INTRODUCTION AUX LOGICIELS D'ANALYSES STATISTIQUES

INTRODUCTION AU LOGICIEL R

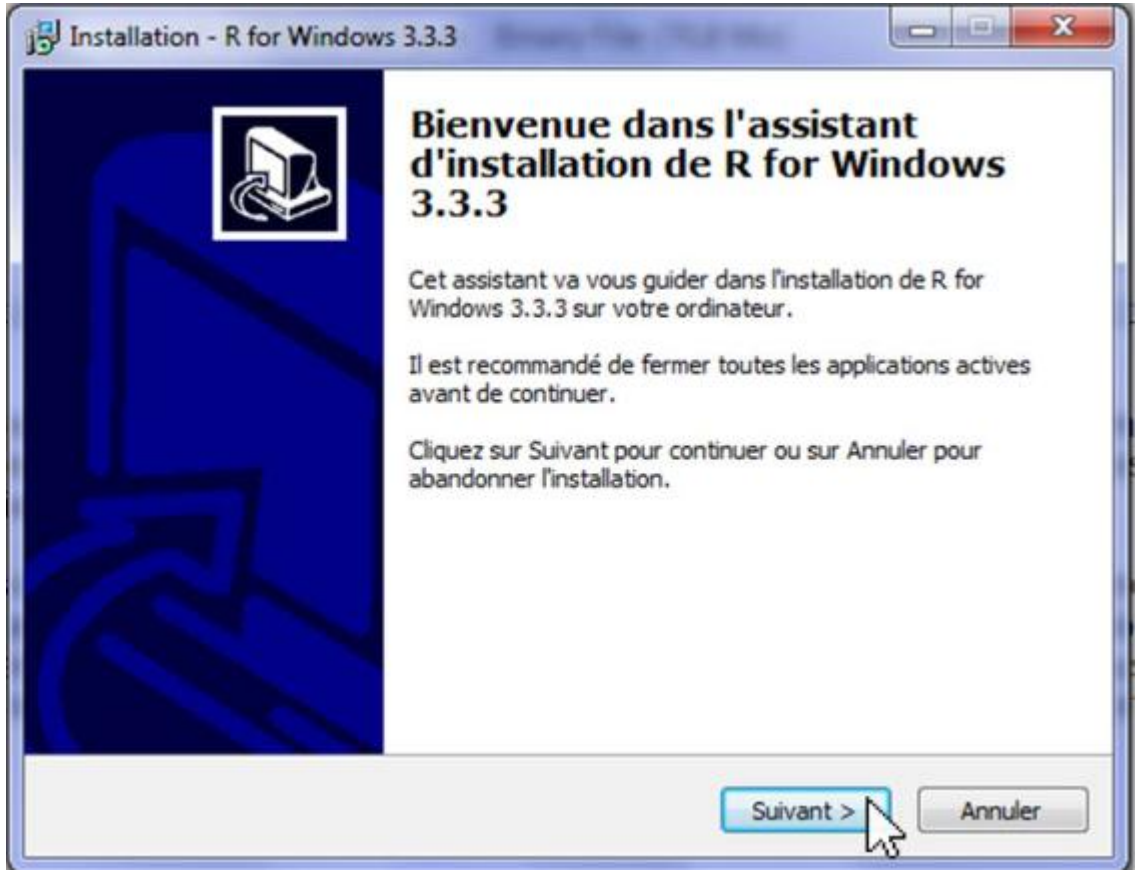
1. Prise en main (préliminaires)

Nous aurons besoin de deux outils pour notre initiation au logiciel R: R et RSTUDIO.

- Pour télécharger R rendez vous sur: https://cran.r-project.org/bin/windows/base/
- Pour télécharger RSTUDIO rendez vous sur: https://rstudio.com/products/rstudio/download/#download

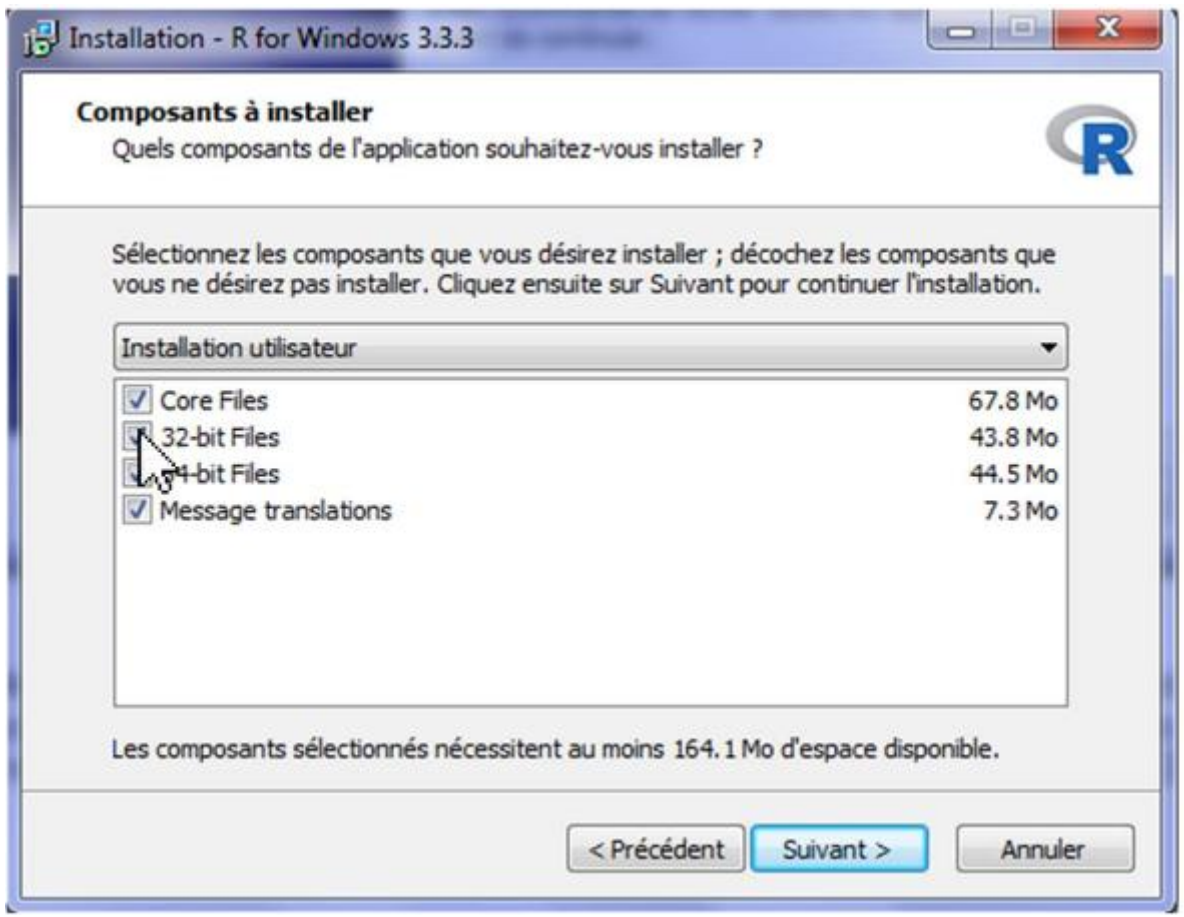
1. Prise en main (préliminaires)

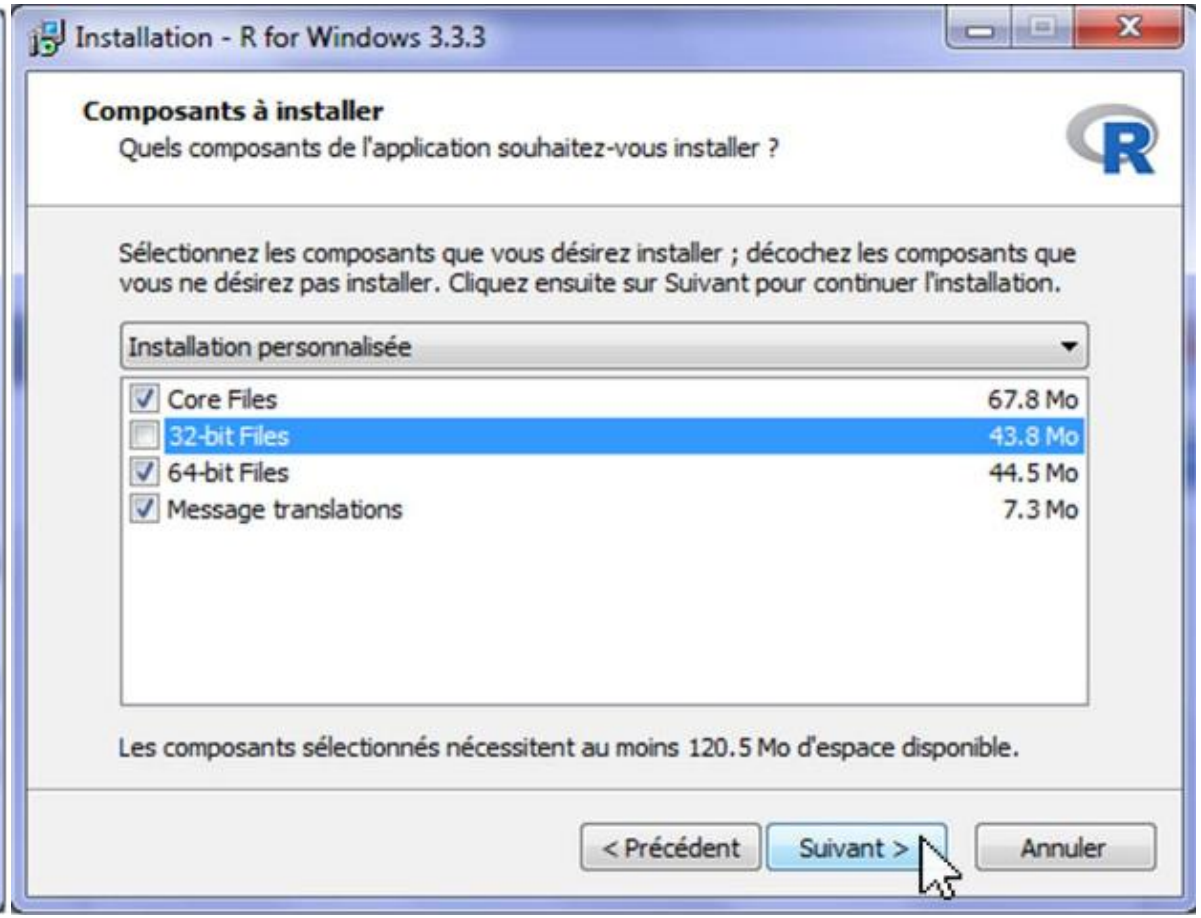
- Installation de R (étape 1)



1. Prise en main (préliminaires)

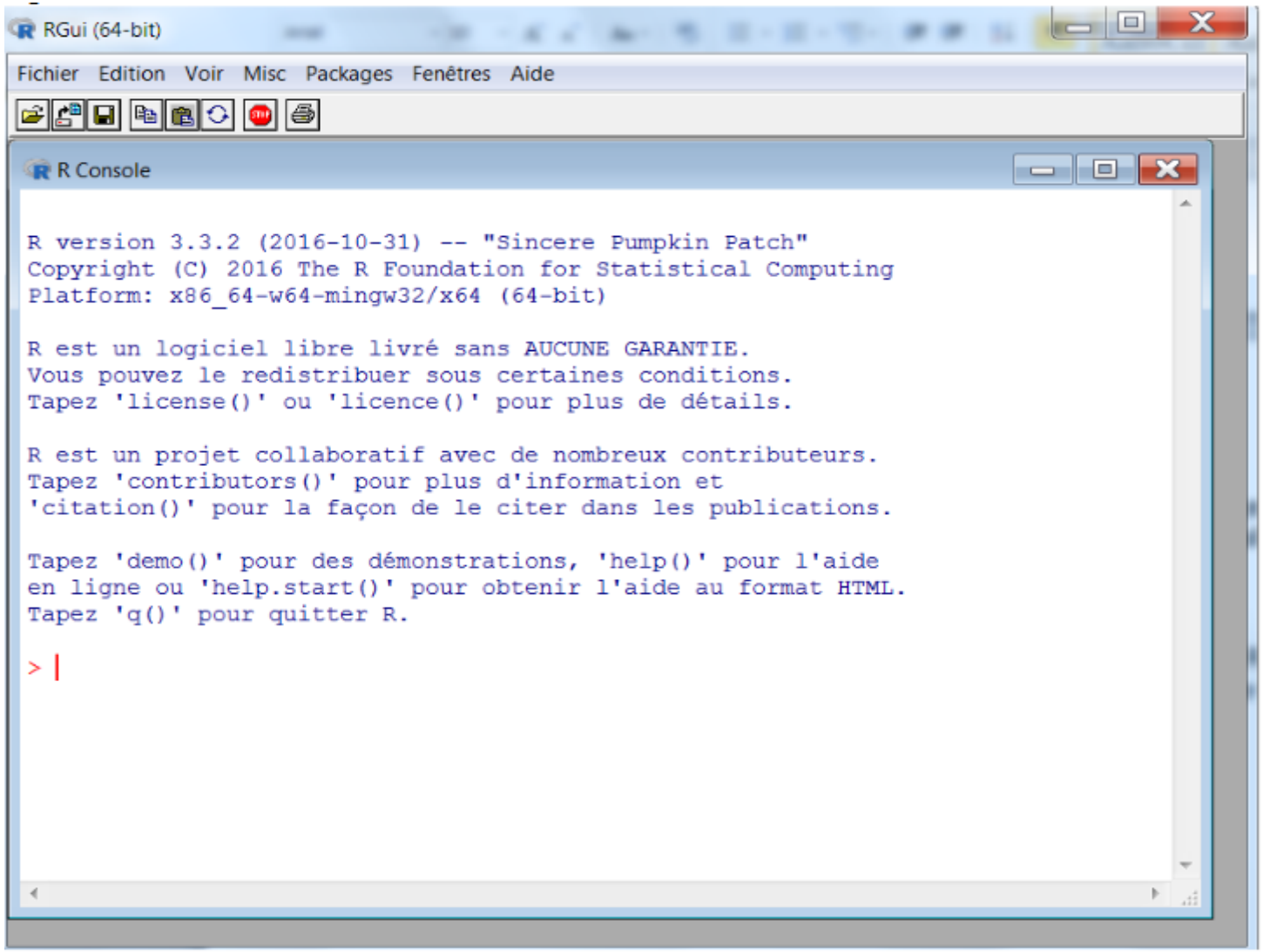
- Installation de R (étape 2)





1. Prise en main (préliminaires)

- Installation de R (étape 3)



1. Prise en main (préliminaires)

- Installation de Rstudio

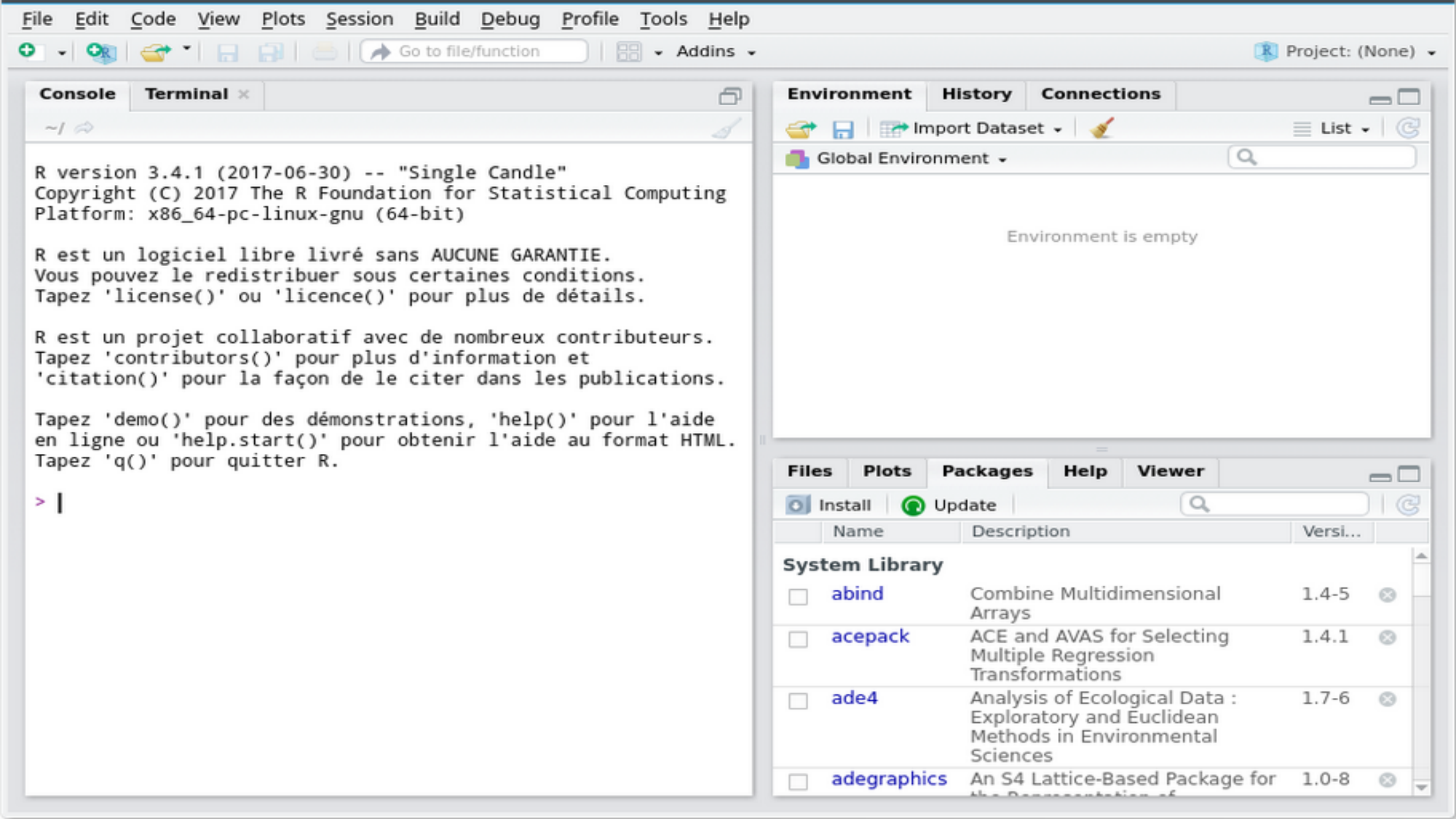
1. Prise en main

Une fois R et RStudio installés sur votre machine, nous n'allons pas lancer R mais plutôt RStudio.

RStudio n'est pas à proprement parler une interface graphique qui permettrait d'utiliser R de manière "classique" via la souris, des menus et des boîtes de dialogue. Il s'agit plutôt de ce qu'on appelle un *Environnement de développement intégré* (IDE) qui facilite l'utilisation de R et le développement de scripts.

1.1 La console
1.1.1 L’invite de commandes

Au premier lancement de RStudio, l’écran principal est découpé en trois grandes zones :


File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Console
Terminal
R version 3.4.1 (2017-06-30) -- "Single Candle"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)
R est un logiciel libre livré sans AUCUNE GARANTIE.
Vous pouvez le redistribuer sous certaines conditions.
Tapez 'license()' ou 'licence()' pour plus de détails.
R est un projet collaboratif avec de nombreux contributeurs.
Tapez 'contributors()' pour plus d'information et
'citation()' pour la façon de le citer dans les publications.
Tapez 'demo()' pour des démonstrations, 'help()' pour l'aide
en ligne ou 'help.start()' pour obtenir l'aide au format HTML.
Tapez 'q()' pour quitter R.
>
Environment
History
Connections
Import Dataset
List
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
Install
Update
Name
Description
Versi...
System Library
abind
Combine Multidimensional Arrays
1.4-5
acepack
ACE and AVAS for Selecting Multiple Regression Transformations
1.4.1
ade4
Analysis of Ecological Data : Exploratory and Euclidean Methods in Environmental Sciences
1.7-6
adegraphics
An S4 Lattice-Based Package for
1.0-8

La zone de gauche se nomme *Console*. À son démarrage, RStudio a lancé une nouvelle session de R et c'est dans cette fenêtre que nous allons pouvoir interagir avec lui.
La *Console* doit normalement afficher un texte de bienvenue ressemblant à ceci :

```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

>
```

Suivi d'une ligne commençant par le caractère > et sur laquelle devrait se trouver votre curseur. Cette ligne est appelée l'*invite de commande* (ou *prompt* en anglais).
Elle signifie que R est disponible et en attente de votre prochaine commande.

Nous pouvons tout de suite lui fournir une première commande, en saisissant le texte suivant puis en appuyant sur Entrée :

```
2 + 2
```

```
[1] 4
```

R nous répond immédiatement, et nous pouvons constater qu'il sait faire des additions à un chiffre. On peut donc continuer avec d'autres opérations :

```
5 - 7
```

```
[1] -2
```

```
4 * 12
```

```
[1] 48
```

```
-10 / 3
```

```
[1] -3.333333
```

1.1.2 Précisions concernant la saisie des commandes

Lorsqu'on saisit une commande, les espaces autour des opérateurs n'ont pas d'importance. Les trois commandes suivantes sont donc équivalentes, mais on privilégie en général la deuxième pour des raisons de lisibilité du code.

```
10+2
10 + 2
10       +       2
```

Quand vous êtes dans la console, vous pouvez utiliser les flèches vers le haut et vers le bas pour naviguer dans l'historique des commandes que vous avez tapées précédemment. Vous pouvez à tout moment modifier la commande affichée, et l'exécuter en appuyant sur Entrée. Enfin, il peut arriver qu'on saisisse une commande de manière incomplète : oubli d'une parenthèse, faute de frappe, etc.
Dans ce cas, R remplace l'invite de commande habituel par un signe + :

```
4 *
+
```

Cela signifie qu'il "attend la suite". On peut alors soit compléter la commande sur cette nouvelle ligne et appuyer sur Entrée, soit, si on est perdu, tout annuler et revenir à l'invite de commandes normal en appuyant sur Esc ou Échap.

1.2 Objets
1.2.1 Objets simples

Faire des calculs c'est bien, mais il serait intéressant de pouvoir stocker un résultat quelque part pour pouvoir le réutiliser ultérieurement sans avoir à faire du copier/coller. Pour conserver le résultat d'une opération, on peut le stocker dans un *objet* à l'aide de l'opérateur d'assignation <-. Cette "flèche" stocke ce qu'il y a à sa droite dans un objet dont le nom est indiqué à sa gauche. Prenons tout de suite un exemple :

```
x <- 2
```

Cette commande peut se lire "*prend la valeur 2 et mets la dans un objet qui s'appelle x*". Si on exécute une commande comportant juste le nom d'un objet, R affiche son contenu :

```
x
```

```
[1] 2
```

On voit donc que notre objet x contient bien la valeur 2.
On peut évidemment réutiliser cet objet dans d'autres opérations. R le remplacera alors par sa valeur :

```
x + 4
```

```
[1] 6
```

```
x <- 2
y <- 5
resultat <- x + y
resultat
```

```
[1] 7
```

Quand on assigne une nouvelle valeur à un objet déjà existant, la valeur précédente est perdue. Les objets n'ont pas de mémoire.

```
x <- 2
x <- 5
x
```

```
[1] 5
```

On le verra, les objets peuvent contenir tout un tas d'informations. Jusqu'ici on n'a stocké que des nombres, mais ils peuvent aussi contenir des chaînes de caractères (du texte), qu'on délimite avec des guillemets simples ou doubles (' ou ") :

```
chien <- "Chihuahua"
chien
```

```
[1] "Chihuahua"
```

1.2.2 Vecteurs

Imaginons maintenant qu'on a demandé la taille en centimètres de 5 personnes et qu'on souhaite calculer leur taille moyenne. On pourrait créer autant d'objets que de tailles et faire l'opération mathématique qui va bien :

```
taille1 <- 156
taille2 <- 164
taille3 <- 197
taille4 <- 147
taille5 <- 173
(taille1 + taille2 + taille3 + taille4 + taille5) / 5
```

```
[1] 167.4
```

Cette manière de faire n'est évidemment pas pratique du tout. On va plutôt stocker l'ensemble de nos tailles dans un seul objet, de type *vecteur*, avec la syntaxe suivante :

```
tailles <- c(156, 164, 197, 147, 173)
```

Si on affiche le contenu de cet objet, on voit qu'il contient bien l'ensemble des tailles saisies :

```
tailles
```

```
[1] 156 164 197 147 173
```

Un *vecteur* dans R est un objet qui peut contenir plusieurs informations du même type, potentiellement en très grand nombre.
L'avantage d'un vecteur est que lorsqu'on lui applique une opération, celle-ci s'applique à toutes les valeurs qu'il contient. Ainsi, si on veut la taille en mètres plutôt qu'en centimètres, on peut faire :

```
tailles_m <- tailles / 100
tailles_m
```

```
[1] 1.56 1.64 1.97 1.47 1.73
```

Cela fonctionne pour toutes les opérations de base :

Imaginons maintenant qu'on a aussi demandé aux cinq mêmes personnes leur poids en kilos. On peut alors créer un deuxième vecteur :

```
poids <- c(45, 59, 110, 44, 88)
```

On peut alors effectuer des calculs utilisant nos deux vecteurs tailles et poids. On peut par exemple calculer l'indice de masse corporelle (IMC) de chacun de nos enquêtés en divisant leur poids en kilo par leur taille en mètre au carré :

```
imc <- poids / (tailles / 100) ^ 2
imc
```

```
[1] 18.49112 21.93635 28.34394 20.36189 29.40292
```

Un vecteur peut contenir des nombres, mais il peut aussi contenir du texte. Imaginons qu'on a demandé aux 5 mêmes personnes leur niveau de diplôme : on peut regrouper l'information dans un vecteur de *chaînes de caractères*. Une chaîne de caractère contient du texte libre, délimité par des guillemets simples ou doubles :

```
diplome <- c("CAP", "Bac", "Bac+2", "CAP", "Bac+3")
diplome
```

```
[1] "CAP"   "Bac"   "Bac+2" "CAP"   "Bac+3"
```

L'opérateur :, lui, permet de générer rapidement un vecteur comprenant tous les nombres entre deux valeurs, opération assez courante sous R :

```
x <- 1:10
x
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

Enfin, notons qu'on peut accéder à un élément particulier d'un vecteur en faisant suivre le nom du vecteur de crochets contenant le numéro de l'élément désiré. Par exemple :

```
diplome[2]
```

```
[1] "Bac"
```

Cette opération, qui utilise l'opérateur [], permet donc la sélection d'éléments d'un vecteur.
Dernière remarque, si on affiche dans la console un vecteur avec beaucoup d'éléments, ceux-ci seront répartis sur plusieurs lignes. Par exemple, si on a un vecteur de 50 nombres on peut obtenir quelque chose comme :

```
 [1] 294 425 339 914 114 896 716 648 915 587 181 926 489
[14] 848 583 182 662 888 417 133 146 322 400 698 506 944
[27] 237 324 333 443 487 658 793 288 897 588 697 439 697
[40] 914 694 126 969 744 927 337 439 226 704 635
```

On remarque que R ajoute systématiquement un nombre entre crochets au début de chaque ligne : il s'agit en fait de la position du premier élément de la ligne dans le vecteur Ainsi, le 848 de la deuxième ligne est le 14e élément du vecteur, le 914 de la dernière ligne est le 40e, etc.

1.2.3 Matrices

La syntaxe basique pour créer une matrice dans R est:

```
matrix(data, nrow, ncol, byrow, dimnames)
```

Exemple:

```r
# les éléments sont rangés sequentiellement par ligne.
M <- matrix(c(3:14), nrow = 4, byrow = TRUE)
print(M)

#les éléments sont rangés sequentiellement par colonnes.
N <- matrix(c(3:14), nrow = 4, byrow = FALSE)
print(N)

# Definir le nom des colonnes etd es lignes.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")

P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
print(P)
```

1.2.3 Matrices

```
     [,1] [,2] [,3]
[1,]    3    4    5
[2,]    6    7    8
[3,]    9   10   11
[4,]   12   13   14
     [,1] [,2] [,3]
[1,]    3    7   11
[2,]    4    8   12
[3,]    5    9   13
[4,]    6   10   14
     col1 col2 col3
row1    3    4    5
row2    6    7    8
row3    9   10   11
row4   12   13   14
```

1.2.3 Matrices

ACCEDER AUX ELEMENTS D'UNE MATRICE

```r
# Noms de colonnes et de lignes.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")

# Creation de la matrice.
P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))

# Acceder à l'élément à la 1ere ligne et 3eme colonne.
print(P[1,3])

# Acceder à l'élément à la 4eme ligne et 2eme colonne.
print(P[4,2])

# 2eme ligne uniquement.
print(P[2,])

# 3eme colonne uniquement.
print(P[,3])
```

1.2.3 Matrices

CALCUL MATRICIEL

- Addition et soustraction

```r
# Creer deux matrices 2X3.
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
print(matrix1)

matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
print(matrix2)

# Addition.
result <- matrix1 + matrix2
cat("Result of addition","\n")
print(result)

# Soustraction
result <- matrix1 - matrix2
cat("Result of subtraction","\n")
print(result)
```

1.2.3 Matrices

CALCUL MATRICIEL

- Multiplication et division

```
# Creer deux matrices 2X3.
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
print(matrix1)

matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
print(matrix2)

# multiplication.
result <- matrix1 * matrix2
cat("Result of multiplication","\n")
print(result)

# Division
result <- matrix1 / matrix2
cat("Result of division","\n")
print(result)
```

1.3 Fonctions
1.3.1 Principe

Nous savons désormais effectuer des opérations arithmétiques de base sur des nombres et des vecteurs, et stocker des valeurs dans des objets pour pouvoir les réutiliser plus tarc

Pour aller plus loin, nous devons aborder les *fonctions* qui sont, avec les objets, un deuxième concept de base de R. On utilise des fonctions pour effectuer des calculs, obtenir des résultats et accomplir des actions.
Formellement, une fonction a un *nom*, elle prend en entrée entre parenthèses un ou plusieurs *arguments* (ou *paramètres*), et retourne un *résultat*.
Prenons un exemple. Si on veut connaître le nombre d'éléments du vecteur tailles que nous avons construit précédemment, on peut utiliser la fonction length, de cette manière :

```
length(tailles)
```

```
[1] 5
```

Ici, length est le nom de la fonction, on l'appelle en lui passant un argument entre parenthèses (en l'occurrence notre vecteur tailles), et elle nous renvoie un résultat, à savoir le nombre d'éléments du vecteur passé en paramètre.
Autre exemple, les fonctions min et max retournent respectivement les valeurs minimales et maximales d'un vecteur de nombres :

```
min(tailles)
```

```
[1] 147
```

```
max(tailles)
```

```
[1] 197
```

La fonction mean calcule et retourne la moyenne d'un vecteur de nombres :

```
mean(tailles)
```

```
[1] 167.4
```

La fonction sum retourne la somme de tous les éléments du vecteur :

```
sum(tailles)
```

```
[1] 837
```

Jusqu'à présent on n'a vu que des fonctions qui calculent et retournent un unique nombre
Mais une fonction peut renvoyer d'autres types de résultats. Par exemple, la fonction range (étendue) renvoie un vecteur de deux nombres, le minimum et le maximum :

```
range(tailles)
```

```
[1] 147 197
```

Ou encore, la fonction unique, qui supprime toutes les valeurs en double dans un vecteur, qu'il s'agisse de nombres ou de chaînes de caractères :

```
diplome <- c("CAP", "Bac", "Bac+2", "CAP", "Bac+3")
unique(diplome)
```

```
[1] "CAP"   "Bac"   "Bac+2" "Bac+3"
```

1.3.2 Arguments

Une fonction peut prendre plusieurs arguments, dans ce cas on les indique toujours entre parenthèses, séparés par des virgules.
On a déjà rencontré un exemple de fonction acceptant plusieurs arguments : la fonction c, qui combine l'ensemble de ses arguments en un vecteur :

```
tailles <- c(156, 164, 197, 181, 173)
```

Ici, c est appelée en lui passant cinq arguments, les cinq tailles séparées par des virgules, et elle renvoie un vecteur numérique regroupant ces cinq valeurs.
Supposons maintenant que dans notre vecteur tailles nous avons une valeur manquante (une personne a refusé de répondre, ou notre mètre mesureur était en panne).
On symbolise celle-ci dans R avec le code interne NA :

```r
tailles <- c(156, 164, 197, NA, 173)
tailles
```

```
[1] 156 164 197  NA 173
```

Si je calcule maintenant la taille moyenne à l'aide de la fonction mean, j'obtiens :

```r
mean(tailles)
```

```
[1] NA
```

En effet, R considère par défaut qu'il ne peut pas calculer la moyenne si une des valeurs n'est pas disponible. Il considère alors que cette moyenne est elle-même "non disponible" et renvoie donc comme résultat NA.
On peut cependant indiquer à mean d'effectuer le calcul en ignorant les valeurs manquantes. Ceci se fait en ajoutant un argument supplémentaire, nommé na.rm (abbréviation de *NA remove*, "enlever les NA"), et de lui attribuer la valeur TRUE (code interne de R signifiant *vrai*) :

```
mean(tailles, na.rm = TRUE)
```

```
[1] 172.5
```

1.3.3 Aide sur une fonction

Il est fréquent de ne pas savoir (ou d'avoir oublié) quels sont les arguments d'une fonctior , ou comment ils se nomment. On peut à tout moment faire appel à l'aide intégrée à R en passant le nom de la fonction (entre guillemets) à la fonction **help** :

```
help("mean")
```

On peut aussi utiliser le raccourci ?mean.
Ces deux commandes affichent une page (en anglais) décrivant la fonction, ses paramètres, son résultat, le tout accompagné de diverses notes, références et exemples. Ces pages d'aide contiennent à peu près tout ce que vous pourrez chercher à savoir, mais elles ne sont pas toujours d'une lecture aisée.
Dans RStudio, les pages d'aide en ligne s'ouvriront par défaut dans la zone en bas à droite, sous l'onglet *Help*. Un clic sur l'icône en forme de maison vous affichera la page d'accueil de l'aide.

1.4 Regrouper ses commandes dans des scripts

Jusqu'ici on a utilisé R de manière "interactive", en saisissant des commandes directement dans la console. Ça n'est cependant pas la manière dont on va utiliser R au quotidien, pour une raison simple : lorsque R redémarre, tout ce qui a été effectué dans la console est perdu.
Plutôt que de saisir nos commandes dans la console, on va donc les regrouper dans des scripts (de simples fichiers texte), qui vont garder une trace de toutes les opérations effectuées, et ce sont ces scripts, sauvegardés régulièrement, qui seront le "coeur" de notre travail. C'est en rouvrant les scripts et en réexécutant les commandes qu'ils contiennent qu'on pourra "reproduire" les données, leur traitement, les analyses et leurs résultats.
Pour créer un script, il suffit de sélectionner le menu *File*, puis *New file* et *R script*. Une quatrième zone apparaît alors en haut à gauche de l'interface de RStudio. On peut enregistrer notre script à tout moment dans un fichier avec l'extension .R, en cliquant sur l'icône de disquette ou en choissant *File* puis *Save*.
Un script est un fichier texte brut, qui s'édite de la manière habituelle. À la différence de la console, quand on appuie sur Entrée, cela n'exécute pas la commande en cours mais insère un saut de ligne (comme on pouvait s'y attendre).
Pour exécuter une commande saisie dans un script, il suffit de positionner le curseur sur la ligne de la commande en question, et de cliquer sur le bouton *Run* dans la barre d'outils juste au-dessus de la zone d'édition du script. On peut aussi utiliser le raccourci clavier Ctrl + Entrée (Cmd + Entrée sous Mac). On peut enfin sélectionner plusieurs lignes avec la souris ou le clavier et cliquer sur *Run* (ou utiliser le raccourci clavier), et l'ensemble des lignes est exécuté d'un coup.

```r
tailles <- c(156, 164, 197, 147, 173)
poids <- c(45, 59, 110, 44, 88)

mean(tailles)
mean(poids)

imc <- poids / (tailles / 100) ^ 2
min(imc)
max(imc)
```

1.4.1 Commentaires

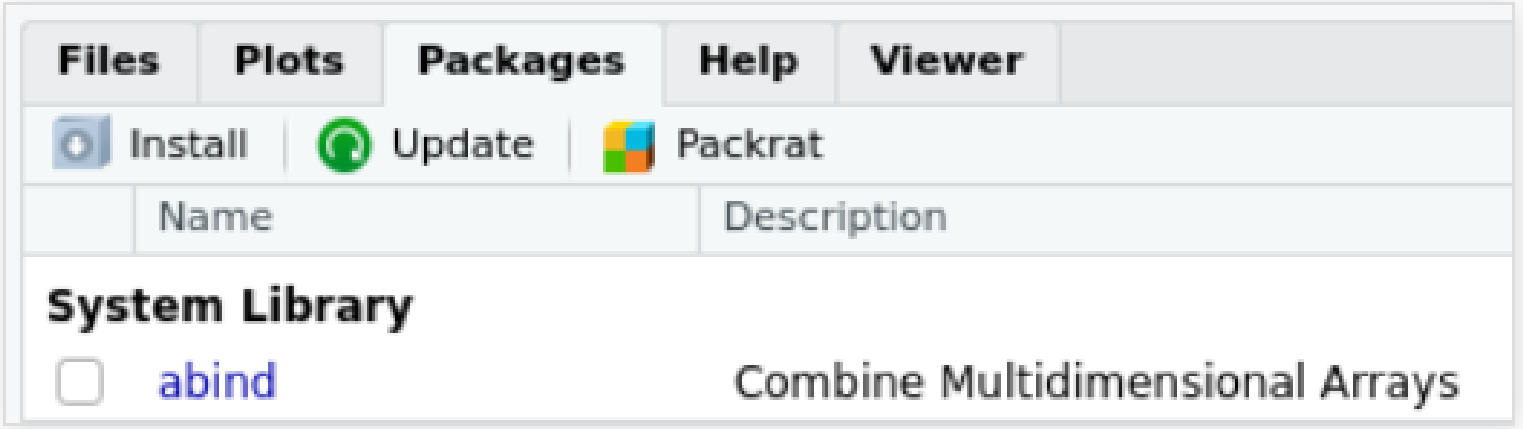
Les commentaires sont un élément très important d'un script. Il s'agit de texte libre, ignoré par R, et qui permet de décrire les étapes du script, sa logique, les raisons pour lesquelles on a procédé de telle ou telle manière… Il est primordial de documenter ses scripts à l'aide de commentaires, car il est très facile de ne plus se retrouver dans un programme qu'on a produit soi-même, même après une courte interruption.
Pour ajouter un commentaire, il suffit de le faire précéder d'un ou plusieurs symboles #.
En effet, dès que R rencontre ce caractère, il ignore tout ce qui se trouve derrière, jusqu'à la fin de la ligne.

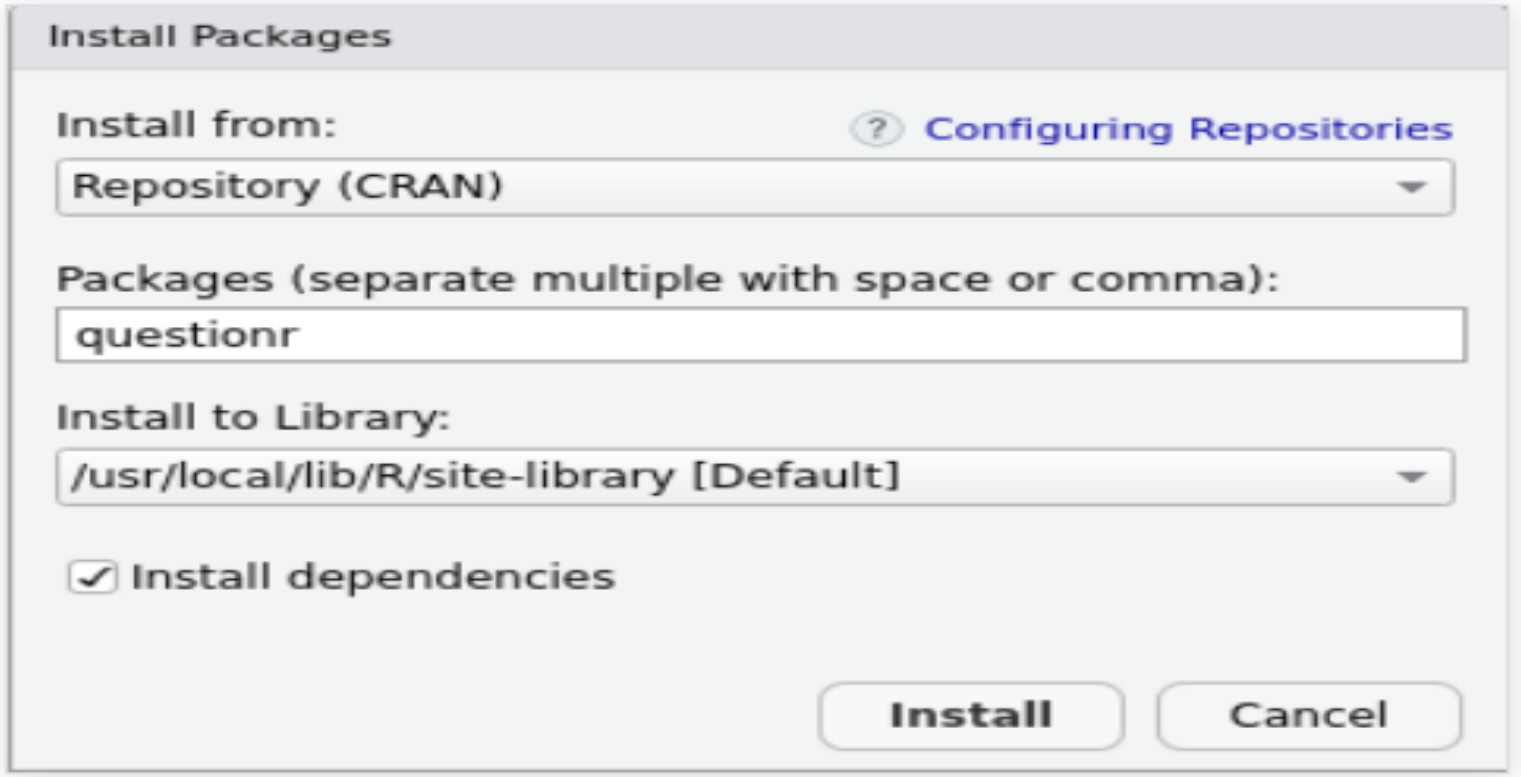
On peut donc documenter le script précédent :

```r
# Saisie des tailles et poids des enquêtés
tailles <- c(156, 164, 197, 147, 173)
poids <- c(45, 59, 110, 44, 88)

# Calcul des tailles et poids moyens
mean(tailles)
mean(poids)

# Calcul de l'IMC (poids en kilo divisé par les tailles en mètre au carré)
imc <- poids / (tailles / 100) ^ 2
# Valeurs extrêmes de l'IMC
min(imc)
max(imc)
```

1.5 Installer et charger des extensions (*packages*)

R étant un logiciel libre, il bénéficie d'un développement communautaire riche et dynamique. L'installation de base de R permet de faire énormément de choses, mais le langage dispose en plus d'un système d'extensions permettant d'ajouter facilement de nouvelles fonctionnalités. La plupart des extensions sont développées et maintenues par la communauté des utilisateurs de R, et diffusées via un réseau de serveurs nommé CRAN (*Comprehensive R Archive Network*).
Pour installer une extension, si on dispose d'une connexion Internet, on peut utiliser le bouton *Install* de l'onglet *Packages* de RStudio.





On peut aussi installer des extensions en utilisant la fonction install.packages() directement dans la console. Par exemple, pour installer le *package* questionr on peut exécuter la commande :

```
install.packages("questionr")
```

Une fois l'extension installée, il faut la "charger" avant de pouvoir utiliser les fonctions qu'elle propose. Ceci se fait avec la fonction library. Par exemple, pour pouvoir utiliser les fonctions de questionr, vous devrez exécuter la commande suivante :

```
library(questionr)
```

Si vous essayez d'exécuter une fonction d'une extension et que vous obtenez le message d'erreur impossible de trouver la fonction, c'est certainement parce que vous n'avez pas exécuté la commande library correspondante.

1.6 Exercices

Exercice 1

Construire le vecteur x suivant :

```
[1] 120 134 256  12
```

Utiliser ce vecteur x pour générer les deux vecteurs suivants :

```
[1] 220 234 356 112
```

```
[1] 240 268 512  24
```

Exercice 2

On a demandé à 4 ménages le revenu des deux conjoints, et le nombre de personnes du ménage :

```
conjoint1 <- c(1200, 1180, 1750, 2100)
conjoint2 <- c(1450, 1870, 1690, 0)
nb_personnes <- c(4, 2, 3, 2)
```

Calculer le revenu total de chaque ménage, puis diviser par le nombre de personnes pour obtenir le revenu par personne de chaque ménage.

Exercice 3

Dans l'exercice précédent, calculer le revenu minimum et maximum parmi ceux du premier conjoint.

```
conjoint1 <- c(1200, 1180, 1750, 2100)
```

Recommencer avec les revenus suivants, parmi lesquels l'un des enquetés n'a pas voulu répondre :

```
conjoint1 <- c(1200, 1180, 1750, NA)
```

Exercice 4

Les deux vecteurs suivants représentent les précipitations (en mm) et la température (en °C) moyennes sur une ville, pour chaque mois de l'année, entre 1981 et 2010 :

```
temperature <- c(3.4, 4.8, 8.4, 11.4, 15.8, 19.4, 22.2, 21.6, 17.6, 13.4, 7
precipitations <- c(47.2, 44.1, 50.4, 74.9, 90.8, 75.6, 63.7, 62, 87.5, 98.
```

Calculer la température moyenne sur l'année.
Calculer la quantité totale de précipitations sur l'année.

À quoi correspond et comment peut-on interpréter le résultat de la fonction suivante ? Vous pouvez vous aider de la page d'aide de la fonction si nécessaire.

```
cumsum(precipitations)
```

```
 [1]  47.2  91.3 141.7 216.6 307.4 383.0 446.7 508.7 596.2 694.8 776.7
[12] 831.9
```

Même question pour :

```
diff(temperature)
```

```
 [1]  1.4  3.6  3.0  4.4  3.6  2.8 -0.6 -4.0 -4.2 -5.8 -3.2
```

Exercice 5

On a relevé les notes en maths, anglais et sport d'une classe de 6 élèves et on a stocké ces données dans trois vecteurs :

```
maths <- c(12, 16, 8, 18, 6, 10)
anglais <- c(14, 9, 13, 15, 17, 11)
sport <- c(18, 11, 14, 10, 8, 12)
```

Calculer la moyenne des élèves de la classe en anglais.
Calculer la moyenne générale de chaque élève.

Essayez de comprendre le résultat des deux fonctions suivantes (vous pouvez vous aider de la page d'aide de ces fonctions) :

```
pmin(maths, anglais, sport)
```

```
[1] 12  9  8 10  6 10
```

```
pmax(maths, anglais, sport)
```

```
[1] 18 16 14 18 17 12
```

Premier travail avec des données

2.1 Jeu de données d'exemple

Dans cette partie nous allons travailler sur des "vraies" données, et utiliser un jeu de données présent dans l'extension questionr. Nous devons donc avant toute chose installer cette extension.
Pour installer ce package, deux possibilités :

- Dans l'onglet *Packages* de la zone de l'écran en bas à droite, cliquez sur le bouton *Install*
- Dans le dialogue qui s'ouvre, entrez "questionr" dans le champ *Packages* puis cliquez Sur *Install*.
- Saisissez directement la commande suivante dans la console :

```
install.packages("questionr")
```

Le jeu de données que nous allons utiliser est un extrait de l'enquête *Histoire de vie* réalisée par l'INSEE en 2003. Il contient 2000 individus et 20 variables. Pour pouvoir utiliser ces données, il faut d'abord charger l'extension questionr (après l'avoir installée, bien entendu) :

L'utilisation de library permet de rendre "disponibles", dans notre session R, les fonctions et jeux de données inclus dans l'extension.
Nous devons ensuite indiquer à R que nous souhaitons accéder au jeu de données à l'aide de la commande data :

```
data(hdv2003)
```

```
View(hdv2003)
```

Filter

	id	age	sexe	nivetud	poids	occup	qualif	freres.sc
1	1	28	Femme	Enseignement superieur y compris technique sup...	2634.3982	Exerce une profession	Employe	
2	2	23	Femme	*NA*	9738.3958	Etudiant, eleve	*NA*	
3	3	59	Homme	Derniere annee d'etudes primaires	3994.1025	Exerce une profession	Technicien	
4	4	34	Homme	Enseignement superieur y compris technique sup...	5731.6615	Exerce une profession	Technicien	
5	5	71	Femme	Derniere annee d'etudes primaires	4329.0940	Retraite	Employe	
6	6	35	Femme	Enseignement technique ou professionnel court	8674.6994	Exerce une profession	Employe	
7	7	60	Femme	Derniere annee d'etudes primaires	6165.8035	Au foyer	Ouvrier qualifie	
8	8	47	Homme	Enseignement technique ou professionnel court	12891.6408	Exerce une profession	Ouvrier qualifie	
9	9	20	Femme	*NA*	7808.8721	Etudiant, eleve	*NA*	
10	10	28	Homme	Enseignement technique ou professionnel long	2277.1605	Exerce une profession	Autre	
11	11	65	Femme	Enseignement superieur y compris technique sup...	704.3227	Retraite	Employe	
12	12	47	Homme	2eme cycle	6697.8682	Exerce une profession	Ouvrier qualifie	
13	13	63	Femme	Derniere annee d'etudes primaires	7118.4659	Retraite	Employe	
14	14	67	Femme	Enseignement technique ou professionnel court	586.7714	Exerce une profession	*NA*	
15	15	76	Femme	A arrete ses etudes, avant la derniere annee d'et...	11042.0774	Retraite	*NA*	
16	16	49	Femme	Enseignement technique ou professionnel court	9958.2287	Exerce une profession	Employe	
17	17	62	Homme	Enseignement superieur y compris technique sup...	4836.1393	Retraite	Cadre	
18	18	20	Femme	*NA*	1551.4846	Etudiant, eleve	*NA*	

Showing 1 to 19 of 2,000 entries

Pour résumer, comme nous avons désormais décidé de saisir nos commandes dans un script et non plus directement dans la console, les premières lignes de notre fichier de travail sur les données de l'enquête *Histoire de vie* pourraient donc ressembler à ceci :

```
## Chargement des extensions nécessaires
library(questionr)


## Jeu de données hdv2003
data(hdv2003)
d <- hdv2003
```

2.2.1 Structure du tableau

Un tableau étant un objet comme un autre, on peut lui appliquer des fonctions.
Par exemple, nrow et ncol retournent le nombre de lignes et de colonnes du tableau :

```
nrow(d)
```

```
[1] 2000
```

```
ncol(d)
```

```
[1] 20
```

```
names(d)
```

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

```
str(d)
```

```
'data.frame':   2000 obs. of  20 variables:
 $ id           : int  1 2 3 4 5 6 7 8 9 10 ...
 $ age          : int  28 23 59 34 71 35 60 47 20 28 ...
 $ sexe         : Factor w/ 2 levels "Homme","Femme": 2 2 1 1 2 2 2 1 2 1 ...
 $ nivetud      : Factor w/ 8 levels "N'a jamais fait d'etudes",..: 8 NA 3 8 
 $ poids        : num  2634 9738 3994 5732 4329 ...
 $ occup        : Factor w/ 7 levels "Exerce une profession",..: 1 3 1 1 4 1 
 $ qualif       : Factor w/ 7 levels "Ouvrier specialise",..: 6 NA 3 3 6 6 2 
 $ freres.soeurs: int  8 2 2 1 0 5 1 5 4 2 ...
 $ clso         : Factor w/ 3 levels "Oui","Non","Ne sait pas": 1 1 2 2 1 2 1
 $ relig        : Factor w/ 6 levels "Pratiquant regulier",..: 4 4 4 3 1 4 3 
 $ trav.imp     : Factor w/ 4 levels "Le plus important",..: 4 NA 2 3 NA 1 NA
 $ trav.satisf  : Factor w/ 3 levels "Satisfaction",..: 2 NA 3 1 NA 3 NA 2 NA
 $ hard.rock    : Factor w/ 2 levels "Non","Oui": 1 1 1 1 1 1 1 1 1 1 ...
 $ lecture.bd   : Factor w/ 2 levels "Non","Oui": 1 1 1 1 1 1 1 1 1 1 ...
 $ peche.chasse : Factor w/ 2 levels "Non","Oui": 1 1 1 1 1 1 2 2 1 1 ...
 $ cuisine      : Factor w/ 2 levels "Non","Oui": 2 1 1 2 1 1 2 2 1 1 ...
 $ bricol       : Factor w/ 2 levels "Non","Oui": 1 1 1 2 1 1 1 2 1 1 ...
 $ cinema       : Factor w/ 2 levels "Non","Oui": 1 2 1 2 1 2 1 1 2 2 ...
 $ sport        : Factor w/ 2 levels "Non","Oui": 1 2 2 2 1 2 1 1 1 2 ...
 $ heures.tv    : num  0 1 0 2 3 2 2.9 1 2 2 ...
```

2.2.2 Accéder aux variables d'un tableau

Une opération très importante est l'accès aux variables du tableau (à ses colonnes) pour pouvoir les manipuler, effectuer des calculs, etc. On utilise pour cela l'opérateur $, qui permet d'accéder aux colonnes du tableau. Ainsi, si l'on tape :

```
d$sexe
```

2.2.3 Créer une nouvelle variable

On peut aussi utiliser l'opérateur $ pour créer une nouvelle variable dans notre tableau : pour cela, il suffit de lui assigner une valeur.
Par exemple, la variable heures.tv contient le nombre d'heures passées quotidiennement devant la télé :

```
head(d$heures.tv, 10)
```

```
 [1] 0.0 1.0 0.0 2.0 3.0 2.0 2.9 1.0 2.0 2.0
```

On peut vouloir créer une nouvelle variable dans notre tableau qui contienne la même durée mais en minutes. On va donc créer une nouvelle variables minutes.tv de la manière suivante :

```
d$minutes.tv <- d$heures.tv * 60
```

On peut alors constater, soit visuellement soit dans la console, qu'une nouvelle variable (une nouvelle colonne) a bien été ajoutée au tableau :

```
head(d$minutes.tv)
```

```
[1]   0  60   0 120 180 120
```

2.3 Analyse univariée

On a donc désormais accès à un tableau de données d, dont les lignes sont des observations (des individus enquêtés), et les colonnes des variables (des caractéristiques de chacun de ces individus), et on sait accéder à ces variables grâce à l'opérateur $.
Si on souhaite analyser ces variables, les méthodes et fonctions utilisées seront différentes selon qu'il s'agit d'une variable *quantitative* (variable numérique pouvant prendre un grand nombre de valeurs : l'âge, le revenu, un pourcentage…) ou d'une variable *qualitative* (variable pouvant prendre un nombre limité de valeurs appelées modalités : le sexe, la profession, le dernier diplôme obtenu, etc.).

2.3.1 Analyser une variable quantitative

Une variable quantitative est une variable de type numérique (un nombre) qui peut prendre un grand nombre de valeurs. On en a plusieurs dans notre jeu de données, notamment l'âge (variable age) ou le nombre d'heures passées devant la télé (heures.tv)

2.3.1.1 Indicateurs de centralité

Caractériser une variable quantitative, c'est essayer de décrire la manière dont ses valeurs se répartissent, ou se distribuent.
Pour cela on peut commencer par regarder les valeurs extrêmes, avec les fonctions min, max ou range :

```r
min(d$age)
```

```
[1] 18
```

```r
max(d$age)
```

```
[1] 97
```

```r
range(d$age)
```

```
[1] 18 97
```

On peut aussi calculer des indicateurs de *centralité* : ceux-ci indiquent autour de quel nombre se répartissent les valeurs de la variable. Il y en a plusieurs, le plus connu étant la moyenne, qu'on peut calculer avec la fonction mean :

```
mean(d$age)
```

```
[1] 48.157
```

2.3.1.2 Indicateurs de dispersion

Les indicateurs de dispersion permettent de mesurer si les valeurs sont plutôt regroupées ou au contraire plutôt dispersées.
L'indicateur le plus simple est l'étendue de la distribution, qui décrit l'écart maximal observé entre les observations :

```
max(d$age) - min(d$age)
```

```
[1] 79
```

Les indicateurs de dispersion les plus utilisés sont la variance ou, de manière équivalente, l'écart-type (qui est égal à la racine carrée de la variance). On obtient la première avec la fonction var, et le second avec sd (abbréviation de *standard deviation*)

Plus la variance ou l'écart-type sont élevés, plus les valeurs sont dispersées autour de la moyenne. À l'inverse, plus ils sont faibles et plus les valeurs sont regroupées.
Une autre manière de mesurer la dispersion est de calculer les quartiles :

- le premier quartile est la valeur pour laquelle on a 25% des observations en dessous et 75% au dessus

- le deuxième quartile est la valeur pour laquelle on a 50% des observations en dessous et 50% au dessus (c'est donc la médiane)

- le troisième quartile est la valeur pour laquelle on a 75% des observations en dessous et 25% au dessus

On peut les calculer avec la fonction quantile :

```r
## Premier quartile
quantile(d$age, prob = 0.25)
```

```
25%
 35
```

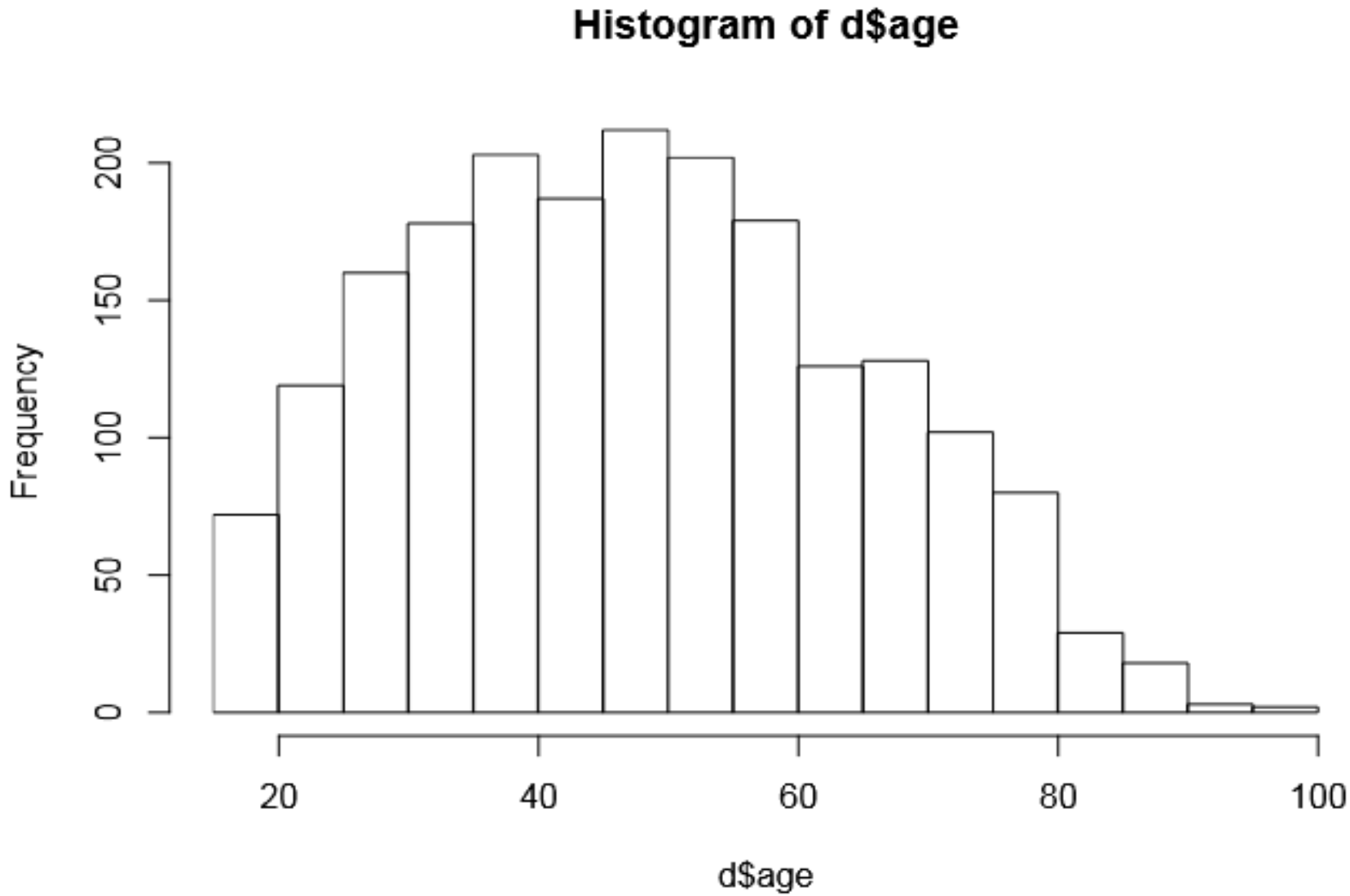
```r
## Troisième quartile
quantile(d$age, prob = 0.75)
```

```
75%
 60
```

2.3.1.3 Représentation graphique

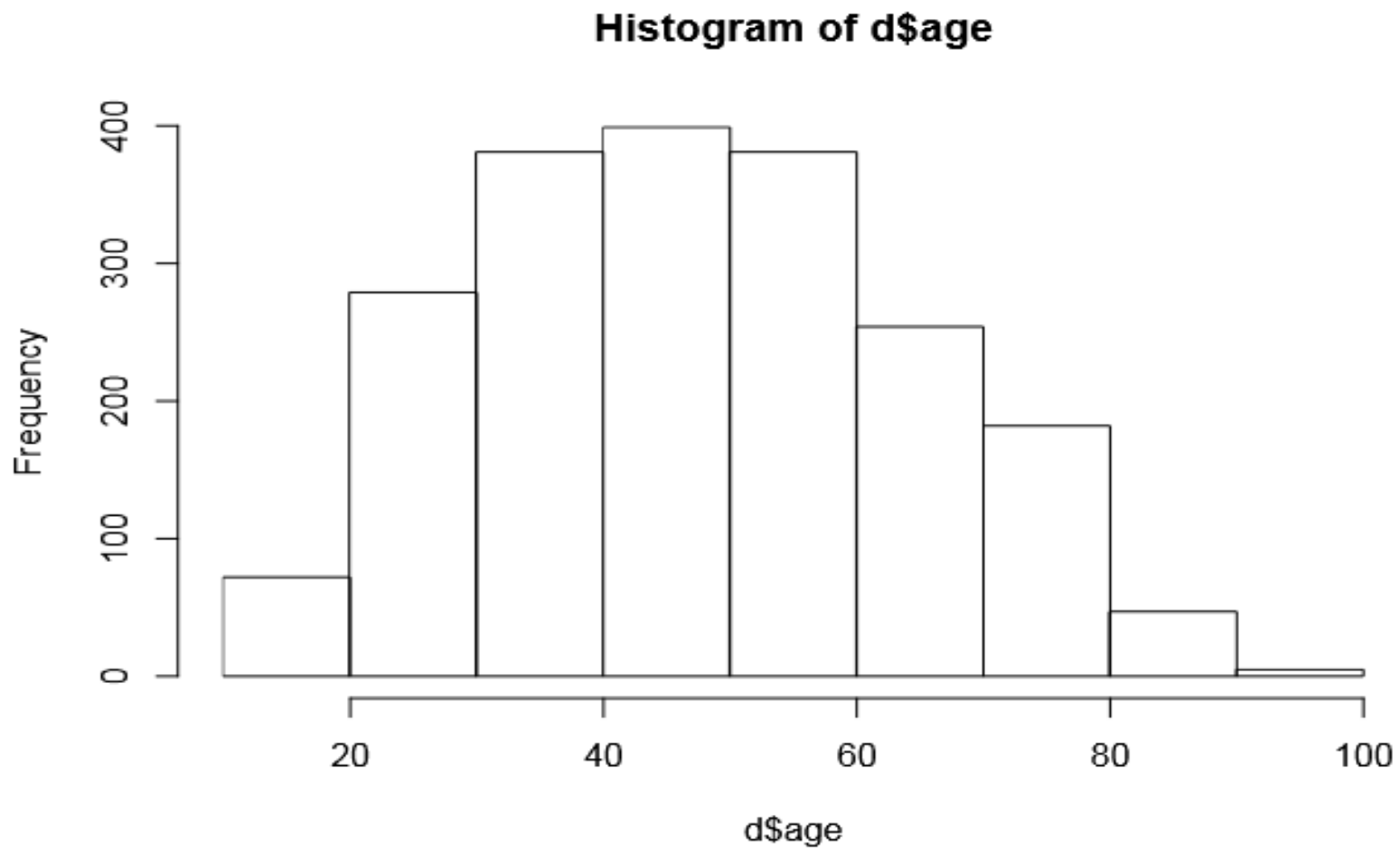
L'outil le plus utile pour étudier la distribution des valeurs d'une variable quantitative reste la représentation graphique.
La représentation la plus courante est sans doute l'histogramme. On peut l'obtenir avec la fonction hist :

```r
hist(d$age)
```

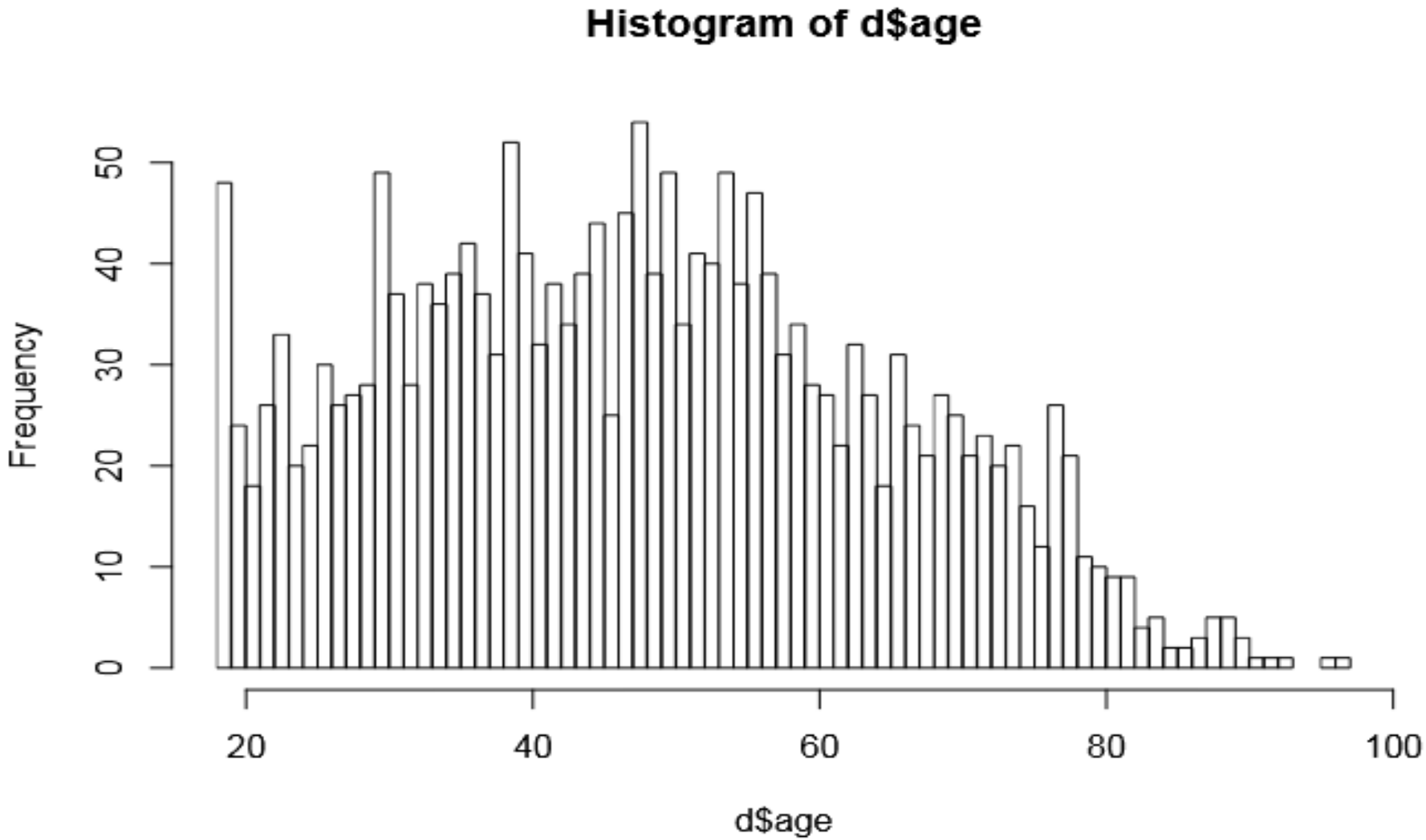

Histogram of d$age
Frequency
0
50
100
150
200
20
40
60
80
100
d$age

On peut personnaliser l'apparence de l'histogramme en ajoutant des arguments supplémentaires à la fonction hist. L'argument le plus important est breaks, qui permet d'indiquer le nombre de classes que l'on souhaite.

```r
hist(d$age, breaks = 10)
```

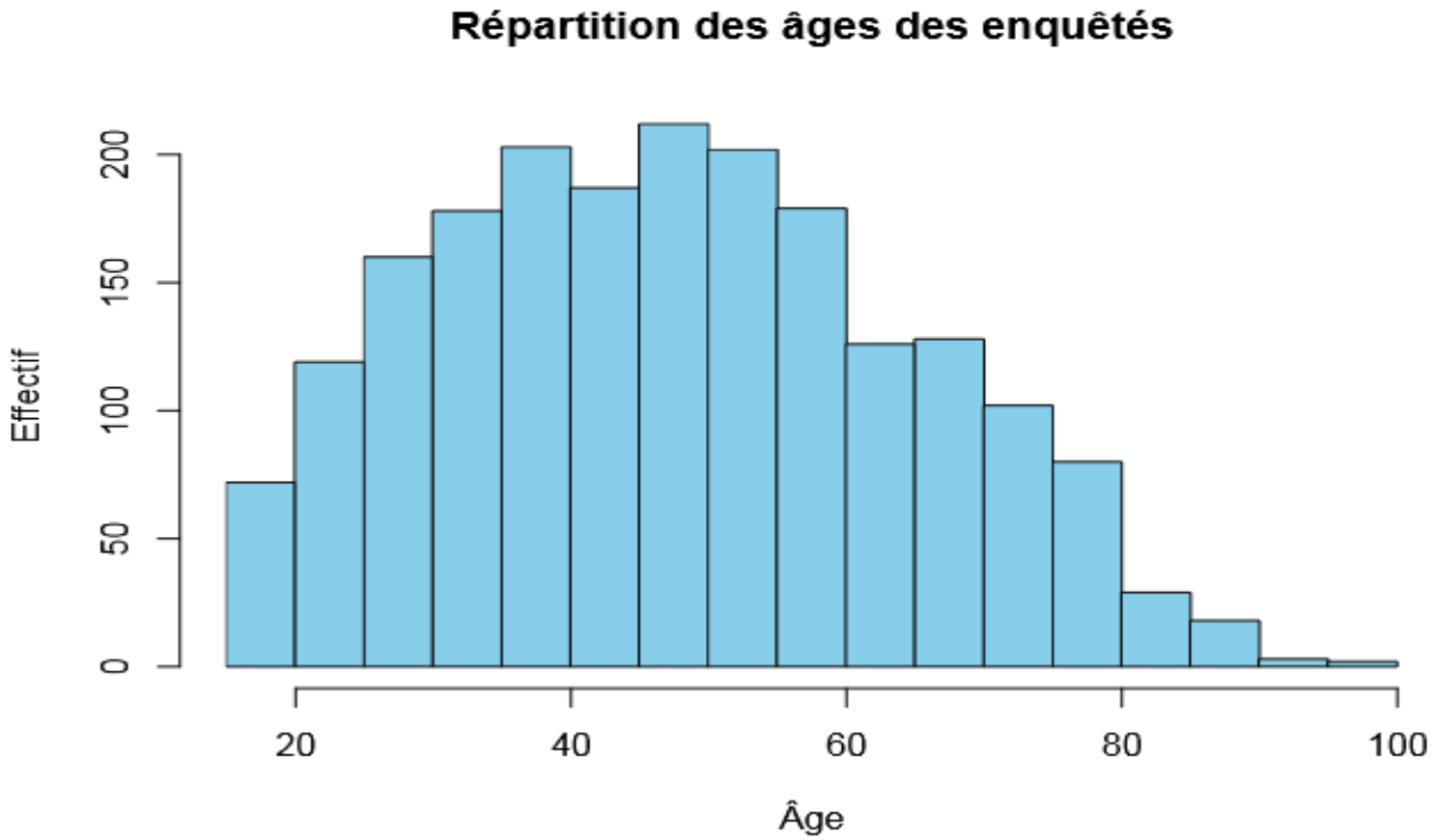


```r
hist(d$age, breaks = 70)
```



Les arguments de hist permettent également de modifier la présentation du graphique.
On peut ainsi changer la couleur des barres avec col, le titre avec main, les étiquettes des axes avec xlab et ylab, etc. :

```
hist(d$age, col = "skyblue",
     main = "Répartition des âges des enquêtés",
     xlab = "Âge",
     ylab = "Effectif")
```



2.3.2 Analyser une variable qualitative

Une variable qualitative est une variable qui ne peut prendre qu'un nombre limité de valeurs, appelées modalités. Dans notre jeu de données on trouvera par exemple le sexe (sexe), le niveau d'études (nivetud), la catégorie socio-professionnelle (qualif)…
À noter qu'une variable qualitative peut tout-à-fait être numérique, et que certaines variables peuvent être traitées soit comme quantitatives, soit comme qualitatives : c'est le cas par exemple du nombre d'enfants ou du nombre de frères et soeurs.

2.3.2.1 Tri à plat

L'outil le plus utilisé pour représenter la répartition des valeurs d'une variable qualitative est le *tri à plat* : il s'agit simplement de compter, pour chacune des valeurs possibles de la variable (pour chacune des modalités), le nombre d'observations ayant cette valeur. Un tri à plat s'obtient sous R à l'aide de la fonction table :

```
table(d$sexe)
```

```
Homme Femme
  899  1101
```

```
table(d$qualif)
```

```
      Ouvrier specialise         Ouvrier qualifie               Technicien
                     203                      292                       86
Profession intermediaire                   Cadre                   Employe
                     160                      260                      594
                   Autre
                      58
```

Un tableau de ce type peut être affiché ou stocké dans un objet, et on peut à son tour lui appliquer des fonctions. Par exemple, la fonction sort permet de trier le tableau selon la valeur de l'effectif. On peut donc faire :

```r
tab <- table(d$qualif)
sort(tab)
```

```
                  Autre                Technicien Profession intermediaire
                     58                        86                       160
      Ouvrier specialise                    Cadre          Ouvrier qualifie
                    203                       260                       292
                 Employe
                    594
```

Attention, par défaut la fonction table n'affiche pas les valeurs manquantes (NA).
Si on souhaite les inclure il faut utiliser l'argument useNA = "always", soit :
table(d$qualif, useNA = "always").

Par défaut ces tris à plat sont en effectifs et ne sont donc pas toujours très lisibles, notamment quand on a des effectifs importants. On leur rajoute donc en général la répartition en pourcentages. Pour cela, nous allons utiliser la fonction freq de l'extension questionr, qui devra donc avoir précédemment été chargée avec library(questionr) :

```
freq(d$qualif)
```

```
                          n    % val%
Ouvrier specialise      203 10.2 12.3
Ouvrier qualifie        292 14.6 17.7
Technicien               86  4.3  5.2
Profession intermediaire 160  8.0  9.7
Cadre                   260 13.0 15.7
Employe                 594 29.7 35.9
Autre                    58  2.9  3.5
NA                      347 17.3   NA
```

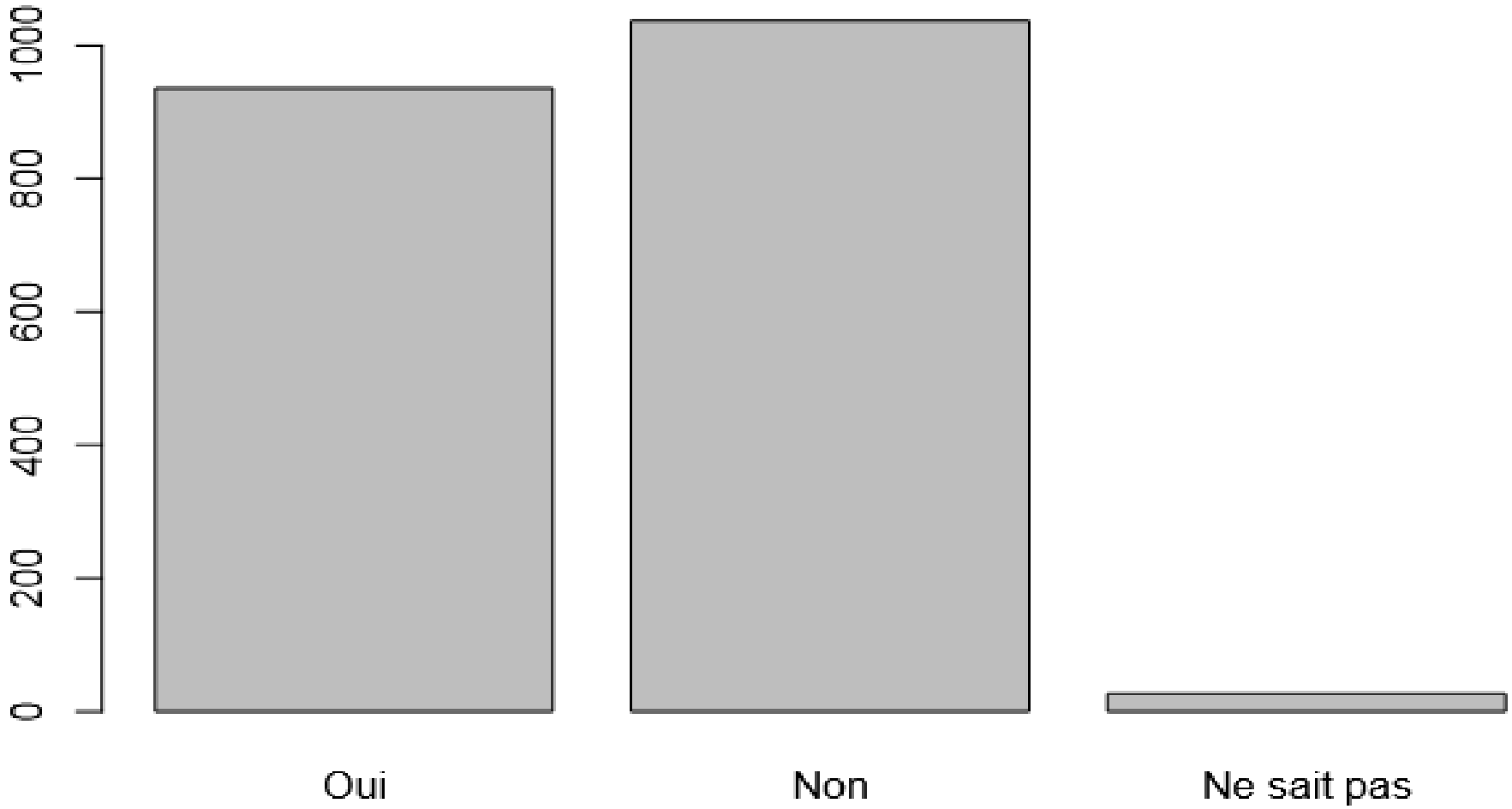
```
freq(d$qualif, valid= FALSE, total = TRUE, sort = "dec")
```

```
                            n     %
Employe                   594  29.7
Ouvrier qualifie          292  14.6
Cadre                     260  13.0
Ouvrier specialise        203  10.2
Profession intermediaire  160   8.0
Technicien                 86   4.3
Autre                      58   2.9
NA                        347  17.3
Total                    2000 100.0
```

2.3.2.2 Représentations graphiques

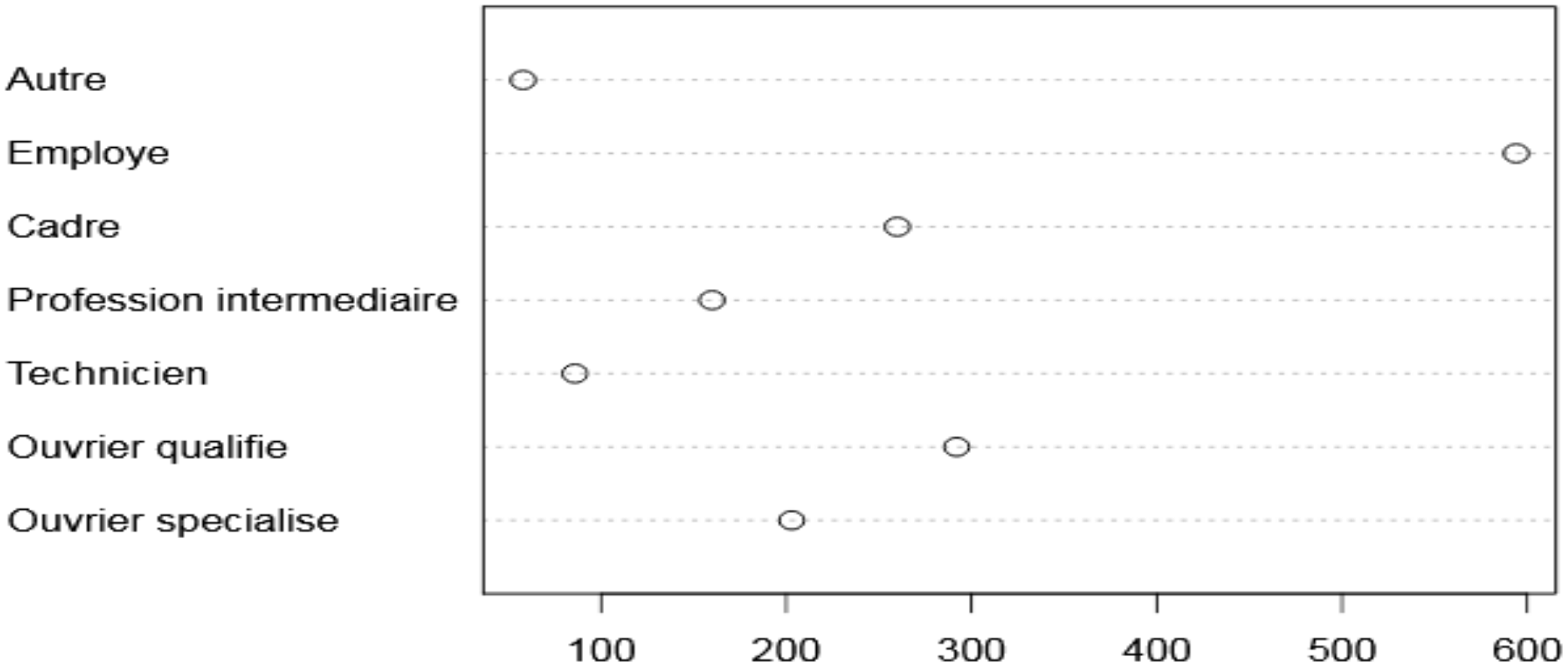
On peut représenter graphiquement le tri à plat d'une variable qualitative avec un diagramme en barres, obtenu avec la fonction barplot. Attention, contrairement à hist cette fonction ne s'applique pas directement à la variable mais au résultat du tri à plat de cette variable, calculé avec table. Il faut donc procéder en deux étapes :

```r
tab <- table(d$clso)
barplot(tab)
```

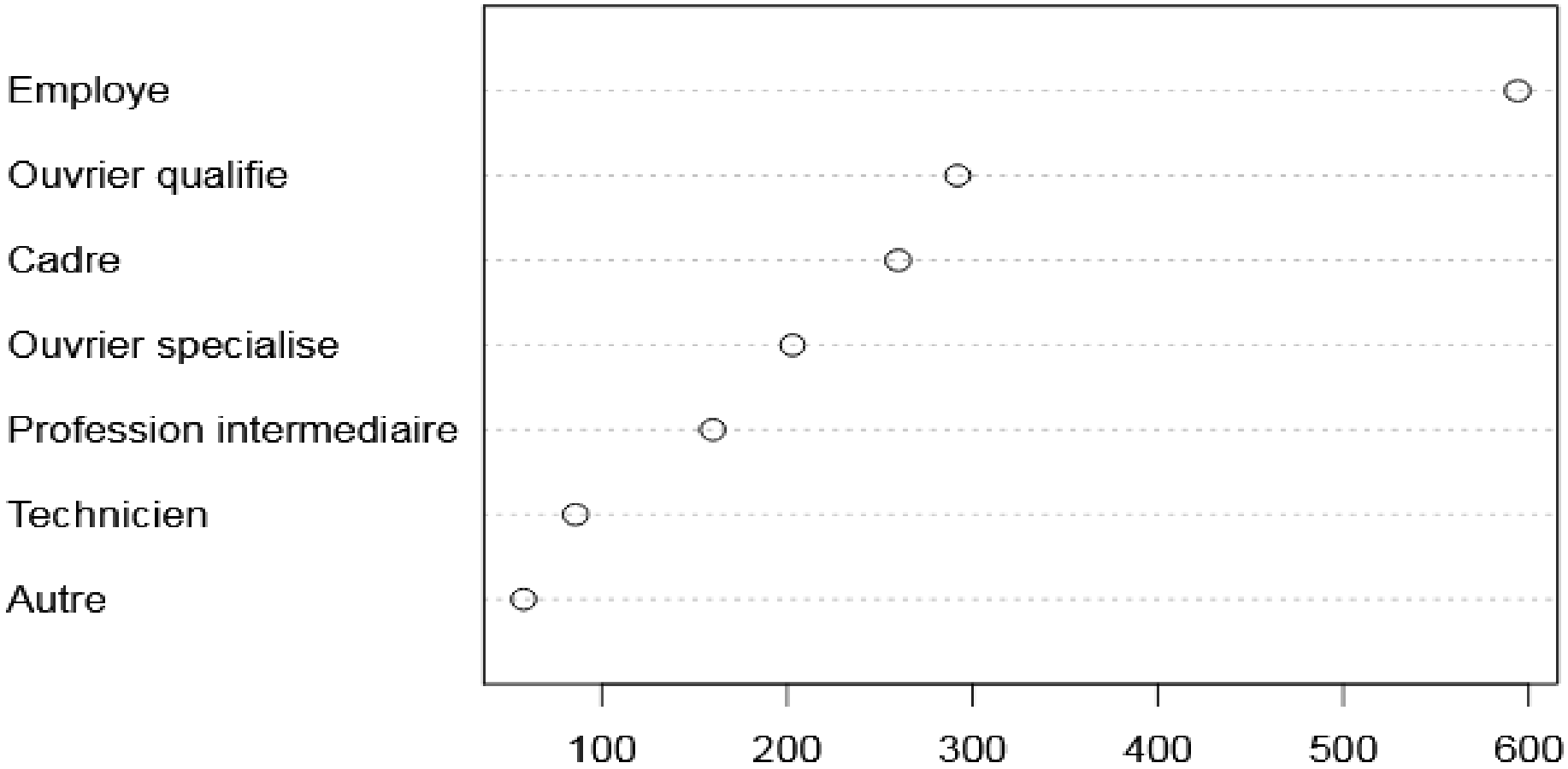


Une alternative au graphique en barres est le *diagramme de Cleveland*, qu'on peut obtenir avec la fonction dotchart. Celle-ci s'applique elle aussi au tri à plat de la variable calculé avec table.

```
dotchart(table(d$qualif))
```



```
dotchart(sort(table(d$qualif)))
```



2.4 Exercices

Exercice 1

Créer un nouveau script qui effectue les actions suivantes :

- charger l'extension questionr
- charger le jeu de données nommé hdv2003
- copier le jeu de données dans un nouvel objet nommé df
- afficher les dimensions et la liste des variables de df

Exercice 2

On souhaite étudier la répartition du temps passé devant la télévision par les enquêtés (variable heures.tv). Pour cela, affichez les principaux indicateurs de cette variable : valeur minimale, maximale, moyenne, médiane et écart-type. Représentez ensuite sa distribution par un histogramme en 10 classes.

Exercice 3

On s'intéresse maintenant à l'importance accordée par les enquêtés à leur travail (variable trav.imp).

1) Faites un tri à plat des effectifs des modalités de cette variable avec la commande table.

2) Faites un tri à plat affichant à la fois les effectifs et les pourcentages de chaque modalité. Y'a-t-il des valeurs manquantes ?

3) Représentez graphiquement les effectifs des modalités à l'aide d'un graphique en barres.

4) Utilisez l'argument col de la fonction barplot pour modifier la couleur du graphique en tomato.

5) Tapez colors() dans la console pour afficher l'ensemble des noms de couleurs disponibles dans R. Testez chaque couleur une à une pour trouver votre couleur préférée.