

S02: Import de données, indexation et recodage

Analyse de données quantitatives avec R

Samuel Coavoux

1 Importer des données

2 Indexation

3 Recodage

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'occurrence 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 repertoire 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, .dlim)
- les données dans un format binaire, généralement propres à un logiciel:
 - R : .RData
 - SAS : .sasb7dat
 - STATA : .dta
 - SPSS : .sav, .por
 - Excel: .xls, .xlsx

Que faut-il utiliser

.RData => fonction `load()`

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

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

Excel

Pour lire des données directement depuis les formats binaires d'Excel, .xls et .xlsx, il existe plusieurs packages. Le plus performant pour le moment est `readxl`.

```
library(readxl)
read_excel("./data/myfile.xlsx")
# Si les données ne sont pas dans la première
# feuille du document, on peut préciser sheet
# l'argument header s'appelle col_names et il
# est TRUE par défaut
read_excel("./data/myfile.xlsx", sheet=2)
```

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

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 certaines 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
```

Cas des valeurs manquantes

Les valeurs NA ne peuvent pas être testées comme les autres. utiliser `var == NA` n'a pas de sens. Il faut employer `is.na()`.

```
# devrait renvoyer FALSE, FALSE, TRUE  
# à la place, renvoie NA, NA, NA  
c(12, 2, NA) == NA
```

```
## [1] NA NA NA
```

```
# on utilise donc is.na()  
is.na(c(12, 2, NA))
```

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

Négation d'un vecteur logique

Pour transformer tous les TRUE en FALSE dans un vecteur logique, il suffit d'employer la négation avec !:

```
c(TRUE, FALSE, TRUE)
```

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

```
!c(TRUE, FALSE, TRUE)
```

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

C'est particulièrement utile avec `is.na()` lorsque l'on souhaite sélectionner les valeurs *non manquantes*:

```
x <- c(12, 2, NA)  
x[!is.na(x)]
```

```
## [1] 12 2
```

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

Enquête emploi en continu

```
load("./data/eec2015.RData")
```

Transformer une classe de variable

La famille de fonction `as.*()` permet de transformer un objet d'une classe à l'autre. On utilisera principalement `as.numeric()`, `as.character()`, et `as.data.frame`.

as.character()

Fonctionne avec tous les types de vecteurs unidimensionnels.

- factor -> character : chaque valeur prend le level correspondant
- numeric/integer -> character : valeur numérique devient une chaîne de caractères (1 devient "1")
- logical -> character : TRUE devient "TRUE"

as.numeric

- character -> numeric : les valeurs qui sont composées uniquement de chiffres et de séparateur de décimale sont converties en numérique ; toutes les autres valeurs deviennent NA.

```
x <- c("12", "14", "50ml")  
as.numeric(x)
```

```
## Warning: NAs introduits lors de la conversion  
## automatique  
## [1] 12 14 NA
```

- logical -> numeric: TRUE devient 1, FALSE devient 0

as.numeric appliqué aux factor

factor -> numeric : le vecteur est converti, chaque valeur est l'index du level correspondant

```
x <- factor(c("CAP", "BEP", "BAC", "BEP"))  
levels(x)
```

```
## [1] "BAC" "BEP" "CAP"
```

```
as.numeric(x)
```

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

as.numeric appliqué aux factor (2)

Par conséquent, si un vecteur numérique est encodé par erreur sous la forme d'un factor, il faut faire attention à transformer le factor en character avant

```
x <- factor(c(12,17,20))  
as.numeric(x)
```

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

```
as.numeric(as.character(x))
```

```
## [1] 12 17 20
```

```
# Solution plus efficace  
as.numeric(levels(x))[x]
```

```
## [1] 12 17 20
```

Découper une variable numérique en classes

On utilise `cut()` avec comme argument `x` (le vecteur à découper) et `breaks` (soit un nombre de classes d'amplitude égales, soit les limites elles-mêmes).

```
head(cut(eec$HHCE, breaks = 6))
```

```
## [1] (30,40] (40,50] <NA>      <NA>      <NA>  
## [6] (40,50]  
## 6 Levels: (-0.06,10] (10,20] (20,30] ... (50,60.1]
```

```
head(cut(eec$HHCE, breaks = c(0, 20, 35, 42, 70)))
```

```
## [1] (20,35] (42,70] <NA>      <NA>      <NA>  
## [6] (35,42]  
## Levels: (0,20] (20,35] (35,42] (42,70]
```

cut()

Par défaut, l'intervalle est fermé à droite (changer avec `right = FALSE`) et exclut la valeur minimale (changer avec `include.lowest=TRUE`)

```
cut(c(0, 10, 20, 12), breaks=c(0, 10, 20))
```

```
## [1] <NA>      (0,10]  (10,20] (10,20]  
## Levels: (0,10] (10,20]
```

```
cut(c(0, 10, 20, 12),  
    breaks=c(0, 10, 20),  
    include.lowest = TRUE,  
    right = FALSE)
```

```
## [1] [0,10)  [10,20] [10,20] [10,20]  
## Levels: [0,10) [10,20]
```

cut()

```
eec$HHCE <- cut(eec$HHCE,  
               breaks = c(0, 20, 35, 42, 70),  
               labels = c("0-19", "20-34",  
                           "35-41", "42-70"))
```

Recoder des valeurs

```
eec$HORAIC[eec$HORAIC == "4"] <- NA
```

Renommer des modalités

```
eec$SEXE <- factor(eec$SEXE,  
                  levels = c("1", "2"),  
                  labels = c("Hommes", "Femmes"))
```


Réordonner des modalités

L'ordre dans lequel on déclare les levels et labels est l'ordre dans lequel les levels seront stockés. Pour le modifier, on peut employer à nouveau `factor()`

```
eec$SEXE <- factor(eec$SEXE,  
                  # changer l'ordre  
                  levels = c("Femmes", "Hommes"))
```

Regrouper des modalités

```
eec$diplome[eec$DIP11 %in% c("10", "11")] <- "Bac+3 et plus"  
eec$diplome[eec$DIP11 %in% c("30", "31", "33")] <- "Bac à h  
eec$diplome[eec$DIP11 %in% c("41", "42")] <- "Bac"  
eec$diplome[eec$DIP11 == "50"] <- "CAP, BEP"  
eec$diplome[eec$DIP11 %in% c("60", "70")] <- "BEPC, CEP"  
eec$diplome[eec$DIP11 == "71"] <- "Sans diplôme"
```

Regrouper des modalités

```
eec$diplome <- factor(eec$diplome,  
                      levels = c("Sans diplôme",  
                                "BEPC, CEP",  
                                "CAP, BEP",  
                                "Bac",  
                                "Bac à bac+2",  
                                "Bac+3 et plus"))
```

Regrouper des modalités: avec un dictionnaire

```
dic <- c(`71` = "Sans diplôme",  
        `70` = "BEPC, CEP",  
        `60` = "BEPC, CEP",  
        `50` = "CAP, BEP",  
        `41` = "Bac",  
        `42` = "Bac",  
        `33` = "Bac à bac+2",  
        `31` = "Bac à bac+2",  
        `30` = "Bac à bac+2",  
        `11` = "Bac+3 et plus",  
        `10` = "Bac+3 et plus")
```

Regrouper des modalités: avec un dictionnaire

```
eec$diplome_d <- dic[eec$DIP11]  
eec$diplome_d <- factor(eec$diplome_d,  
                        levels = unique(dic))
```

Concaténer deux vecteurs caractère

On utilise `paste()` pour coller deux vecteurs caractères l'un à l'autre ; l'argument `sep` permet de définir ce qui les séparera (par défaut un espace, " ") ; `paste0(x)` est défini comme `paste(x, sep="")`.

```
eec$dipl_sexe <- paste(eec$diplome_d, eec$SEXE)
```

Appariements

Appariements

Un appariement est une opération qui consiste à fusionner deux bases de données à partir d'un ou plusieurs identifiants communs. En base R, on utilise le plus souvent `merge()`. C'est particulièrement utile quand les données dont on dispose consistent en un ensemble de bases dont les observations sont de nature différentes.

Dans l'exemple suivant, on a une base d'ouvrages et une base d'écrivains. Tous les ouvrages n'ont pas leur écrivains dans l'autre bases ; tous les écrivains n'ont pas d'ouvrages ; certains en ont plusieurs. Les variables ne sont pas les mêmes : on ne dispose du pays que de l'écrivain, pas de l'ouvrage. Comment réinsérer le pays dans la base ouvrages?

```
ouvrages <- read.csv("../data/ouvrages.csv", stringsAsFactor = FALSE)  
ecrivains <- read.csv("../data/ecrivains.csv", stringsAsFactor = FALSE)
```


Ouvrages

ouvrages

##	auteur	titre	annee
## 1	Balzac	La peau de chagrin	1831
## 2	Woolf	Ms. Dalloway	1925
## 3	Woolf	To the lighthouse	1927
## 4	Sand	La mare au diable	1846

Écrivains

```
ecrivains
```

```
##      nom      prenom pays
## 1 Balzac    Honoré   FR
## 2 Woolf    Virginia  UK
## 3 Proust    Marcel   FR
```

Merge

```
merge(ouvrages, ecrivains,  
      by.x="auteur", by.y="nom")
```

##	auteur	titre	annee	prenom	pays
## 1	Balzac	La peau de chagrin	1831	Honoré	FR
## 2	Woolf	Ms. Dalloway	1925	Virginia	UK
## 3	Woolf	To the lighthouse	1927	Virginia	UK

Merge: noms identiques

```
names(ouvrages)[1] <- "nom"  
merge(ouvrages, ecrivains,  
      by="nom")
```

##	nom	titre	annee	prenom	pays
## 1	Balzac	La peau de chagrin	1831	Honoré	FR
## 2	Woolf	Ms. Dalloway	1925	Virginia	UK
## 3	Woolf	To the lighthouse	1927	Virginia	UK

Merge: toutes les lignes d'une base

```
merge(ouvrages, ecrivains,  
      by="nom",  
      all.x = TRUE)
```

##	nom	titre	annee	prenom	pays
## 1	Balzac	La peau de chagrin	1831	Honoré	FR
## 2	Sand	La mare au diable	1846	<NA>	<NA>
## 3	Woolf	Ms. Dalloway	1925	Virginia	UK
## 4	Woolf	To the lighthouse	1927	Virginia	UK

Ajouter des lignes à un data.frame

```
d1 <- hdv2003[1:10, ]  
d2 <- hdv2003[45:50, ]  
d3 <- rbind(d1, d2)
```

Exercices

- 1 Renommez les modalités de la variable catégorie socioprofessionnelle (1 chiffre).
- 2 **À partir de la variable NAFG088UN**, créez une variable de secteur économique prenant pour modalités “primaire”, “secondaire” et “tertiaire”.
- 3 Créez une variable “Nombre de modes de recherche active d’emploi” (pour simplifier, à calculer seulement à partir de trois modes différents)
- 4 Créez la variable “Nombre moyen d’heures travaillées par jour”