data.frame). Fonctionne avec le **nom** de l'objet ou avec son index (uniquement pour [[).

$ et [[ne peuvent sélectionner qu'**un seul** objet stocké dans une liste. Ils renvoient un objet de la classe correspondant à l'objet sélectionné, et non une liste.

```
# hdv2003$age
head(hdv2003$age)
```

```
## [1] 28 23 59 34 71 35
```

$ et [[

```
# hdv2003[["age"]]
head(hdv2003[["age"]])
```

```
## [1] 28 23 59 34 71 35
```

$

```
names(hdv2003) # age est la variable n°2
```

```
##  [1] "id"             "age"
##  [3] "sexe"           "nivetud"
##  [5] "poids"          "occup"
##  [7] "qualif"         "freres.soeurs"
##  [9] "clso"           "relig"
## [11] "trav.imp"       "trav.satisf"
## [13] "hard.rock"      "lecture.bd"
## [15] "peche.chasse"   "cuisine"
## [17] "bricol"         "cinema"
## [19] "sport"          "heures.tv"
```

```
# hdv2003[[2]]
head(hdv2003[[2]])
```

[

Le crochet simple est un opérateur d'indexation qui permet de sélectionner deux ou plusieurs valeurs. Il fonctionne avec les **vecteurs unidimensionnels**, les **matrices** et les **data.frames**, mais **pas avec les listes**.

Il y a trois manières de sélectionner des valeurs avec `[` ; pour le moment, on se contente de l'index, c'est-à-dire la position dans le vecteur.

[avec un vecteur unidimensionnel

```
class(hdv2003$age)
```

```
## [1] "integer"
```

```
hdv2003$age[1] # age du premier individu
```

```
## [1] 28
```

```
hdv2003$age[c(1, 5, 7)] # age des individus 1, 5, 7
```

```
## [1] 28 71 60
```

```
hdv2003$age[1:10] # age des individus 1 à 10
```

```
##  [1] 28 23 59 34 71 35 60 47 20 28
```

[avec un vecteur pluridimensionnel

Si le vecteur a deux dimensions, [doit avoir deux arguments, séparés par une virgule.

```
class(hdv2003)
```

```
## [1] "data.frame"
```

```
# sélectionner les individus 1 à 10 (premier argument)
# sélectionner la variable n° 2 (deuxième argument)
hdv2003[1:10, 2]
```

```
##  [1] 28 23 59 34 71 35 60 47 20 28
```

[avec un vecteur pluridimensionnel

```
# Sélectionner les individus 1 à 5, les colonnes 1 à 3
hdv2003[1:5, 1:3]
```

```
##   id age  sexe
## 1  1  28 Femme
## 2  2  23 Femme
## 3  3  59 Homme
## 4  4  34 Homme
## 5  5  71 Femme
```

Sélection par le nom

Il est également possible de sélectionner des éléments par leur nom, en particulier pour les vecteurs d'un data.frame. Le vecteur nom peut avoir une taille supérieure à 1.

```
hdv2003[1:5, "age"]
```

```
## [1] 28 23 59 34 71
```

```
hdv2003[1:5, c("age", "sexe")]
```

```
##   age  sexe
## 1  28 Femme
## 2  23 Femme
## 3  59 Homme
## 4  34 Homme
## 5  71 Femme
```

Sélection vide

Si on laisse l'un des deux arguments vide, [sélectionne l'ensemble des lignes/colonnes.

```
# Toutes les lignes, seulement deux colonnes
# hdv2003[, c("age", "sexe")]
# Toutes les colonnes, seulement 5 lignes
# hdv2003[1:5, ]
```

Sélection logique

Le vecteur logique doit faire la même taille que la dimension à indexer.

```
# On réduit hdv à ses 5 premières colonnes
d <- hdv2003[1:5, c("age", "occup")]
d
```

```
##   age                    occup
## 1  28 Exerce une profession
## 2  23       Etudiant, eleve
## 3  59 Exerce une profession
## 4  34 Exerce une profession
## 5  71                Retraite
```

Sélection logique

```
d[c(TRUE, FALSE, FALSE, TRUE, FALSE), ]
```

```
##   age                    occup
## 1  28 Exerce une profession
## 4  34 Exerce une profession
```

Calculer une condition logique

Le plus souvent, cependant, le vecteur logique d'indexation est produit de façon programmatique: on n'écrit pas les TRUE et FALSE, mais on les calcule. Pour tester une condition, on dispose des opérateurs suivants:

- Vecteurs numériques ou integer: <, >, <=, >=, == (égal), != (différent de)
- Vecteurs character ou factor == (égal), != (différent), `%in%` (appartenant à un ensemble)
- Combinaison de vecteurs logiques: & (et), | (ou)

Condition logique: numérique

```
d$age > 30
```

```
## [1] FALSE FALSE  TRUE  TRUE  TRUE
```

```
d[d$age > 30, ]
```

```
##   age                 occup
## 3  59 Exerce une profession
## 4  34 Exerce une profession
## 5  71               Retraite
```

Condition logique: numérique, deux conditions

```
d$age > 30 & d$age < 70
```

```
## [1] FALSE FALSE  TRUE  TRUE FALSE
```

```
d[d$age > 30 & d$age < 70, ]
```

```
##   age                  occup
## 3  59 Exerce une profession
## 4  34 Exerce une profession
```

Condition logique: character

```
d$occup == "Exerce une profession"
```

```
## [1]  TRUE FALSE  TRUE  TRUE FALSE
```

```
d[d$occup == "Exerce une profession", ]
```

```
##   age                   occup
## 1  28 Exerce une profession
## 3  59 Exerce une profession
## 4  34 Exerce une profession
```

Condition logique: character

```
d$occup %in% c("Etudiant, eleve", "Retraite")
```

```
## [1] FALSE  TRUE FALSE FALSE  TRUE
```

```
d[d$occup %in% c("Etudiant, eleve", "Retraite"), ]
```

```
##   age           occup
## 2  23 Etudiant, eleve
## 5  71        Retraite
```

Exercices

On revient à la base de données `hdv2003` dans son intégralité.

- extraire les variables de pratique (celles auxquelles les enquêtés ont répondu par oui ou non) des enquêtés qui sont cadres.
- extraire toutes les variables des 15 derniers enquêtés
- extraire l'âge des personnes qui ne sont pas retraitées.

Recodage