Initiation et prise en main rapide de R

P. Mercier - J. Nicolas - B. DeMalglaive - R. Dumas

Mai 2019

Sommaire

Séquencement

- 01 - Présentation de R
- 02 - Prise en main de données
- 03 - Premières statistiques
- 04 - Traitement de données
- 05 - Création de variables
- 06 - Statistiques par modalité
- 07 - Combiner plusieurs tables
- 08 - Stockage de données
- 09 - Réaliser des traitements statistiques
- 10 - Quelques éléments graphiques

01 - Présentation de R

AveRRRrrr…tissement

R n'est pas un logiciel statistique

… c'est un environnement de programmation
… doté d'un vrai langage de programmaton complet

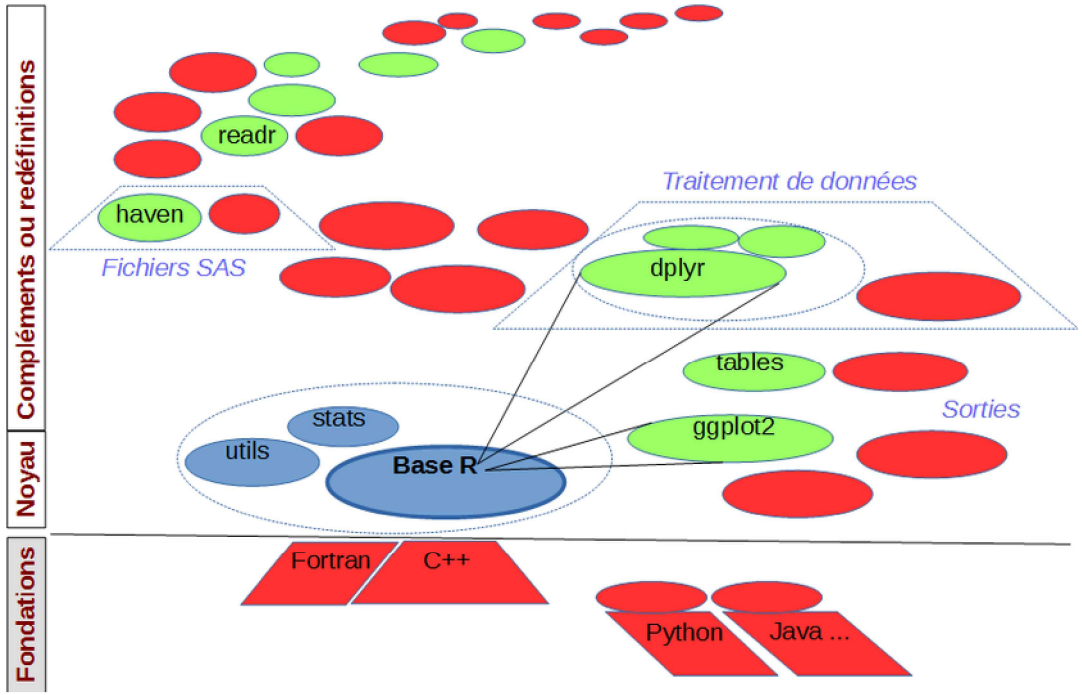
- scientifique
- multi-paradigme (plusieurs façons de parvenir à l'objectif)
- programmable

… efficace si on sait l'utiliser

Ce qui suit est destiné à ceux qui ont autre chose à faire qu'apprendre la programmation !

Principes généraux

Une galaxie de 13 000 packages et 220 000 fonctions



R vs SAS

Points forts

- Logiciel gratuit et opensource
- Nombreux packages scientifiques
- Rapidité des traitements (usage de la mémoire vive)
- Fonctions graphiques étendues

Point faibles

- Proche d'un langage informatique
- Problème de compatibilité des versions de R
- Calcul sur des grosses volumétries plus complexe

Le travail sous R: juste une question de mémoire

A l'inverse de SAS, R travaille sur une image des données à traiter chargée en mémoire vive:

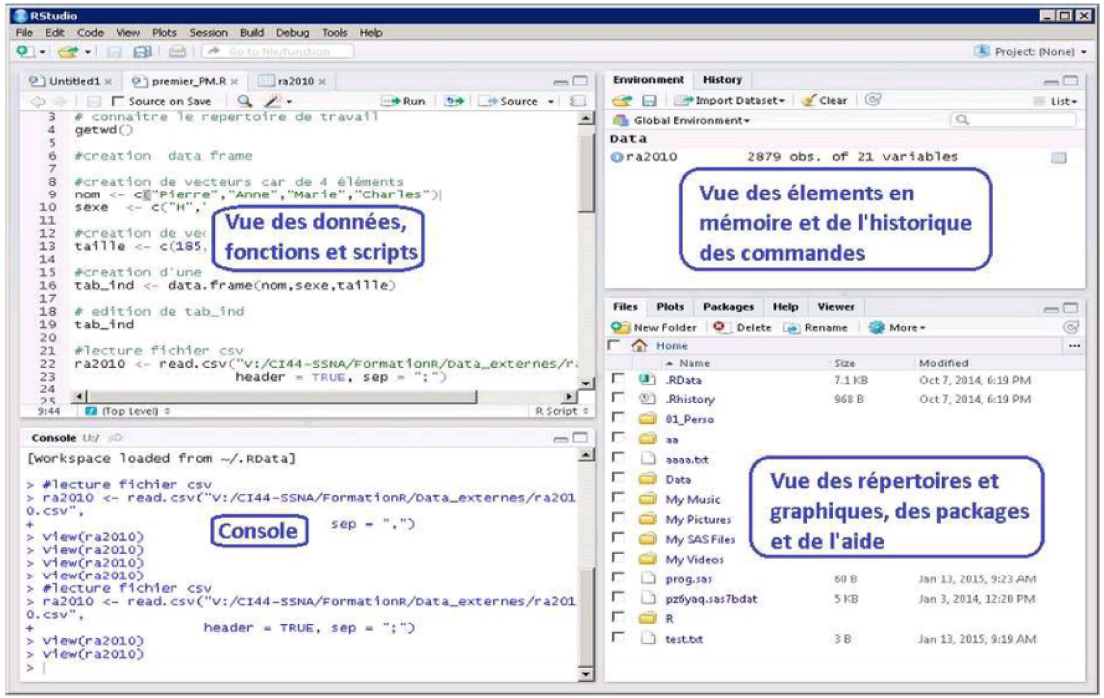
- il en découle la rapidité des traitements…
- … mais la taille de la mémoire devient décisive!

A l'Insee, trois environnements peuvent être mobilisés

- Le poste de travail individuel, limité en mémoire et en données accessibles
- AUS/LINUX ("Plateforme R"), où la mémoire utilisée s'adapte automatiquement avec une limite de 100GO
- AUS/windows, où on peut manuellement choisir la limite de la mémoire avec une limite de 100GO

L'interface graphique de RStudio

Un environnement de développement intégré (EDI)



Site de RStudio

Les 4 fenêtres de RStudio

Source (en haut à gauche)

- gestion des scripts
- visualisation des données

Console (en bas à gauche)

- visualisation de la log (journal d'exécution)
- visualisation des résultats d'exécution
- permet de tester des morceaux de programme

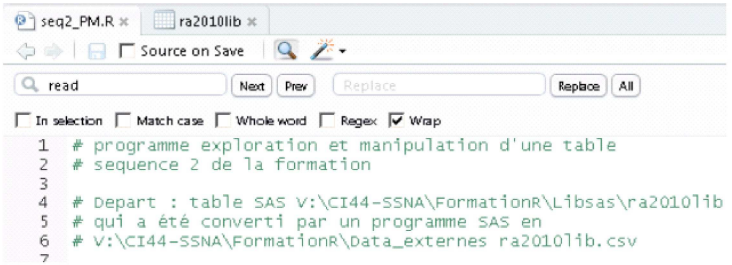
Environnement (en haut à droite)

- permet de gérer les objets
- accéder à l'historique d'exécution

Services (en bas à droite)

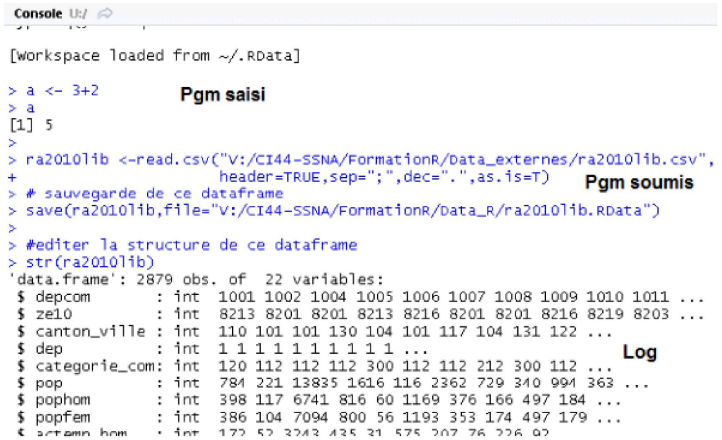
- accès aux fichiers
- visualisation des graphiques
- gestion des packages
- accès à l'aide ! (en anglais)

Fenêtre Source



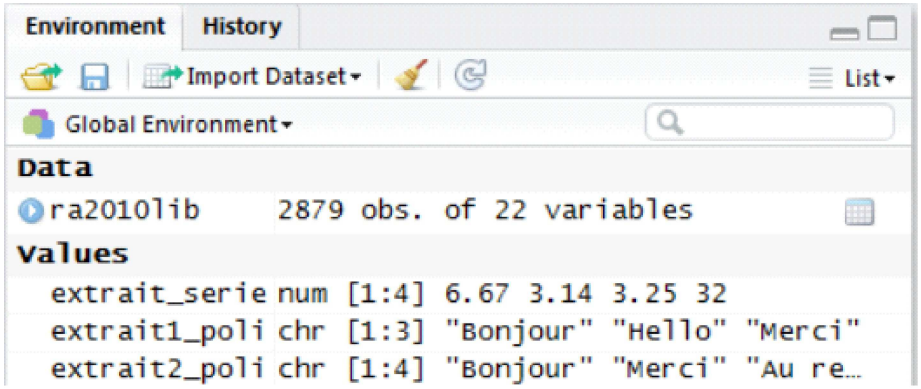
Pour soumettre en exécution une ligne ou un ensemble de lignes sélectionné: Ctrl + Enter

Fenêtre console de RStudio



- soumettre une commande par **Entrée**
- Aide sur une fonction: **?nom_de_fonction**
- Recherche dans l'aide **??mots-clés**
- Pour effacer le contenu de la console:
- menu Edit -> Clear console
- ou **Ctrl + L**

Fenêtre Environnement



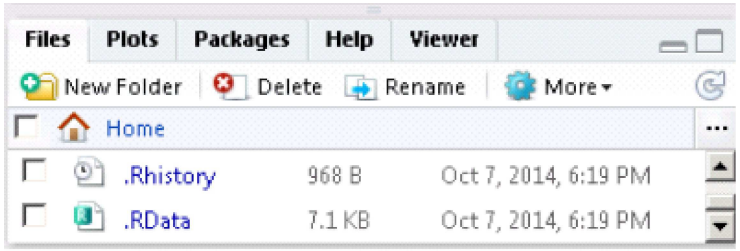
Cette fenêtre permet de :

- gérer les éléments en mémoire
- visualiser l'historique des commandes

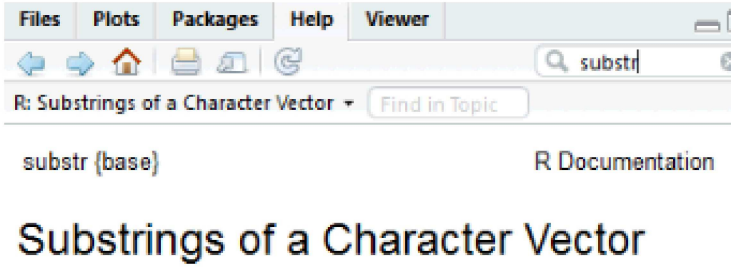
Avec les icônes, il est possible :

- d'importer des données
- de sauvegarder des données
- de vider des éléments en mémoire

Fenêtre Services



- Un onglet Files -> accéder à l'arborescence des fichiers
- Un onglet Help -> accéder à l'aide
- Un onglet Plots -> visualiser les graphiques



- Un onglet Packages -> gérer (installer/charger) les "paquets"

Répertoire de travail

R possède par défaut un répertoire de travail

Sous AUS, ce répertoire est la racine de l'espace personnel (U:/)

Pour changer le répertoire de travail:

il faut utiliser la commande **setwd("chemin du répertoire souhaité")**

Pour connaître le répertoire de travail :

il faut utiliser la commande **getwd()**

Les scripts (ou programmes)

Quand on programme, il est préférable d'écrire les commandes dans la fenêtre de scripts (sources), plutôt que dans la console.

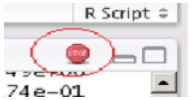
- Ces fichiers, appelés scripts, pourront être sauvegardés, et porteront l'extension **.R** .
- Pour créer un script : **File > New File > R script**
- Pour travailler avec un script existant : **File > Open File**
- Pour sauvegarder un script : **File > Save** ou **File > Save as**

Les scripts (ou programmes)

Pour exécuter des commandes d'un script, on se positionne sur la ligne ou bien on sélectionne l'ensemble des commandes à exécuter puis :

- **Code > Run Selected Line(s)**, ou
- **Ctrl + Enter**, ou
- 

.

Pour interrompre une exécution, on utilise le bouton **STOP** de la console : 

02 - Prise en main de données

Chargement de données sous R

.rds et .RData sont des formats de stockage natif à R

- plus rapide que des approches non natives
- permet de conserver des informations spécifiques à R (attributs de variables)

readRDS() permet de lire un seul objet d'extension .rds

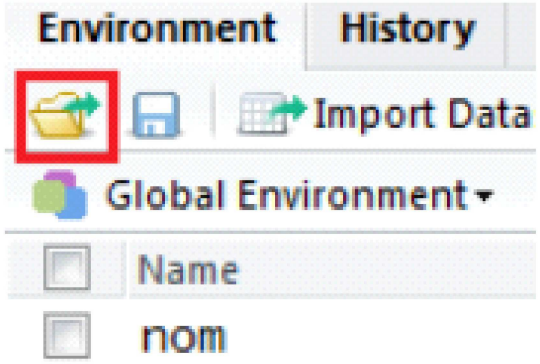
- le nom de stockage en mémoire sous R est choisi

load() permet de charger un ensemble d'objets , qui porte l'extension .Rdata

- les noms de stockage en mémoire sous R sont ceux présents dans le fichier .Rdata

Chargement de données sous R

L'interface de RStudio permet aussi de charger des objets



… mais uniquement d'extension .Rdata

Quelques règles sous R

R est sensible à la casse

Le nom des objets est assez libre

- … mais commence par une lettre
- usage possible des majuscules et minuscules
- usage possible des chiffres et underscore (_)
- il est déconseillé d'utiliser les accents

Les commentaires s'écrivent après le symbole #

Certains noms sont réservés à R (else, for, TRUE, FALSE,…)

Dans les chemins Windows, les \ sont remplacés par des /

Les objets R

R utilise des fonctions ou des opérateurs qui agissent sur des objets (vecteurs, matrices, etc.)

La création d'un objet peut se faire par affectation avec un des trois opérateurs "<-", "->" ou "=".

```
a <- 25 # crée l'objet a en lui donnant la valeur 25
x <- a  # x reçoit la valeur a
x = a   # x reçoit la valeur a
a -> x  # x reçoit la valeur a
```

Le symbole # permet d'empêcher l'interprétation d'une partie d'une ligne, et ainsi d'ajouter des commentaires dans un programme.

Les objets R - Affichage

Si un objet n'existe pas, l'affectation le crée. Sinon, l'affectation écrase la valeur précédente sans message d'avertissement! On affiche la valeur d'un objet x via la commande :

```
print(x)
```

```
## [1] 25
```

ou encore plus simplement :

```
x
```

```
## [1] 25
```

Les objets R - Suppression

Par défaut, R conserve en mémoire (vive) tous les objets créés lors de la session. Il est donc recommandé de régulièrement supprimer les objets non (ou plus) nécessaires à la programmation.

Pour connaître les objets de la session il est possible de regarder la fenêtre d'environnement de Rstudio ou bien par la fonction :

```
ls()
```

```
## [1] "a"                "install_and_load" "x"
```

Pour supprimer l'objet a, il suffit de taper :

```
rm(a)
```

La preuve :

```
ls()
```

```
## [1] "install_and_load" "x"
```

Les objets R - Leurs types

Avant d'aborder les différents objets de R, il faut connaître les principaux modes ou types de ces objets :

- null (objet vide) : **NULL**
- logical (booléen) : **TRUE**, **FALSE** ou **T**, **F**
- numeric (nombre réel) : **1**, **2.37**, **pi**, **1e-10**
- complex (nombre complexe) : **2i**
- character (chaîne de caractères) : **"bonjour!"**

Pour connaître l'attribut mode ou type d'un objet x de R, il suffit de taper :

```
mode(x)
```

```
## [1] "numeric"
```

Les objets R - Tester ou Modifier le mode d'un objet R

Tester le mode :
is.null(x)
is.logical(x)
is.numeric(x)
is.complex(x)
is.character(x)

Modifier le mode :
as.null(x)
as.logical(x)
as.numeric(x)
as.complex(x)
as.character(x)

```
x <- as.character(x)
is.character(x)
```

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

Les vecteurs

Le vecteur est un objet dit atomique de R, c'est-à-dire d'un type unique (null, logique, etc.). Un vecteur est composé d'un ensemble de valeurs appelées composantes, coordonnées ou éléments.

L'attribut "longueur", obtenu avec la fonction **length**, donne le nombre d'éléments du vecteur.

```
length(x)
```

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

Les vecteurs

La fonction **is.vector()** permet de savoir si un objet est un vecteur :

```
is.vector(x)
```

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

Les vecteurs numériques

Pour construire un vecteur, il existe différentes méthodes. Par exemple, par la fonction dite collecteur :

```
x <- c(2,5.1,-13.7,0.1) # par la fonction c() dite collecteur
x
```

```
## [1]   2.0   5.1 -13.7   0.1
```

```
length(x)
```

```
## [1] 4
```

Les vecteurs numériques

Mais également par séquence :

```
1:6 # par l'opérateur : de séquence
seq(1,6, by=0.5) # par la fonction seq() avec l'argument by
seq(1,6, length=5) # par la fonction seq() de séquence avec l'argument length
rep(1,6) # par la fonction rep() de séquence
```

any() et all()

```
x <- c(1,4,6,-4)
any(x > 0)
```

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

```
all(x[1:3]>0)
```

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

Les fonctions sur le type numérique

```
x <- pi
round(x,2)
```

```
## [1] 3.14
```

```
floor(x)
```

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

```
ceiling(x)
```

```
## [1] 4
```

Liste de fonctions mathématiques: "abs", "sign", "sqrt", "trunc", "cummax", "cummin", "cumprod", "cumsum", "log", "log10", "log2", "log1p", "acos", "acosh", "asin", "asinh", "atan", "atanh", "exp", "expm1", "cos", "cosh", "cospi", "sin", "sinh", "sinpi", "tan", "tanh", "tanpi", "gamma", "lgamma", "digamma", "trigamma"

Le type character

```
a <- "Insee"
b <- "Mesurer pour comprendre"
```

Quelques fonctions utiles:

```
paste0(a,b) # [1] "InseeMesurer pour comprendre"
paste(a,b) # [1] "Insee Mesurer pour comprendre"
paste(a,b,sep=" - ") # [1] Insee - Mesurer pour comprendre
paste0(a,seq(2010,2025,5)) # [1] "Insee2010" "Insee2015" "Insee2020" "Insee2025"
substr(b,2,12) # [1] "esurer pour"
strsplit(b," ") # [1] "Mesurer"    "pour"       "comprendre"
```

Les vecteurs caractères

Il est possible de créer des vecteurs de caractères de la même façon

```
x <- c("A","B","C")
x
```

```
## [1] "A" "B" "C"
```

```
x <- rep("A", 5)
x
```

```
## [1] "A" "A" "A" "A" "A"
```

Sélection d'une partie d'un vecteur

Elle s'opère grâce à l'opérateur [] et un vecteur de sélection :

```
x <- c(80,102,127,92,54) # création d'un vecteur x de longueur 5
x[3] # renvoi le 3ème élément de x
```

```
## [1] 127
```

```
x[-c(1,2)] # renvoi x sans les deux premiers éléments
```

```
## [1] 127  92  54
```

La valeur manquante

Pour différentes raisons, il se peut lors d'une expérience ou d'une enquête que certaines données ne soient pas collectées. R les note **NA** pour "Not Available".

Pour tester la présence de valeurs manquantes: *is.na()*

```
x<- c(5, NA, 10, 9, NA)
is.na(x)
```

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

Dans les calculs, il est possible de les exclure: na.rm = TRUE

```
mean(x) # renvoie NA
```

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

```
mean(x, na.rm = TRUE) # renvoie 8
```

```
## [1] 8
```

Les facteurs

Les facteurs permettent de manipuler des données qualitatives. Un facteur possède les attributs **length** et **mode** mais aussi **levels** qui renvoit les modalités du facteur.

```
Sexe <- factor(c("H","F","F","H","H","F","F","F","H","F"))
levels(Sexe)
```

```
## [1] "F" "H"
```

```
table(Sexe)
```

```
## Sexe
## F H
## 6 4
```

Les data.frames

Ce sont des listes particulières dont les composantes sont de même longueur, mais de modes qui peuvent être différents.

```
C1 <- seq(1,6, length=5)
C2 <- factor(c("A","B","B","A","C"))
DF <- data.frame(C1,C2)
DF
```

```
##     C1 C2
## 1 1.00  A
## 2 2.25  B
## 3 3.50  B
## 4 4.75  A
## 5 6.00  C
```

Les data.frames

Un data.frame est un tableau de données…

…composé de vecteurs…

- vecteur = variable = colonne

…qui ont tous le même nombre d'éléments

- nombre d'éléments = nombre d'observations = nombre de lignes

Les data.frames

```
DF[2,] # renvoie la deuxième ligne du data.frame DF
```

```
##     C1 C2
## 2 2.25  B
```

```
DF[,1] # renvoie la 1ere colonne du data.frame DF
```

```
## [1] 1.00 2.25 3.50 4.75 6.00
```

```
DF$C1 # renvoie la colonne (variable) nommée C1
```

```
## [1] 1.00 2.25 3.50 4.75 6.00
```

```
DF$C2 # renvoie la colonne (variable) nommée C2
```

```
## [1] A B B A C
## Levels: A B C
```

Les data.frames

Pour connaître la structure d'un data.frame

```
str(DF)
```

```
## 'data.frame':    5 obs. of  2 variables:
##  $ C1: num  1 2.25 3.5 4.75 6
##  $ C2: Factor w/ 3 levels "A","B","C": 1 2 2 1 3
```

On peut aussi connaître les dimensions de la table par

```
dim(DF)
```

```
## [1] 5 2
```

Les data.frames

Et les noms des colonnes par

```
colnames(DF)
```

```
## [1] "C1" "C2"
```

str(), dim(), colnames() sont des fonctions…

```
names(DF) <- c("Moyenne", "Type")
str(DF)
```

```
## 'data.frame':    5 obs. of  2 variables:
##  $ Moyenne: num  1 2.25 3.5 4.75 6
##  $ Type   : Factor w/ 3 levels "A","B","C": 1 2 2 1 3
```

Les fonctions

R construit autour de ces outils appelés "fonctions" qui:

- réalisent une tâche particulière
- à partir d'informations qu'on leur fournit (arguments)
- et produisent un résultat

Exemple de la fonction dim()

- tâche: connaître la dimension d'une table de données
- l'argument est la table en question
- le résultat est le nombre de lignes et de colonnes
- **Les fonctions sous R ont été nommées par leurs concepteurs...**

L'aide sur les fonctions

Dans la console:

- help("nom-fonction")
- ?nom-fonction

Dans la fenêtre Help de RStudio

On obtient un résultat seulement si le "package" contenant cette fonction est chargé

Une option de la fonction help()

```
#help("dim", try.all.packages = TRUE)
```

Les packages

Un package (paquet ou bilbiothèque) est un ensemble de programmes R qui permet d'augmenter les fonctionnalités.

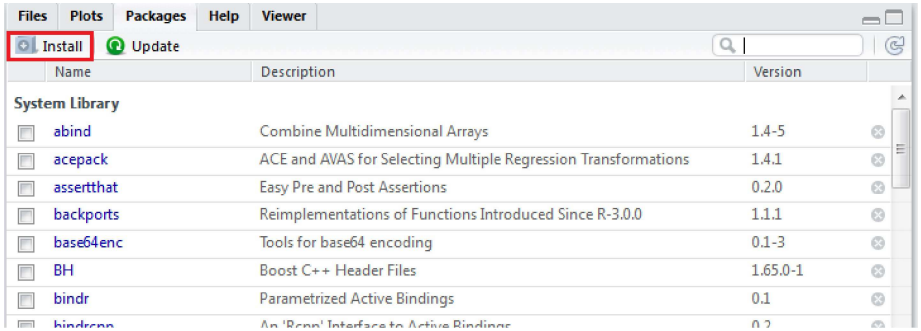
Il existe plus de 200 000 packages. Un certain nombre d'entre eux sont jugés indispensables et sont de fait présents lors de l'installation de R.
Les autres peuvent être téléchargés librement sur le réseau CRAN (The Comprehensive R Archive Network).

Pour utiliser un package, il faut qu'il soit installé puis chargé en mémoire.
Cela peut se faire soit par la programmation soit par l'interface utilisateur.

Les packages - installation

Elle peut se faire de différentes façons:

- par ligne de programme **install.packages("FactoMineR", dependencies = T)**
- par le menu **Tools > Install Packages...**
- par l'onglet **Packages** de la fenêtre en bas à droite de Rstudio

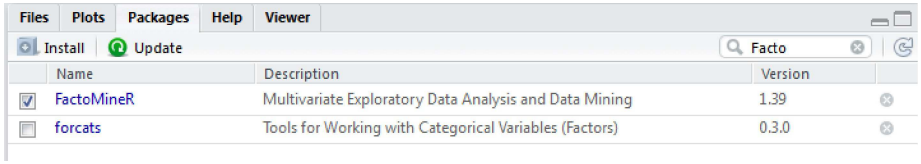


Les packages - chargement

Une fois installés, les packages doivent être chargés pour être utilisables.

Cela est possible par :

- une ligne de commande **library("FactoMineR")**
- la fenêtre Rstudio en cochant le package :



Quelques opérateurs pour info

Les opérateurs arithmétiques : + - * / ^

Les opérateurs de comparaison

- **==** pour "égal à"
- **!=** pour "différent de"
- **>** pour "supérieur"
- **>=** pour "supérieur ou égal à"
- **<** pour "inférieur"
- **<=** pour "inférieur ou égal à"
- **%in%** pour "appartenance à un objet" (retourne un booléen)

Les opérateurs logiques

- **!** pour la négation *(également la fonction Negate() pour les fonctions)*
- **&** pour "et"
- **|** pour "ou"

03 - Premières statistiques

Le package dplyr

Dans la formation, nous allons utiliser le package dplyr

- **dplyr** est un package développé par RStudio.
- Ce package permet d'utiliser une syntaxe propre à la manipulation de données.

dplyr

- identifie les principales fonctions nécessaires à la manipulation de données;
- améliore les performances de mise en mémoire des données en utilisant C++;
- utilise la même interface de travail que les données soient des data-frames ou d'autres types de bases de données

Comment installer dplyr ?

dplyr est disponible dans le répertoire mis à disposition par l'Insee via RStudio.

La première chose à faire est donc d'installer le package.

```
install.packages("dplyr")
```

On peut alors le charger :

```
library("dplyr")
```

Enchainer les fonctions: le *Pipe* %>%

La syntaxe classique de l'enchainement de fonctions:

- l'ordre chronologique est inversé
- digérer(manger(cuisiner(recette_tajine)))

La syntaxe de dplyr avec le *Pipe*

- l'ordre chronologique est respecté
- recette_tajine %>% cuisiner %>% manger %>% digérer

Raccourci clavier: Ctrl + Shift + M générère le %>%

Premières statistiques: *summarise()*

```
tab_naissances <- readRDS(file = "../RData/naissances2016.rds") # chargement des données
```

On souhaite connaître le nombre de lignes de la table:

```
tab_naissances %>% summarise(nb_naissances=n())
```

```
## # A tibble: 1 x 1
##   nb_naissances
##           <int>
## 1         79286
```

On souhaite connaître la moyenne d'age de la mère:

```
tab_naissances %>% summarise(age_moyen_mere=mean(agemere))
```

```
## # A tibble: 1 x 1
##   age_moyen_mere
##            <dbl>
## 1       30.70156
```

Premières statistiques: *summarise()*

summarise() propose un ensemble de statistiques

- nombre, somme: **n()**, **n_distinct()**, **sum()**
- tendance centrale: **mean()**, **median()**
- dispersion: **min()**, **max()**, **quantile()**, **sd()**, **var()**
- position: **first()**, **last()**, **nth()**

Premières statistiques: distribution

Calculons les quartiles sur la variable *agemere*

```
tab_naissances %>% summarise(
  q1_mere = quantile(agemere, 0.25),
  median_mere = quantile(agemere, 0.5),
  q3_mere = quantile(agemere, 0.75)
)
```

```
## # A tibble: 1 x 3
##   q1_mere median_mere q3_mere
##     <dbl>       <dbl>   <dbl>
## 1      27          31      34
```

Et les valeurs extrêmes:

```
tab_naissances %>% summarise(min_age_mere = min(agemere),
                             max_age__mere = max(agemere))
```

```
## # A tibble: 1 x 2
##   min_age_mere max_age__mere
##          <dbl>         <dbl>
## 1           13            55
```

Premières statistiques: comptages

Calculons le nombre de naissances multiples, avec la variable *jumeau*:

- jumeau vaut "0" si naissance unique
- jumeau vaut "1" si naissance multiple

```
tab_naissances %>% summarise(nb_naissances_multiples = sum(jumeau == "1"))
```

```
## # A tibble: 1 x 1
##   nb_naissances_multiples
##                     <int>
## 1                    2688
```

R réalise en fait ici la conversion des booléens:

- vrai vaut 1
- faux vaut 0

04 - Traitement de données

Liste de modalités: *distinct()*

```
tab_naissances %>% distinct(regnais)
```

Il s'agit de l'équivalent de la syntaxe SQL

```
select distinct regnais from tab_naissances
```

Extraire des colonnes *select()*

Une sélection par les noms

```
# une liste de colonnes de notre choix
tab_naissances %>% select(sexe, regnais, agemere, csp) %>% head(3)
```

```
## # A tibble: 3 x 4
##    sexe regnais agemere   csp
##   <chr>   <chr>   <dbl> <chr>
## 1     1      84      33    46
## 2     1      84      38    65
## 3     2      84      36    63
```

```
# toutes les colonnes, réordonnées
tab_naissances %>% select(sexe, regnais, agemere, csp, everything()) %>% head(3)
#supprimer des colonnes
tab_naissances %>% select(-sexe, regnais, agemere, csp, everything()) %>% head(3)
```

- outre everything(), le package **dplyr** continet de nombreuses fonctions utiles: starts_with(), ends_with(), contains(), matches(), num_range(), one_of(), group_cols()

Extraire des colonnes *select()*

Une sélection par les positions

```
# une liste de colonnes spécifiées par leur position
tab_naissances %>% select(10,5,42,71) %>% head(2)
```

```
## # A tibble: 2 x 4
##    sexe regnais agemere   csp
##   <chr>   <chr>   <dbl> <chr>
## 1     1      84      33    46
## 2     1      84      38    65
```

Une sélection par type de colonne avec *select_if*

```
# toutes les variables numériques
tab_naissances %>% select_if(is.numeric) %>% head(2)
```

```
## # A tibble: 2 x 10
##   dureconj durerecm durerecp agemere agexactm agepere agexactp durecmar
##      <dbl>    <dbl>    <dbl>   <dbl>    <dbl>   <dbl>    <dbl>    <dbl>
## 1       95       NA       NA      33       33      33       33       NA
## 2       NA       NA       NA      38       38      41       40       12
## # ... with 2 more variables: nbenf <dbl>, durecevp <dbl>
```

Renommer des colonnes *rename()*

```
# une liste de colonnes spécifiées par leur position
tab_naissances %>% select(sexe , regnais , agemere ,csp) %>% rename(Genre=sexe, region=regnais)
```

```
## # A tibble: 79,286 x 4
##    Genre region agemere   csp
##    <chr>  <chr>   <dbl> <chr>
##  1     1     84      33    46
##  2     1     84      38    65
##  3     2     84      36    63
##  4     2     84      31    48
##  5     1     84      38    21
##  6     1     84      27    63
##  7     2     84      33    38
##  8     1     84      30    63
##  9     2     84      41    63
## 10     2     84      31    63
## # ... with 79,276 more rows
```

ATTENTION: L'ordre dans rename() est NOUVEAU NOM = ANCIEN NOM

Sélectionner des individus *filter()*

On souhaite travailler sur un sous-ensemble:

- les naissances en Nouvelle-Aquitaine (code de région: 75)
- de mères françaises (indicateur de nationalité: 1)

```
tab_naissances %>%
  filter(regnais == "75" &
           indnatm == "1") %>%
  summarise(nb_naissances_NA = n())
```

```
## # A tibble: 1 x 1
##   nb_naissances_NA
##              <int>
## 1             5104
```

05 - Création de variables

Création de variables: *mutate()*

On peut créer une variable à partir d'une ou plusieurs autres

```
tab_naissances %>% mutate(jumeau2 = as.numeric(jumeau)) %>%
  select(sexe , regnais , agemere ,csp,jumeau2,jumeau) %>% head(2)
```

```
## # A tibble: 2 x 6
##    sexe regnais agemere   csp jumeau2 jumeau
##   <chr>   <chr>   <dbl> <chr>   <dbl>  <chr>
## 1     1      84      33    46       0      0
## 2     1      84      38    65       0      0
```

Ou modifier une variable (en réutilisant le même nom)

```
tab_naissances %>% mutate(jumeau = as.numeric(jumeau)) %>%
  select(sexe , regnais , agemere ,csp,jumeau) %>% head(2)
```

```
## # A tibble: 2 x 5
##    sexe regnais agemere   csp jumeau
##   <chr>   <chr>   <dbl> <chr>  <dbl>
## 1     1      84      33    46      0
## 2     1      84      38    65      0
```

Création de variables: *mutate()*

Il est possible de faire plusieurs opérations dans un appel à *mutate*

```
tab_naissances %>% mutate(jumeau2=jumeau, jumeau = as.numeric(jumeau)) %>%
  select(sexe , regnais , agemere ,csp,jumeau2,jumeau) %>% head(3)
```

```
## # A tibble: 3 x 6
##    sexe regnais agemere   csp jumeau2 jumeau
##   <chr>   <chr>   <dbl> <chr>   <chr>  <dbl>
## 1     1      84      33    46       0      0
## 2     1      84      38    65       0      0
## 3     2      84      36    63       0      0
```

Recoder des variables: *ifelse()*

Syntaxe de la fonction ifelse()

```
ifelse(condition, valeur_si_vrai, valeur_si_faux)
```

```
tab_naissances %>% mutate(sexe_ = ifelse(sexe == "1", "garçon", "fille")) %>%
  select(sexe , regnais , agemere ,csp,sexe_) %>% head(10)
```

```
## # A tibble: 10 x 5
##    sexe regnais agemere   csp  sexe_
##   <chr>   <chr>   <dbl> <chr>  <chr>
##  1     1      84      33    46 garçon
##  2     1      84      38    65 garçon
##  3     2      84      36    63  fille
##  4     2      84      31    48  fille
##  5     1      84      38    21 garçon
##  6     1      84      27    63 garçon
##  7     2      84      33    38  fille
##  8     1      84      30    63 garçon
##  9     2      84      41    63  fille
## 10     2      84      31    63  fille
```

Recoder des variables: *case_when()*

Syntaxe de la fonction case_when()

```
case_when(condition1 ~ valeur 1,

    condition2 ~ valeur2, ...

    TRUE ~ valeur_par_défaut)
```

Les conditions sont passées en revue jusqu'à donner TRUE

```
tab_naissances %>% mutate(sexe_ = case_when(sexe == "1" ~ "garçon",
                                            sexe == "2" ~ "fille",
                                            TRUE ~ "inconnu")) %>%
  select(sexe , regnais , agemere ,csp,sexe_) %>% head(5)
```

```
## # A tibble: 5 x 5
##    sexe regnais agemere   csp  sexe_
##   <chr>   <chr>   <dbl> <chr>  <chr>
## 1     1      84      33    46 garçon
## 2     1      84      38    65 garçon
## 3     2      84      36    63  fille
## 4     2      84      31    48  fille
## 5     1      84      38    21 garçon
```

Créer une variable sans conserver les autres: *transmute*

```
tab_naissances %>% transmute(datenais = paste(jnais,mnais,anais,sep="/")) %>% head(5)
```

```
## # A tibble: 5 x 1
##     datenais
##        <chr>
## 1 14/10/2016
## 2 08/07/2016
## 3 24/01/2016
## 4 21/02/2016
## 5 15/03/2016
```

06 - Statistiques par modalité

Statistiques par modalités: *group_by()*

```
tab_naissances %>% group_by(regnais) %>%
  summarise( nb_naissances = n(),
             nb_max_enfant_nes_par_accouchement = max(nbenf)) %>% head(10)
```

```
## # A tibble: 10 x 3
##    regnais nb_naissances nb_max_enfant_nes_par_accouchement
##      <chr>         <int>                              <dbl>
##  1      01           479                                  2
##  2      02           389                                  2
##  3      03           735                                  3
##  4      04          1405                                  3
##  5      06           946                                  2
##  6      11         18260                                  3
##  7      24          2643                                  2
##  8      27          2835                                  3
##  9      28          3597                                  3
## 10      32          7242                                  3
```

cela ne conserve que les variables ayant servi au regroupement, et les calculs

Statistiques par croisements: *group_by()*

```
tab_naissances %>% group_by(regnais, sexe) %>%
  summarise( nb_naissances = n(),
             nb_max_enfant_nes_par_accouchement = max(nbenf)) %>% head(10)
```

```
## # A tibble: 10 x 4
## # Groups:   regnais [5]
##    regnais  sexe nb_naissances nb_max_enfant_nes_par_accouchement
##      <chr> <chr>         <int>                               <dbl>
##  1      01     1           256                                   2
##  2      01     2           223                                   2
##  3      02     1           208                                   2
##  4      02     2           181                                   2
##  5      03     1           394                                   3
##  6      03     2           341                                   2
##  7      04     1           679                                   2
##  8      04     2           726                                   3
##  9      06     1           484                                   2
## 10      06     2           462                                   2
```

Trier: *arrange()*

Le top2 par sexe du nombre de naissance dans chaque région:

```
tab_naissances %>% group_by(sexe, regnais) %>%
  summarise( nb_naissances = n()) %>%
  arrange(sexe,desc(nb_naissances)) %>% top_n(2)
```

```
## # A tibble: 4 x 3
## # Groups:   sexe [2]
##    sexe regnais nb_naissances
##   <chr>   <chr>         <int>
## 1     1      11          9393
## 2     1      84          4818
## 3     2      11          8867
## 4     2      84          4493
```

Attention, la fonction **head(n)** donne les enregistrements de la table sans groupement

```
tab_naissances %>% group_by(sexe, regnais) %>% summarise(nb_naissances = n()) %>%
  arrange(sexe,desc(nb_naissances)) %>% head(2)
# Sexe            regnais            nb_naissances
# <chr>           <chr>             <int>
# 1               11                9393
# 1               84                4818
# 2 rows
```

Dégrouper: *ungroup()*

Ainsi, on récupère le département ayant le plus grand nombre de naissances, dans chaque région:

```
a <- tab_naissances %>% group_by(regnais, depnais) %>%
      summarise( nb_naissances = n()) %>%
      filter(nb_naissances == max(nb_naissances))
```

Pour maintenant, ne plus tenir compte du regroupement par région, et ainsi obtenir le département ayant le plus grand nombre de naissances France entière:

```
a %>% ungroup() %>%
  filter(nb_naissances == max(nb_naissances))
```

07 - Combiner plusieurs tables

Concaténer deux tables avec dplyr: *bind_rows()*

Il s'agit d'empiler les lignes d'une table derrière les lignes d'une autre.

```
load(file = "../RData/fillesgarcons.Rdata") #charge deux tables
# colnames(filles)
# colnames(garcons)
ensemble <- filles %>% bind_rows(garcons) # empile les lignes
ensemble
```

```
## # A tibble: 5 x 8
##     prenom poids taille salaire commune activite  sexe  poid
##      <chr> <dbl>  <dbl>   <dbl>   <chr>    <chr> <chr> <dbl>
## 1 Nathalie 68.23 172.62 1968.68   69123       F2     F    NA
## 2  Valérie 54.77 169.69 2345.00   2A041       L0     F    NA
## 3   Sylvie 57.35 154.31 1612.12   98818       R1     F    NA
## 4 Philippe    NA 183.50 1864.54   69123       R1     M 85.25
## 5      Bob    NA 173.45 2103.00   19110       A1     M 68.00
```

Attention aux noms des colonnes… Cela peut générer des valeurs manquantes!

Coller deux tables avec dplyr: *bind_cols()*

Il s'agit de juxtaposer des colonnes de deux tables dont le nombre de lignes est identique.

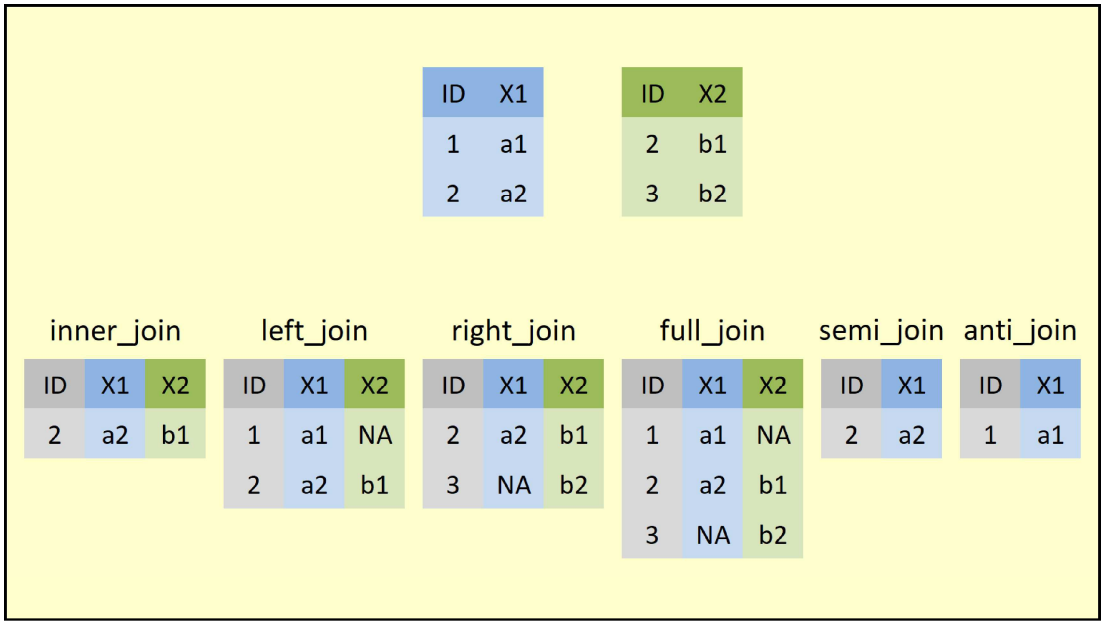
```
vec_alea <- data.frame(alea=runif(nrow(ensemble),1,10))
# une table d'une colonne avec autant de nombres aléatoires
# que d'individus dans la table "ensemble"

ensemble2 <-  ensemble %>% bind_cols(vec_alea) # colle les colonnes
ensemble2
```

```
## # A tibble: 5 x 9
##     prenom poids taille salaire commune activite  sexe  poid     alea
##      <chr> <dbl>  <dbl>   <dbl>   <chr>    <chr> <chr> <dbl>    <dbl>
## 1 Nathalie 68.23 172.62 1968.68   69123       F2     F    NA 7.225171
## 2  Valérie 54.77 169.69 2345.00   2A041       L0     F    NA 1.196379
## 3   Sylvie 57.35 154.31 1612.12   98818       R1     F    NA 8.647354
## 4 Philippe    NA 183.50 1864.54   69123       R1     M 85.25 7.989964
## 5      Bob    NA 173.45 2103.00   19110       A1     M 68.00 4.171247
```

Fusionner deux tables avec dplyr

Il s'agit de contrôler le résultat final… et pour cela nous disposons d'un ensemble de fonctions:



Fusionner deux tables avec dplyr

```
personnes <- data.frame(nom = c("Sylvie", "Sylvie", "Monique", "Gunter","Rayan", "Rayan"),
            voiture = c("Twingo", "Ferrari", "Scenic", "Lada","Twingo","Clio"))
voitures <- data.frame(voiture = c("Twingo", "Ferrari", "Clio", "Lada", "208"),
           vitesse = c("140", "280", "160", "85", "160"))
```

```
personnes
```

```
##       nom voiture
## 1  Sylvie  Twingo
## 2  Sylvie Ferrari
## 3 Monique  Scenic
## 4  Gunter    Lada
## 5   Rayan  Twingo
## 6   Rayan    Clio
```

```
voitures
```

```
##   voiture vitesse
## 1  Twingo     140
## 2 Ferrari     280
## 3    Clio     160
## 4    Lada      85
## 5     208     160
```

left_join()

on vient enrichir "personnes" par "voitures"

```
left_join(personnes,voitures,by="voiture")
```

```
##       nom voiture vitesse
## 1  Sylvie  Twingo     140
## 2  Sylvie Ferrari     280
## 3 Monique  Scenic    <NA>
## 4  Gunter    Lada      85
## 5   Rayan  Twingo     140
## 6   Rayan    Clio     160
```

Dans le cas du Scenic, nous n'avions pas de ligne correspondante dans "voitures".

left_join()

on vient enrichir "voitures" par "personnes"

```
left_join(voitures,personnes,by="voiture")
```

```
##   voiture vitesse    nom
## 1  Twingo     140 Sylvie
## 2  Twingo     140  Rayan
## 3 Ferrari     280 Sylvie
## 4    Clio     160  Rayan
## 5    Lada      85 Gunter
## 6     208     160   <NA>
```

Dans "personnes", aucune personne ne conduit de 208.

inner_join()

on souhaite conserver la partie commune des tables "voitures" et "personnes"

```
inner_join(voitures,personnes,by="voiture")
```

```
##   voiture vitesse    nom
## 1  Twingo     140 Sylvie
## 2  Twingo     140  Rayan
## 3 Ferrari     280 Sylvie
## 4    Clio     160  Rayan
## 5    Lada      85 Gunter
```

full_join()

on souhaite conserver l'ensemble des infos des tables "voitures" et "personnes"

```
full_join(voitures,personnes,by="voiture")
```

```
##   voiture vitesse     nom
## 1  Twingo     140  Sylvie
## 2  Twingo     140   Rayan
## 3 Ferrari     280  Sylvie
## 4    Clio     160   Rayan
## 5    Lada      85  Gunter
## 6     208     160    <NA>
## 7  Scenic    <NA> Monique
```

semi_join()

on conserve les lignes de "personnes" qui existent dans "voitures"

```
semi_join(personnes,voitures,by="voiture")
```

```
##      nom voiture
## 1 Sylvie  Twingo
## 2 Sylvie Ferrari
## 3 Gunter    Lada
## 4  Rayan  Twingo
## 5  Rayan    Clio
```

anti_join()

on conserve les lignes de "personnes" qui sont absentes dans "voitures"

```
anti_join(personnes,voitures,by="voiture")
```

```
##       nom voiture
## 1 Monique  Scenic
```

L'importance du by=

C'est la clé de fusion, et en son absence…

… R cherche les variables communes aux deux tables pour fabriquer une clé de fusion

avec une clé différente dans les deux tables

```
inner_join(df1,df2,by=c("a"="b"))
# a est une colonne de df1
# b est une colonne de df2
```

avec plusieurs clés

```
inner_join(df1,df2,by=(c("a","c")=c("b","d")))
# a et c colonnes de df1
# b et d colonnes de df2
```

08 - Stockage de données

Les fonctions natives de sauvegarde

La fonction *save()*

- elle permet de sauvegarder un ou plusieurs objets sous forme d'un seul fichier
- l'extension est .Rdata

```
save(objet1, objet2,...,file="ensemble.Rdata")
```

La fonction *saveRDS()*

- elle permet de sauvegarder un seul objet
- l'extension est .rds

```
saveRDS(objet1, file="objet1.rds")
```

Importer les types de fichiers courants: Le package *readODS*

- il permet d'importer et d'exporter des fichiers CALC
- le parsing est peu optimisé, il est plus pratique de passer par le format CSV

```
library(readODS) # read_ODS et write_ODS
data <- read_ods("../Data_nonR/Dep02.ods") #import de fichier CALC
data <- data %>% mutate(densite=Population/Superficie)

write_ods(data, "../Data_nonR/Dep02b.ods")#export de fichier CALC
```

Importer les types de fichiers courants: Le package *readxl*

- il permet d'importer des fichiers EXCEL
- le parsing est meilleur que celui du package *xlsx*
- pas de fonction d'export

```
library("readxl") #read_xls
data <- read_xls("../Data_nonR/cnaf.xls") #import de fichier EXCEL
```

Importer les types de fichiers courants: Le package *haven*

- il permet d'importer et d'exporter des fichiers SAS

```
library("haven") #read_sas
data <- read_sas("../Data_nonR/ra2010lib.sas7bdat") #import de fichier SAS
data <- data %>% mutate(chom = chom_hom + chom_fem)

write_sas(data,"../Data_nonR/ra2010lib2.sas7bdat")
```

Importer les types de fichiers courants: Le package *readr*

- il permet d'importer des fichiers CSV (Comma Separated Values)
- le parsing est meilleur que celui de la fonction RBase
- privilégier read_csv2 qui est adaptée aux conventions françaises des séparateurs

```
library("readr") #read_csv2 (préférable à la version de Rbase)
data <- read_csv2("../Data_nonR/dep01.csv")
data <- data %>%
  mutate(
    partchfem = round(100 * chom_fem / sum(chom_hom+chom_fem, na.rm = T), 1)
    )

write_csv(data, "../Data_nonR/dep01b.csv")


# avec du pipe, sans ecrire de variable
read_csv2("./Data_nonR/dep01.csv") %>%
  mutate(
    partchfem = round(100 * chom_fem / sum(chom_hom+chom_fem, na.rm = T), 1)
  ) %>%
  write_csv("./Data_nonR/dep01b.csv")
```

Un package à utiliser: fst

'Lightening Fast Serialization of Data Frames'

- c'est un format efficace pour transmettre une table de données avec des métadonnées
- une vitesse de chargement/déchargement beaucoup plus élevée
- une consommation d'espace disque moindre
- un accès possible à une portion de l'objet sauvegardé (utile pour les grosses bases)

```
install.packages("fst")
library("fst")
write_fst(objet,"objet.fst")
```

Un package à utiliser: fst

Connaître les métadonnées du fichier

```
metadata_fst("objet.fst")
```

Lire une table entière

```
read_fst("objet.fst")
```

Lire uniquement certaines lignes

```
read_fst("objet.fst", from = 1, to=100)
```

Lire uniquement certaines colonnes

```
read_fst("objet.fst", columns = c("col1","col2","col12"))
```

09 - Réaliser des traitements statistiques

Le package skimr

```
install.packages("skimr")
```

```
library("skimr")
iris %>% skim %>% kable
```

variable	type	stat	level	value	formatted
Sepal.Length	numeric	missing	.all	0.0000000	0
Sepal.Length	numeric	complete	.all	150.0000000	150
Sepal.Length	numeric	n	.all	150.0000000	150
Sepal.Length	numeric	mean	.all	5.8433333	5.84
Sepal.Length	numeric	sd	.all	0.8280661	0.83
Sepal.Length	numeric	p0	.all	4.3000000	4.3
Sepal.Length	numeric	p25	.all	5.1000000	5.1
Sepal.Length	numeric	p50	.all	5.8000000	5.8
l.Length	numeric	p75	.all	6.4000000	6.4
Sepal.Length	numeric	p100	.all	7.9000000	7.9

Le package skimr

```
iris %>% skim(Sepal.Length) %>% kable # sur une seule variable
```

variable	type	stat	level	value	formatted
Sepal.Length	numeric	missing	.all	0.0000000	0
Sepal.Length	numeric	complete	.all	150.0000000	150
Sepal.Length	numeric	n	.all	150.0000000	150
Sepal.Length	numeric	mean	.all	5.8433333	5.84
Sepal.Length	numeric	sd	.all	0.8280661	0.83
Sepal.Length	numeric	p0	.all	4.3000000	4.3
Sepal.Length	numeric	p25	.all	5.1000000	5.1
Sepal.Length	numeric	p50	.all	5.8000000	5.8
Sepal.Length	numeric	p75	.all	6.4000000	6.4
Sepal.Length	numeric	p100	.all	7.9000000	7.9
l.Length	numeric	hist	.all	NA	▂▇▅▇▆▅▂▂

Le package skimr

```
iris %>% group_by(Species) %>% skim() %>% kable # selon des groupes
```

Species	variable	type	stat	level	value	formatted
setosa	Sepal.Length	numeric	missing	.all	0.0000000	0
setosa	Sepal.Length	numeric	complete	.all	50.0000000	50
setosa	Sepal.Length	numeric	n	.all	50.0000000	50
setosa	Sepal.Length	numeric	mean	.all	5.0060000	5.01
setosa	Sepal.Length	numeric	sd	.all	0.3524897	0.35
setosa	Sepal.Length	numeric	p0	.all	4.3000000	4.3
setosa	Sepal.Length	numeric	p25	.all	4.8000000	4.8
setosa	Sepal.Length	numeric	p50	.all	5.0000000	5
setosa	Sepal.Length	numeric	p75	.all	5.2000000	5.2
setosa	Sepal.Length	numeric	p100	.all	5.8000000	5.8
sa	Sepal.Length	numeric	hist	.all	NA	

Le package skimr

Le résultat de la fonction *skim()* est en fait une table, avec des attributs…

```
iris %>% skim() %>% filter(stat=="mean") %>% arrange(desc(value))
```

```
## # A tibble: 4 x 6
##       variable    type  stat level    value formatted
##          <chr>   <chr> <chr> <chr>    <dbl>     <chr>
## 1 Sepal.Length numeric  mean  .all 5.843333      5.84
## 2 Petal.Length numeric  mean  .all 3.758000      3.76
## 3  Sepal.Width numeric  mean  .all 3.057333      3.06
## 4  Petal.Width numeric  mean  .all 1.199333       1.2
```

Réaliser un tri à plat

La fonction *table()* donne les effectifs par modalité d'une variable.

```
ra2010 <- readRDS("../RData/ra2010lib.rds")
effplat <- table(ra2010$dep, useNA="ifany")
effplat
```

```
##
##   01   07   26   38   42   69   73   74 <NA>
##  418  339  369  533  327  293  305  294    1
```

La fonction *cumsum()* appliquée à la table des effectifs donne des effectifs cumulés.

```
somcum <- cumsum(effplat)
somcum
```

```
##   01   07   26   38   42   69   73   74 <NA>
##  418  757 1126 1659 1986 2279 2584 2878 2879
```

Réaliser un tri à plat

Avec dplyr:

```
ra2010 %>%  group_by(dep) %>% summarise(nbcom=n()) #Tri à plat
```

```
ra2010 %>%  group_by(dep) %>% summarise(nbcom=n()) %>%
  mutate (cumul = cumsum(nbcom)) #Effectifs cummulés en supplément
```

```
## # A tibble: 9 x 3
##     dep nbcom cumul
##   <chr> <int> <int>
## 1    01   418   418
## 2    07   339   757
## 3    26   369  1126
## 4    38   533  1659
## 5    42   327  1986
## 6    69   293  2279
## 7    73   305  2584
## 8    74   294  2878
## 9  <NA>     1  2879
```

Réaliser un tri à plat

Avec dplyr et enjolivé:

```
ra2010 %>%  group_by(dep) %>% summarise(nbcom=n())%>% mutate (cumul = cumsum(nbcom)) %>%
 knitr::kable(col.names = c("Département","Effectif","Effectifs cumulés "))
```


```
ra2010 %>%  group_by(dep) %>% summarise(nbcom=n())%>% mutate (cumul = cumsum(nbcom)) %>%
 knitr::kable(col.names = c("Département","Effectif","Effectifs cumulés "))
```


Département	Effectif	Effectifs cumulés
01	418	418
07	339	757
26	369	1126
38	533	1659
42	327	1986
69	293	2279
73	305	2584
	294	2878

Réaliser un tri à plat

La fonction *prop.table()* donne les fréquences à partir de la table des effectifs.

```
freqplat <- round(prop.table(effplat) *100,2)
freqplat
```

```
##
##    01    07    26    38    42    69    73    74  <NA>
## 14.52 11.77 12.82 18.51 11.36 10.18 10.59 10.21  0.03
```

La fonction *addmargins()* permet d'ajouter la marge.

```
freqplat <- round(addmargins(prop.table(effplat)) *100,2)
freqplat
```

```
##
##     01     07     26     38     42     69     73     74   <NA>    Sum
##  14.52  11.77  12.82  18.51  11.36  10.18  10.59  10.21   0.03 100.00
```

Réaliser un tri à plat

Avec dplyr et enjolivé:

```
ra2010 %>% group_by(dep) %>% summarise(nbcom=n()) %>%
  mutate(pour_com = nbcom/ sum(nbcom) * 100) %>%
  knitr::kable(col.names = c("Département","Effectif","Part en % "))
```

Département	Effectif	Part en %
01	418	14.5189302
07	339	11.7749218
26	369	12.8169503
38	533	18.5133727
42	327	11.3581105
69	293	10.1771448
73	305	10.5939562
74	294	10.2118791
	1	0.0347343

Réaliser un croisement

La fonction *table()* permet de réaliser un croisement.

```
effcrois <- table(ra2010$dep, ra2010$categorie_com, useNA="ifany",
                  dnn = c("Département","Catégorie de commune"))
effcrois
```

```
##             Catégorie de commune
## Département 111 112 120 211 212 221 222 300 400
##         01    39 223  46   2  17   9   4  65  13
##         07    37  75  46   6   8  11   8  31 117
##         26    25  59  41   5   3  11  13  75 137
##         38    69 290  58   8   8   6   2  50  42
##         42    55 124  35   2   1  12   0  59  39
##         69   114 135   8   5   4   5   0  10  12
##         73    56  56  59   6   8  10   3  44  63
##         74    87 137  28   2   1  18   0  14   7
##         <NA>   0   1   0   0   0   0   0   0   0
```

Réaliser un croisement

Avec dplyr:

```
library(tidyr)
ra2010 %>%  group_by(dep,categorie_com) %>% summarise(nbcom=n()) %>%
  spread(categorie_com,nbcom)
```

```
## # A tibble: 9 x 10
## # Groups:   dep [9]
##     dep `111` `112` `120` `211` `212` `221` `222` `300` `400`
##   <chr> <int> <int> <int> <int> <int> <int> <int> <int> <int>
## 1    01    39   223    46     2    17     9     4    65    13
## 2    07    37    75    46     6     8    11     8    31   117
## 3    26    25    59    41     5     3    11    13    75   137
## 4    38    69   290    58     8     8     6     2    50    42
## 5    42    55   124    35     2     1    12    NA    59    39
## 6    69   114   135     8     5     4     5    NA    10    12
## 7    73    56    56    59     6     8    10     3    44    63
## 8    74    87   137    28     2     1    18    NA    14     7
## 9  <NA>    NA     1    NA    NA    NA    NA    NA    NA    NA
```

Réaliser un croisement

La fonction *prop.table()* donne par défaut les fréquences relatives à l'ensemble des effectifs.

```
freq <- round(prop.table(effcrois,2) *100,2)
freq
```

```
##            Catégorie de commune
## Département   111   112   120   211   212   221   222   300   400
##        01    8.09 20.27 14.33  5.56 34.00 10.98 13.33 18.68  3.02
##        07    7.68  6.82 14.33 16.67 16.00 13.41 26.67  8.91 27.21
##        26    5.19  5.36 12.77 13.89  6.00 13.41 43.33 21.55 31.86
##        38   14.32 26.36 18.07 22.22 16.00  7.32  6.67 14.37  9.77
##        42   11.41 11.27 10.90  5.56  2.00 14.63  0.00 16.95  9.07
##        69   23.65 12.27  2.49 13.89  8.00  6.10  0.00  2.87  2.79
##        73   11.62  5.09 18.38 16.67 16.00 12.20 10.00 12.64 14.65
##        74   18.05 12.45  8.72  5.56  2.00 21.95  0.00  4.02  1.63
##        <NA>  0.00  0.09  0.00  0.00  0.00  0.00  0.00  0.00  0.00
```

Pour obtenir des fréquences en lignes ou colonnes:

- valeur à 1 pour des fréquences en lignes
- valeur à 2 pour des fréquences en colonnes

Réaliser un croisement

La fonction *addmargins()* permet d'ajouter des marges au croisement.

```
freq <- round(addmargins(prop.table(effcrois,2) *100,1),2)
freq
```

```
##             Catégorie de commune
## Département    111    112    120    211    212    221    222    300    400
##         01    8.09  20.27  14.33   5.56  34.00  10.98  13.33  18.68   3.02
##         07    7.68   6.82  14.33  16.67  16.00  13.41  26.67   8.91  27.21
##         26    5.19   5.36  12.77  13.89   6.00  13.41  43.33  21.55  31.86
##         38   14.32  26.36  18.07  22.22  16.00   7.32   6.67  14.37   9.77
##         42   11.41  11.27  10.90   5.56   2.00  14.63   0.00  16.95   9.07
##         69   23.65  12.27   2.49  13.89   8.00   6.10   0.00   2.87   2.79
##         73   11.62   5.09  18.38  16.67  16.00  12.20  10.00  12.64  14.65
##         74   18.05  12.45   8.72   5.56   2.00  21.95   0.00   4.02   1.63
##         <NA>  0.00   0.09   0.00   0.00   0.00   0.00   0.00   0.00   0.00
##         Sum 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00 100.00
```

Pour obtenir des marges soit en lignes soit en colonnes:

- valeur à 2 pour des marges en lignes
- valeur à 1 pour des marges en colonnes

Les statistiques pondérées *Hmisc*

Voici un jeu d'essai:

```
poids <- c(1,2,1.5,3,2.5,1,1)
age <- c(20,25,30,35,40,45,50)
DF <- data.frame(poids,age)
```

```
DF
```

```
##   poids age
## 1   1.0  20
## 2   2.0  25
## 3   1.5  30
## 4   3.0  35
## 5   2.5  40
## 6   1.0  45
## 7   1.0  50
```

```
library("Hmisc")
DF %>% summarise(quartile1=wtd.quantile(age,w=poids, probs=0.25),
                 mediane=wtd.quantile(age,w=poids, probs=0.5),
                 moyenne=wtd.mean(age,w=poids))
```

```
##   quartile1 mediane  moyenne
## 1     28.75      35 34.58333
```

10 - Quelques éléments graphiques

script graphique standard

R permet de réaliser un grand nombre de graphiques pouvant être extrêmement élaborés.
Cette séquence se contente de traiter quelques graphiques communément utilisés.

Script standard :

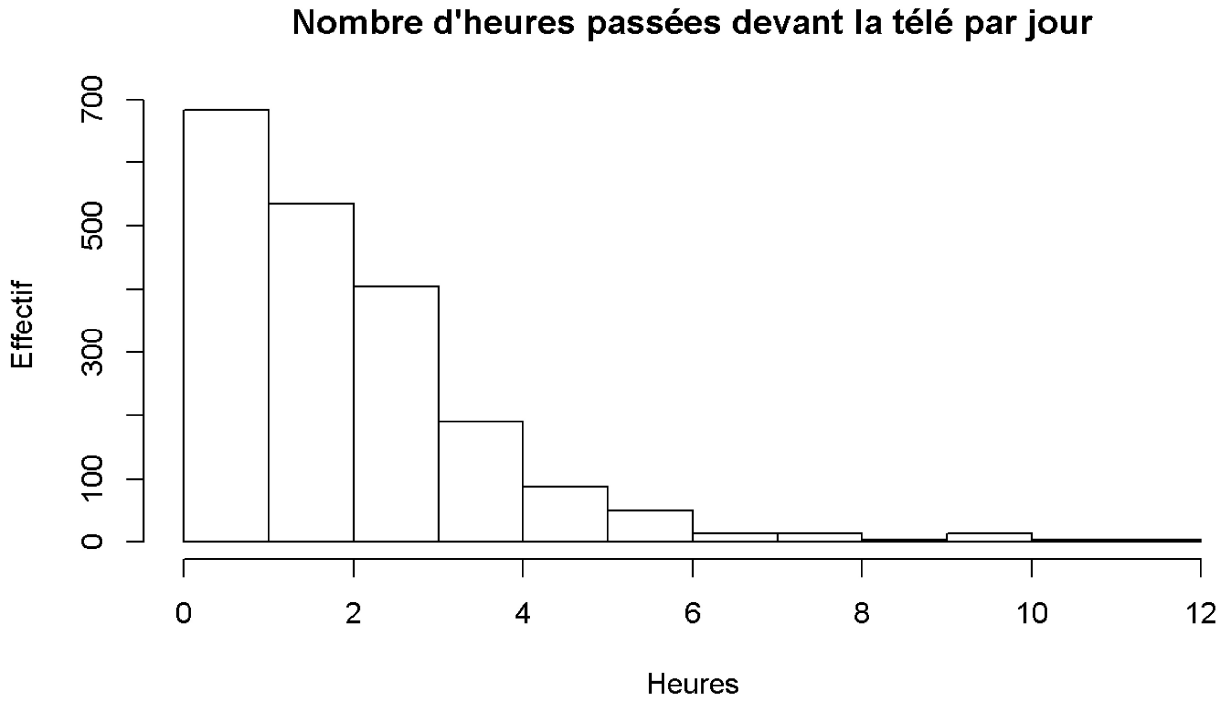
```
nomGraph(données,main="titre",...arguments du graphique sur les couleurs et les placements des indicateurs)
```

La fonction `par()` permet de gérer les très nombreux paramètres graphiques.

```
library("questionr")
data("hdv2003")
donnee <- hdv2003 # on charge le jeu de donnée fournit par questionr
```

Histogramme *hist*

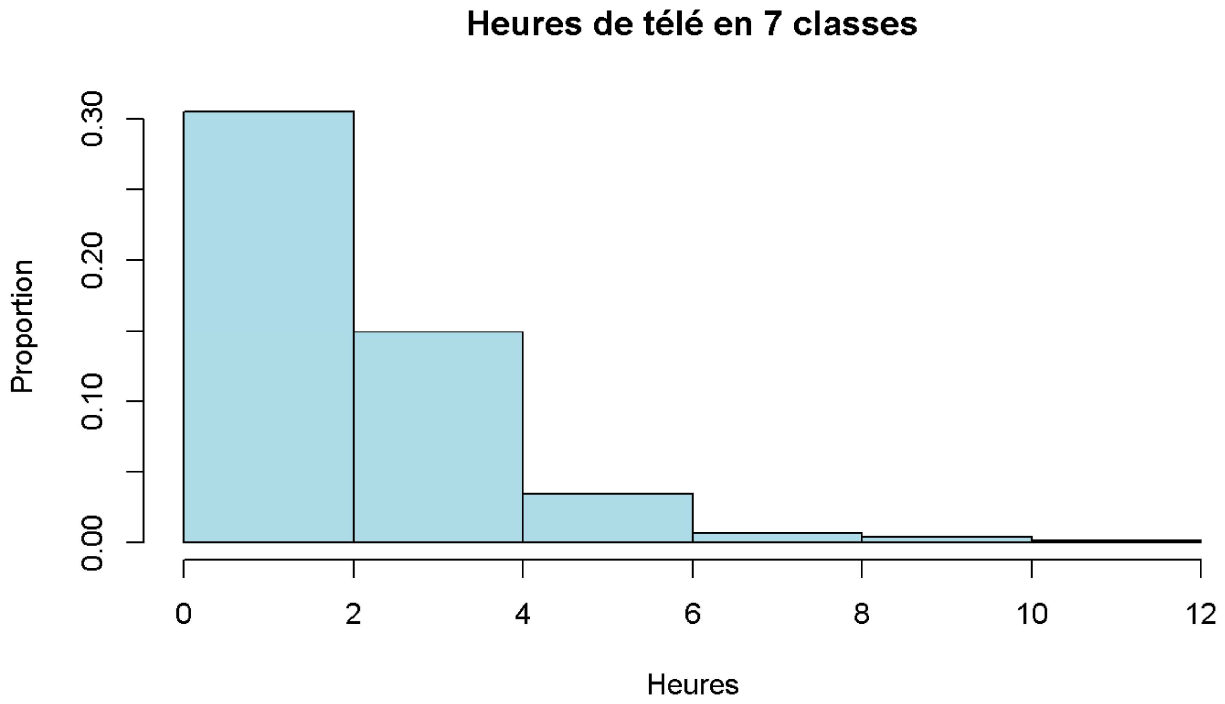
```
hist(donnee$heures.tv, main = "Nombre d'heures passées devant la télé par jour",
     xlab = "Heures", ylab = "Effectif")
```



Histogramme *hist*

Nombre de classes choisi

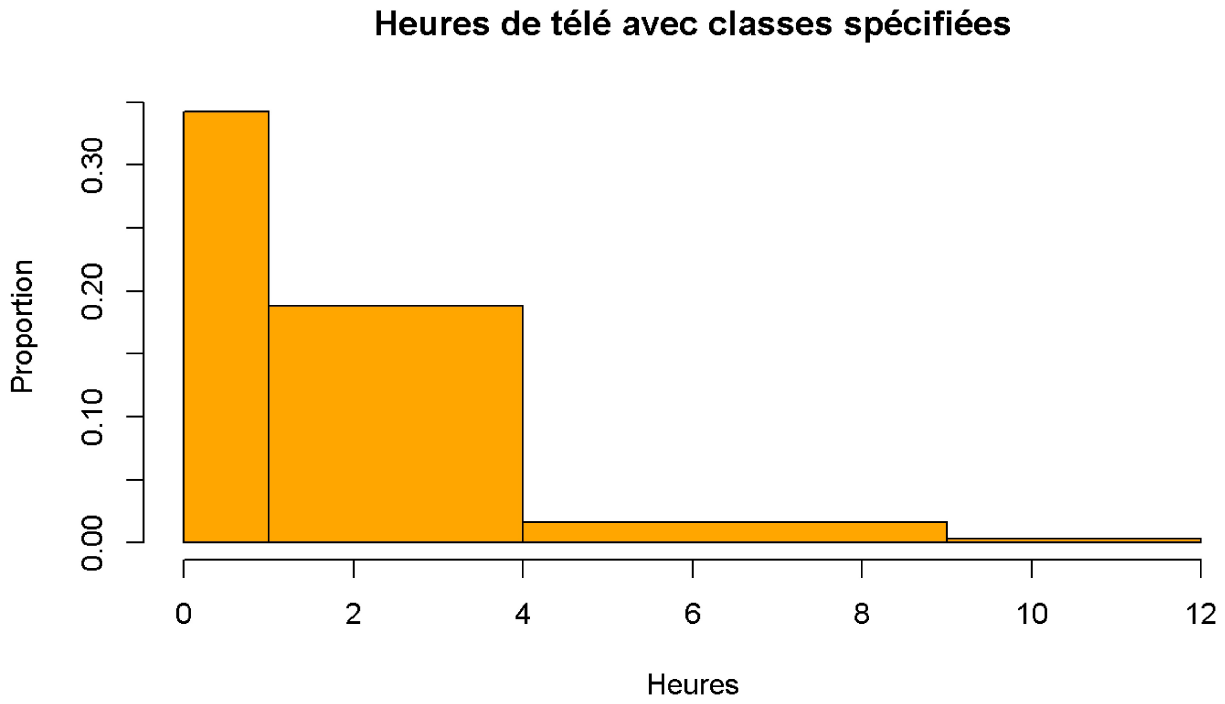
```
hist(donnee$heures.tv, main = "Heures de télé en 7 classes", breaks = 7, xlab = "Heures",
  ylab = "Proportion", probability = TRUE, col = "lightblue")
```



Histogramme *hist*

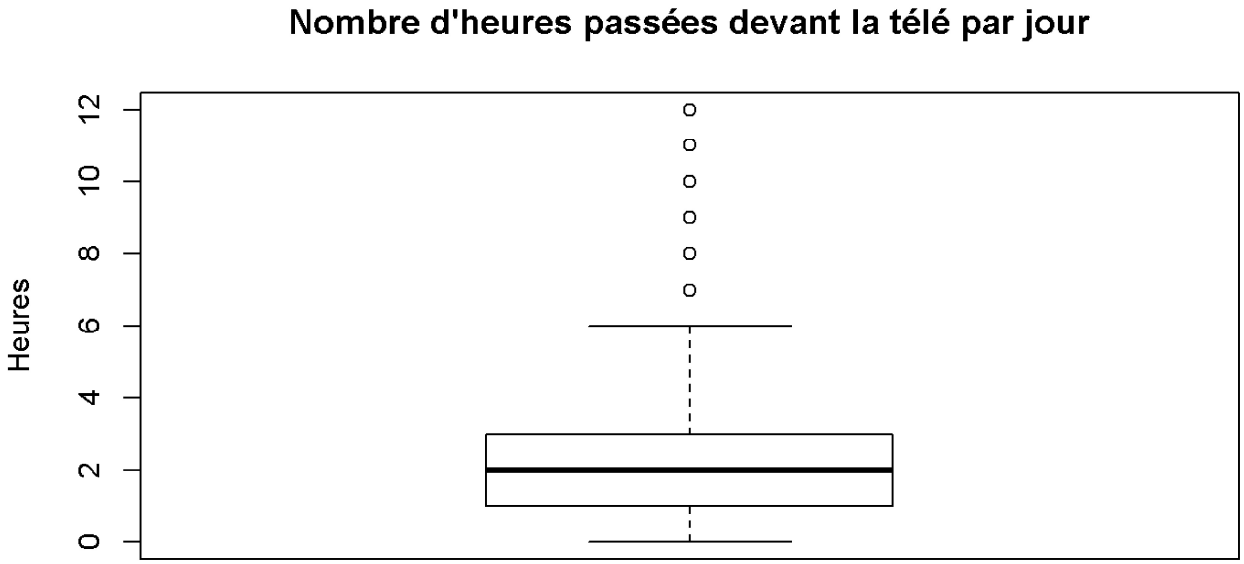
Classes choisies

```
hist(donnee$heures.tv, main = "Heures de télé avec classes spécifiées", breaks = c(0,
  1, 4, 9, 12), xlab = "Heures", ylab = "Proportion", col = "orange")
```



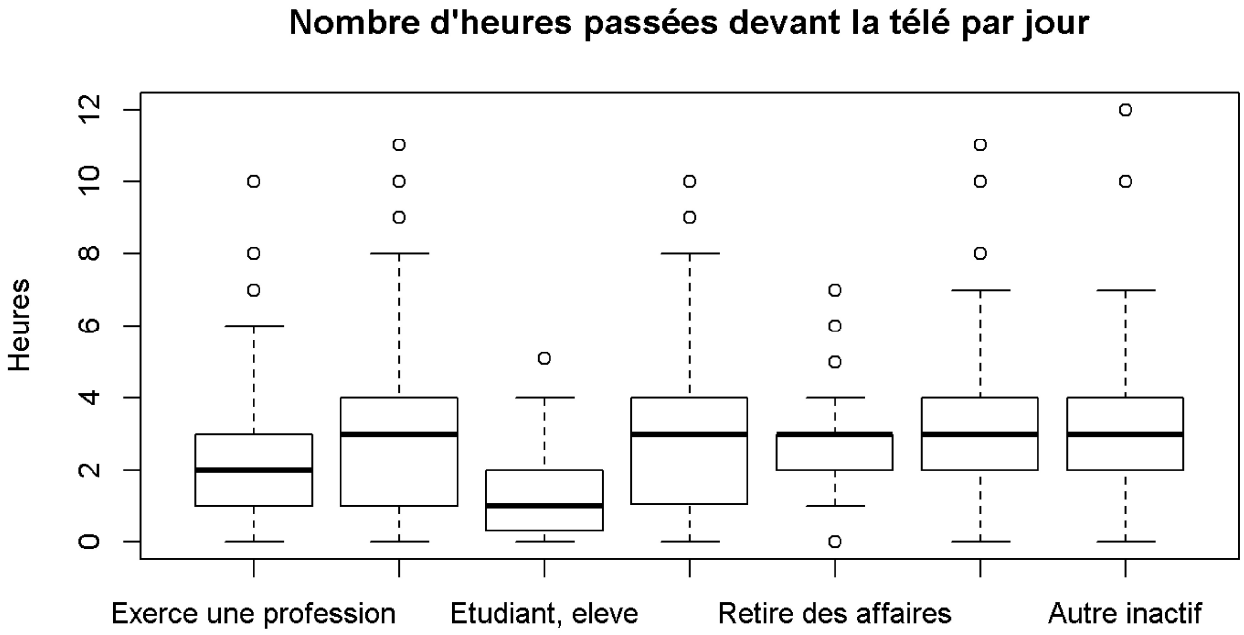
Boîte à moustaches *boxplot*

```
boxplot(donnee$heures.tv, main = "Nombre d'heures passées devant la télé par jour",
        ylab = "Heures")
```



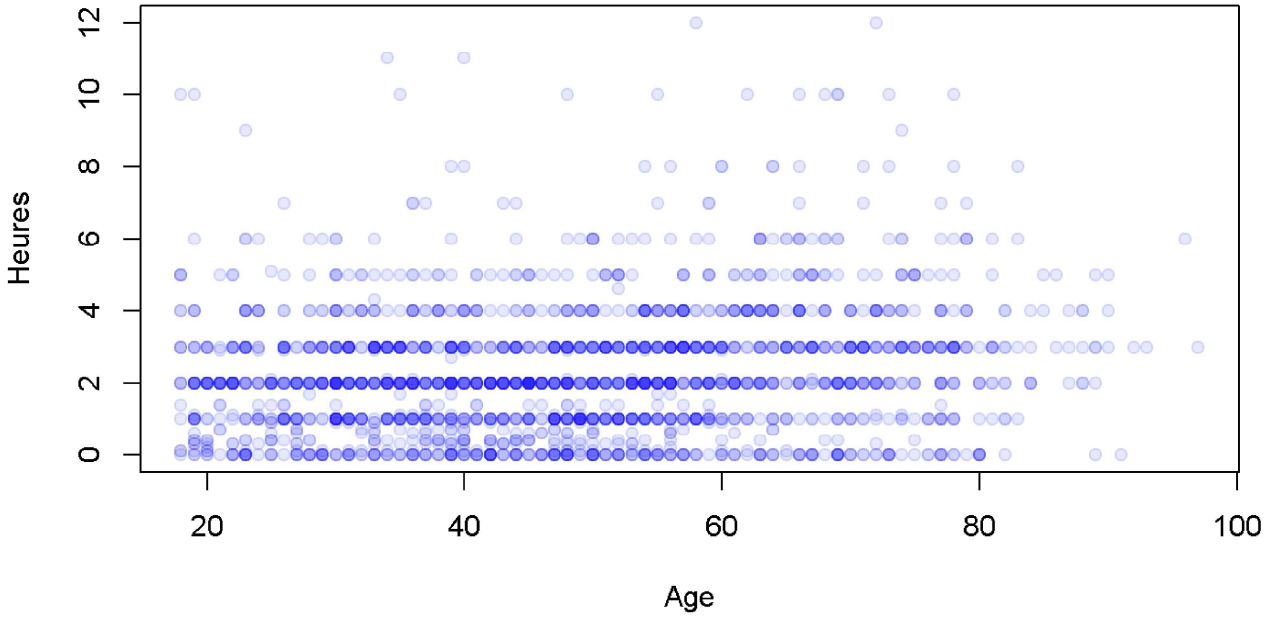
Boîte à moustaches *boxplot*

```
boxplot(donnee$heures.tv~donnee$occup, main = "Nombre d'heures passées devant la télé par jour",
        ylab = "Heures")
```



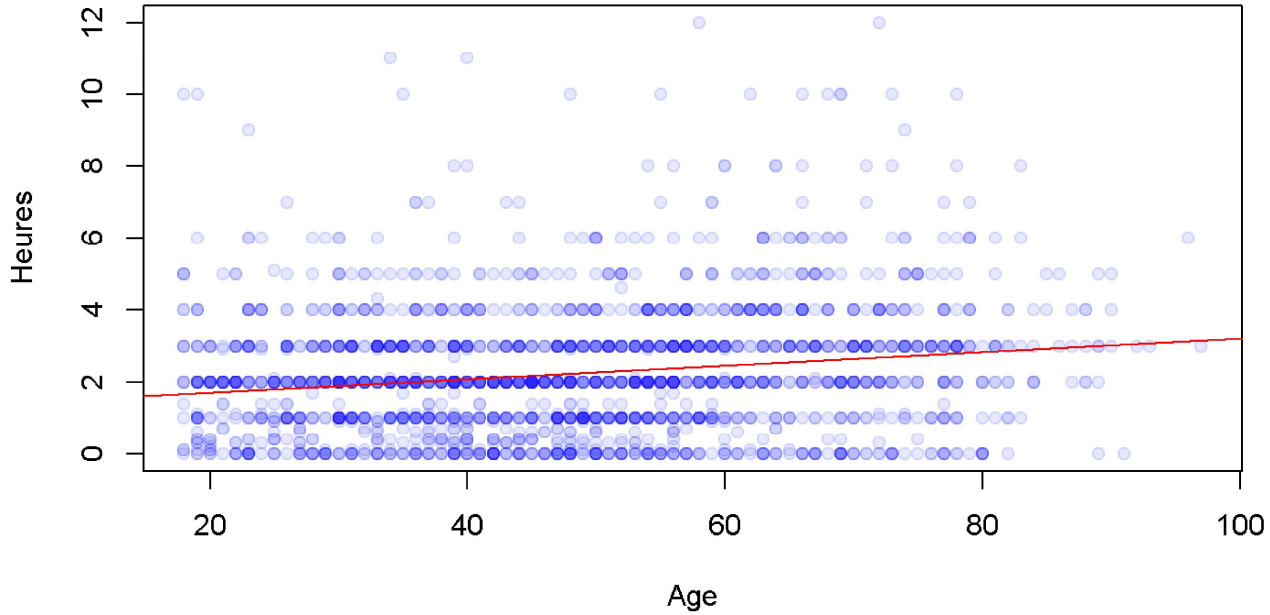
Nuage de points *plot*

```
plot(donnee$age, donnee$heures.tv, pch = 19, col = rgb(0, 0, 1, 0.1),
     xlab = "Age", ylab = "Heures")
```



Nuage de points avec une droite de régression

```
plot(donnee$age, donnee$heures.tv, pch = 19, col = rgb(0, 0, 1, 0.1),
     xlab = "Age", ylab = "Heures")
abline(lm(donnee$heures.tv~donnee$age),col="red")
```



Sauvegarde d'un graphique

Via l'interface de RStudio

L'export de graphiques est très facile avec RStudio. Lorsque l'on créé un graphique, ce dernier est affiché sous l'onglet Plots dans le quadrant inférieur droit. Il suffit de cliquer sur Export pour avoir accès à trois options différentes :

- Save as image pour sauvegarder le graphique en tant que fichier image ;
- Save as PDF pour sauvegarder le graphique dans un fichier PDF ;
- Copy to Clipboard pour copier le graphique dans le presse-papier (et pouvoir ainsi le coller ensuite dans un document Word par exemple).

via un script

```
boxplot(rnorm(100))
dev.print(device = png, file = "export.png", width = 600)
```