Guide R

Mégane Bollenrücher

2025-02-11

Contents

Merci de prendre note que ce bookdown est en cours de rédaction.

Chapter 1

Introduction

Ceci est le guide R que nous proposons pour vous accompagner durant les travaux pratiques de méthodologique.

Chapter 2

Installation et environnement R et Rstudio

2.1 Présentation des logiciels

R est un langage de programmation adapté au traitement de données et à l'analyse statistique.

Pour programmer en langage R, il est nécessaire d'installer deux outils essentiels:

1. Le **logiciel R** permet de traduire du texte sous forme de code R en binaire qui est le langage interne du processeur de l'ordinateur.
2. Le **logiciel RStudio** permet de faciliter l'utilisation du logiciel R en donnant l'accès à une interface utilisateur.

Il est possible de faire une analogie avec une voiture. Le logiciel R est le moteur et RStudio est le tableau de bord. Sans le tableau de bord, il n'est pas possible de controler le moteur.

2.2 Installation

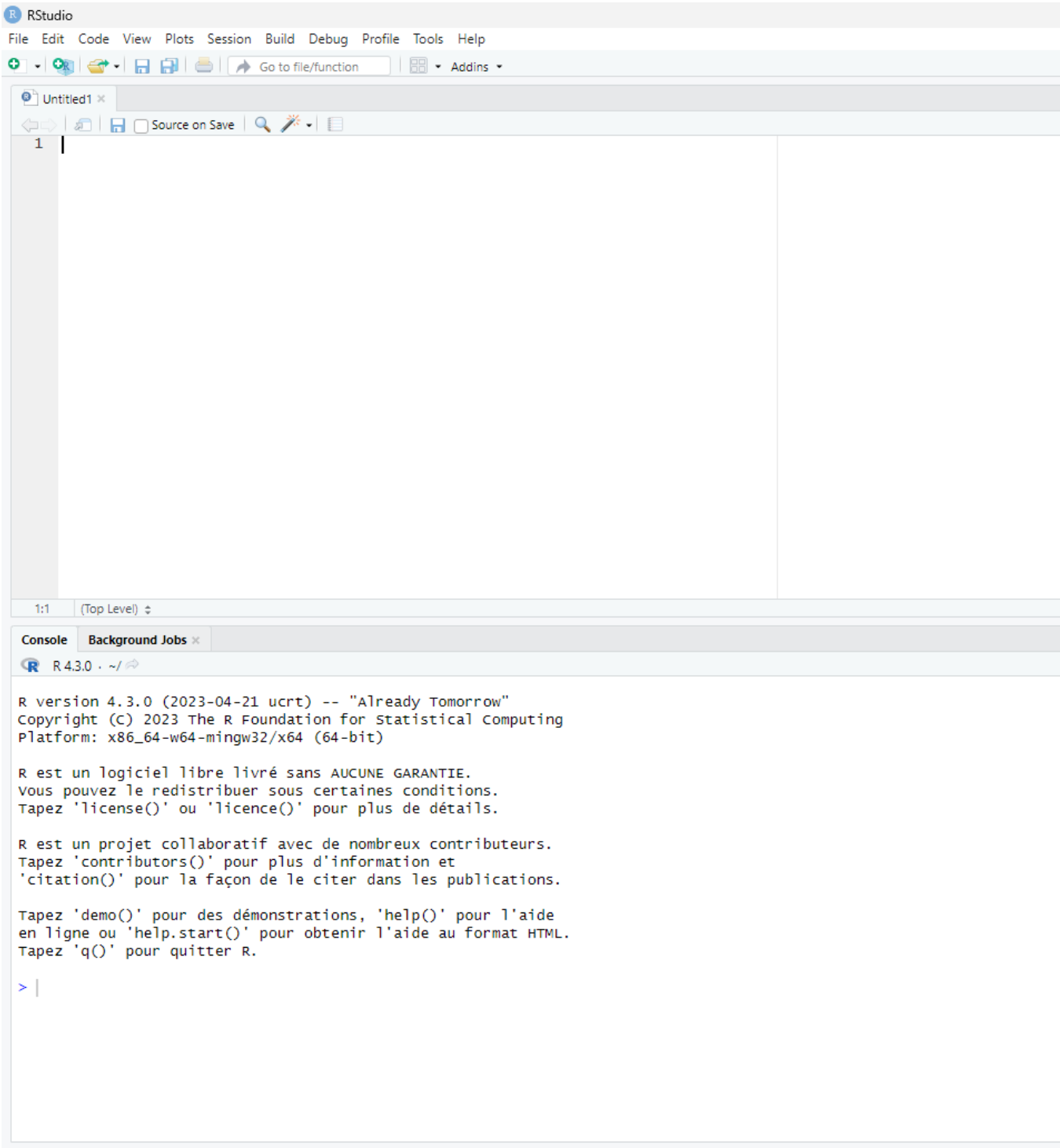
1. Installer R sur le site de R.
    i. Choisir et télécharger la version de R selon votre système d'exploitation.
        - Pour windows : https://cran.r-project.org/bin/windows/base/
        - Pour MAC : https://cran.r-project.org/bin/macosx/
        - Pour Linux : https://cran.r-project.org/index.html

   ii. Installer le logiciel R sur votre ordinateur en exécutant le fichier téléchargé.
2. Installer RStudio sur le site suivant: https://posit.co/download/rstudio-desktop/

Après avoir installé ces deux logiciels, vous aurez accès à deux nouvelles applications. Cependant, nous utiliserons uniquement RStudio pour programmer. Lorsque vous exécuterez votre code écrit sur RStudio, ce dernier fera automatiquement appel à R pour exécuter les codes.

2.3 Environnement de travail

Une fois que RStudio est lancé, une interface découpée en plusieurs zones se présente. Ces parties parties peuvent être redimensionnées, masquées ou maximisées selon vos préférences.


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Untitled1
Source on Save
1
1:1
(Top Level)
Console
Background Jobs
R 4.3.0 · ~/
R version 4.3.0 (2023-04-21 ucrt) -- "Already Tomorrow"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (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.
>

Chacune des quatre zones a sa propre utilité:

1. Cette zone est dédiée aux fichiers sources. Ce volet permet d'écrire et de sauvegarder les lignes de code. Ce sera la partie la plus utilisée lors de la programmation. Un nouveau script peut être ouvert à partir de l'onglet `File` en haut à gauche de l'écran, puis `New File` et `R Script`. Chaque script peut être enregistré soit via le même onglet en choisissant `Save As`, soit en utilisant les raccourcis classiques de votre clavier.
2. Cette zone fournit des informations sur les objets, les variables et les données en mémoire sous l'onglet `Environment`.
3. La console est affichée en bas à gauche. Cette partie permet d'entrer et d'exécuter des instructions et voir les résultats s'afficher.
4. Cette zone permet de naviguer dans le répertoire de travail dans l'onglet `Files`, d'afficher les graphes réalisés dans l'onglet `Plots`, d'afficher les extensions/packages disponibles sous l'onglet `Packages` et également d'afficher l'aide (qui est très complète) sous l'onglet `Help`.

Chapter 3

Objets et opérateurs

3.1 Objets dans R

Une variable permet de stocker une valeur ou un objet dans R. De cette façon, il sera possible d'accéder à la valeur ou à l'objet qui est stocké dans la variable.

```
# Assignation de la valeur 3 à la variable "ma_variable"
ma_variable <- 3
```

La ligne de code ci-dessus déclare une variable nommée "ma_variable" et lui assigne la valeur de 3. L'exécution de ce code n'affiche pas de résultat dans la console, mais l'objet nommé "ma_variable" est bien créée et stockée dans l'environnement.

Pour afficher le contenu de la variable, il suffit de taper le nom de celle-ci pour l'afficher dans la console.

```
# Affichage du contenu de la variable "ma_variable"
ma_variable
```

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

De plus, il est également possible d'utiliser la fonction `print()` qui permet d'afficher la valeur ou l'objet de la variable sélectionnée.

```
# Affichage du contenur de la variable "ma_variable"
print(ma_variable)
```

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

En langage R, il existe différents types d'objets qui peuvent être assignés à des variables comme les scalaires, les vecteurs, les facteurs, les matrices et les bases de données. Ces objets sont présentés dans les points suivants.

3.1.1 Scalaire

Un scalaire permet de stocker un objet sous forme de valeur numérique, de chaîne de caractères ou de valeur logique.

```
# Scalaire numerique
a <- 3
a
```

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

Une chaîne de caractères est une suite de caractères qui doit être écrit entre guillemets (" ").

```
# Scalaire sous forme d'une chaine de caractères
b <- "Statistique"
b
```

```
## [1] "Statistique"
```

Une valeur logique est une quantitée binaire (vrai ou faux). Ces variables s'écrivent `TRUE` et `FALSE`. Il est également possible d'utiliser `T` et `F` comme abréviations.

```
# Scalaire logique
c <- TRUE
c
```

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

```
d <- F
d
```

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

3.1.2 Vecteur

Un vecteur est un objet qui permet de stocker une liste ordonnée d'éléments. Les éléments d'un vecteur doivent être du même type. Pour pouvoir stocker une information dans un seul objet, il faut utiliser la fonction `c()` qui permet de combiner les arguments de la fonction.

```
# Vecteur numérique
a <- c(1, 2, 3, 4, 5, 6)
a
```

```
## [1] 1 2 3 4 5 6
```

```
# Vecteur sous forme d'une chaine de caractères
b <- c("Un", "Deux", "Trois", "Quatre", "Cinq", "Six")
b
```

```
## [1] "Un"     "Deux"   "Trois"  "Quatre" "Cinq"   "Six"
```

```
# Vecteur logique
c <- c(TRUE, FALSE, TRUE, FALSE, FALSE, TRUE)
c
```

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

Étant donné que le vecteur est un objet ordonné, il est possible d'accéder, remplacer ou modifier un ou plusieurs éléments par rapport à leur position dans l'objet. Il est nécessaire d'utiliser les crochets `[ ]` pour indiquer le ou les éléments à manipuler.

Pour accéder un seul élément du vecteur, il suffit d'écrire le nom de la variable suivi de crochets contenant la position de l'élément sélectionné.

```
# Extraction d'un élément:
b[2]
```

```
## [1] "Deux"
```

Il est possible d'accéder à plusieurs éléments en mettant un vecteur de position entre crochets.

```
# Extraction de plusieurs éléments:
b[c(2,3,5)]
```

```
## [1] "Deux"  "Trois" "Cinq"
```

Il est aussi possible d'accéder à une série d'éléments à la suite en mettant le signe `:` entre les 2 positions désirées.

```
# Extraction d'une série d'éléments:
b[2:5]
```

```
## [1] "Deux"   "Trois"  "Quatre" "Cinq"
```

Il est également possible d'accéder à certaines lignes en fonction de la valeur logique d'un vecteur ayant la même taille du vecteur sélectionné. Si la position est mise à `TRUE`, la valeur sera sélectionnée et dans le cas dans lequel la valeur est mise à `FALSE` la valeur ne sera pas retenue.

```
# Extraction de plusieurs éléments en fontion de la valeur logique:
b[c(FALSE, TRUE, TRUE, FALSE, TRUE, TRUE)]
```

```
## [1] "Deux"  "Trois" "Cinq"  "Six"
```

3.1.3 Facteur

Un facteur est un vecteur dont les éléments peuvent prendre que des valeurs prédéfinies. Un facteur dispose de l'argument `levels` qui permet de définir des catégories de valeurs. Le facteur est généralement utilisé pour stocker des variables catégorielles.

Pour commencer, il faut définir un vecteur qui peut être numérique, logique ou chaine de caractères. Le facteur est une variable nominale.

```
genre <- c("Homme", "Femme", "Femme", "Femme", "Homme")
genre
```

```
## [1] "Homme" "Femme" "Femme" "Femme" "Homme"
```

La fonction `factor()` permet de créer un facteur à partir d'un vecteur.

```
genre <- factor(genre)
genre
```

```
## [1] Homme Femme Femme Femme Homme
## Levels: Femme Homme
```

On note que la sortie est légèrement différente lorsque le vecteur est mis sous forme de facteur à 2 niveaux (Femme, Homme). Ceci s'affiche à la ligne `Levels`. Par défaut, les niveaux d'un facteur sont affichés par ordre alphabétique et numérique croissant. Il est possible de fixer l'ordre en ajoutant l'argument `levels` en appliquant la fonction `factor()`.

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

```
## [1] H F F F H
## Levels: H F
```

Dans le cas dans lequel on aimerait modifier un élément, il n'est pas possible d'affecter une valeur qui n'est pas défini comme un niveau. On voit donc apparaître une erreur dans la console.

```
genre[2] <- "Fille"
```

```
## Warning in `[<-.factor`(`*tmp*`, 2, value = "Fille"): niveau de facteur
## incorrect, NAs générés
```

Il est possible de renommer les niveaux en utilisant la fonction `levels()`. **Il faut faire attention à l'ordre lorsqu'on utilise la fonction `levels()`.** L'argument `order` permet d'ordonner les labels proposés. Le facteur est dès lors une variable ordinale.

```
levels(genre) <- c("Fille", "Garcon")
genre
```

```
## [1] Garcon <NA>   Fille  Fille  Garcon
## Levels: Fille Garcon
```

Il est possible d'avoir un facteur numérique en y affectant une catégorie avec l'argument `labels` lors de l'utilisation de la fonction `factor()`.

```
satisfaction <- factor(c(3, 3, 4, 1, 2, 1, 1),
                       labels = c("Pas du tout d'accord", "Pas d'accord",
                                  "D'accord", "Tout à fait d'accord"),
                       order = TRUE)
satisfaction
```

```
## [1] D'accord             D'accord             Tout à fait d'accord
## [4] Pas du tout d'accord Pas d'accord         Pas du tout d'accord
## [7] Pas du tout d'accord
## 4 Levels: Pas du tout d'accord < Pas d'accord < ... < Tout à fait d'accord
```

3.1.4 Matrice

Une matrice est un vecteur dont les éléments sont disposés sous forme d'un tableau qui comporte des lignes et des colonnes. De façon équivalente au vecteur, les éléments de la matrices doivent être de même classe (numérique, logique ou chaîne de caractères). La fonction `matrix()` permet de déclarer une matrice. Il faut ajouter l'argument `ncol` et/ou `nrow` pour déterminer la forme de la matrice.

```
A <- matrix(1:24, nrow=6, ncol=4, byrow=FALSE)
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    7   13   19
## [2,]    2    8   14   20
## [3,]    3    9   15   21
## [4,]    4   10   16   22
## [5,]    5   11   17   23
## [6,]    6   12   18   24
```

Par défaut, le remplissage se fait par colonne. Il faut donc mettre l'argument `byrow` à `TRUE` pour remplir la matrice par ligne.

```
B <- matrix(1:24, nrow=6, ncol=4, byrow=TRUE)
B
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
## [5,]   17   18   19   20
## [6,]   21   22   23   24
```

L'objet matrice dispose de la fonction `dim()` qui permet d'obtenir sa dimension. Le premier terme correspond aux nombres de lignes et le deuxième correspond aux nombres de colonnes.

```r
dim(B)
```

```
## [1] 6 4
```

Les fonctions `rownames()` et `colnames()` permettent de récupérer ou de définir les noms des lignes et des colonnes. Attention de bien mettre le bon nombre de noms aux lignes et aux colonnes.

```r
rownames(B) <- c("L1", "L2", "L3", "L4", "L5", "L6")
colnames(B) <- c("C1", "C2", "C3", "C4")
B
```

```
##    C1 C2 C3 C4
## L1  1  2  3  4
## L2  5  6  7  8
## L3  9 10 11 12
## L4 13 14 15 16
## L5 17 18 19 20
## L6 21 22 23 24
```

Comme pour le vecteur, il est possible d'accéder à un ou plusieurs éléments de la matrice. Pour extraire une ligne de la matrice, il faut utiliser les crochets avec une virgule pour délimiter les deux dimensions de la matrice`[ , ]`. Le premier terme (celui avant la virgule) permet d'accéder aux colonnes et le deuxième terme (celui après la virgule) permet d'accéder aux lignes. Pour accéder à un seul élément, il faut indiquer la position de la ligne et de la colonne désirée.

```r
# Extraction d'un seul élément
A[2,3]
```

```
## [1] 14
```

Pour accéder à une ligne complète, il suffit de mettre la position de la ligne désirée avant la virgule.

```r
# Extraction d'une ligne
A[2, ]
```

```
## [1]  2  8 14 20
```

Pour accéder à une colonne complète, il suffit de mettre la position de la colonne désirée après la virgule.

```r
# Extraction d'une colonne
A[, 3]
```

```
## [1] 13 14 15 16 17 18
```

Pour accéder à un groupe d'élément, il faut déterminer l'intervalle des lignes et des colonnes désirées.

```
# Extraction de quelques éléments regroupées
A[3:5, 2:3]
```

```
##      [,1] [,2]
## [1,]    9   15
## [2,]   10   16
## [3,]   11   17
```

3.1.5 Dataframe

Un jeu de données se structure sous forme d'un tableau dans lequel chaque ligne correspond à une observation (individu) et chaque colonne à une caractéristique (variable). Les data frame sont les objets les plus utilisées lors de l'analyse d'une base de données. Contrairement aux vecteurs et aux matrices, une dataframe peut avoir différents type de variables (numérique, logique et chaînes de caractères). La fonction `data.frame()` permet la création de la base de données.

```
dataframe <- data.frame(
  ID = 1:5,
  Genre = c("Homme", "Femme", "Femme", "Femme", "Homme"),
  Age = c(45, 42, 45, 43, 44)
)
dataframe
```

```
##   ID Genre Age
## 1  1 Homme  45
## 2  2 Femme  42
## 3  3 Femme  45
## 4  4 Femme  43
## 5  5 Homme  44
```

Les colonnes d'une dataframe sont toujours nommées et correspondent à la variable mesurée. Les lignes sont automatiquement numérotées par ordre.

La fonction `str()` permet d'afficher la structure de la dataframe en affichat le nom de la variable, le type de celle-ci ainsi que les valeurs des observations.

```
# Structure
str(dataframe)
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ ID   : int  1 2 3 4 5
##  $ Genre: chr  "Homme" "Femme" "Femme" "Femme" ...
##  $ Age  : num  45 42 45 43 44
```

La fonction `View()` permet de visionner la data frame dans une autre fenêtre.

```
# Structure
View(dataframe)
```

Afin d'analyser les données, il est important de pouvoir d'en extraire uniquement une partie. Il existe deux façons d'extraire une colonne. La première consiste à reproduire le cas de la matrice en séléctionnant la position de la colonne.

```
# Extraction de colonnes
dataframe[, 2]
```

```
## [1] "Homme" "Femme" "Femme" "Femme" "Homme"
```

La deuxième option est d'utiliser le symbole `$`. Il doit être placé entre le nom de la data frame et le nom de la colonne. **Il est conseillé d'utiliser cette option pour extraire une colonne d'une data frame.**

```
# Extraction de colonnes
dataframe$Genre
```

```
## [1] "Homme" "Femme" "Femme" "Femme" "Homme"
```

Pour extraire une ligne de la base de donnée, il faut procéder comme pour la matrice.

```
# Extraction de ligne
dataframe[2, ]
```

```
##   ID Genre Age
## 2  2 Femme  42
```

Pour extraire les observations (lignes) qui possèdent certaines caractéristiques, il est possible d'écrire la ligne suivante comme suit:

```
# Extraction de ligne
dataframe[dataframe$Genre == "Homme", ]
```

```
##   ID Genre Age
## 1  1 Homme  45
## 5  5 Homme  44
```

Il est également possible de mettre plusieurs conditions.

```
# Extraction de ligne
dataframe[dataframe$Genre == "Femme" & dataframe$Age < 44, ]
```

```
##   ID Genre Age
## 2  2 Femme  42
## 4  4 Femme  43
```

Il est possible d'ajouter une colonne à la base de données. Plusieurs options sont possibles:

1. Créer un vecteur de même taille que la longueur de la base de données et l'ajouter à la base de données en utilisant la fonction `cbind()`.
2. Créer une nouvelle variable directement dans la base de données en déterminant son nom grâce au signe $.

```r
yeux <- c("brun", "brun", "bleu", "bleu", "brun")
dataframe <- cbind(dataframe, yeux)

dataframe$cheveux <- c("blond", "brun", "blond", "noir", "noir")

str(dataframe)
```

```
## 'data.frame':    5 obs. of  5 variables:
##  $ ID     : int  1 2 3 4 5
##  $ Genre  : chr  "Homme" "Femme" "Femme" "Femme" ...
##  $ Age    : num  45 42 45 43 44
##  $ yeux   : chr  "brun" "brun" "bleu" "bleu" ...
##  $ cheveux: chr  "blond" "brun" "blond" "noir" ...
```

3.2 Opérateurs logiques

Opérateur	Description
<	strictement inférieur
<=	inférieur ou égal
>	strictement supérieur
>=	supérieur ou égal
==	égal
!=	différent
!x	non x
x | y	x ou y
x & y	x et y

Chapter 4

Packages et données

Ce premier chapitre introduit deux concepts importants dans R. Le premier est les packages et le second concerne les données.

4.1 Installation et gestion des packages

Les packages sont des regroupements de fonctions et de jeux de données développés dans R et qui doivent se télécharger une seule fois, mais ils devront être importés à chaque utilisation. Le code ci-dessous permet d'installer le package ggplot2:

```
install.packages("ggplot2", dependencies = TRUE)
```

Avant chaque utilisation des packages, il est nécessaire d'importer le package grâce au code suivant:

```
library(ggplot2)
```

4.2 Téléchargement des données

Dans R, les bases de données se déclinent de plusieurs façons:

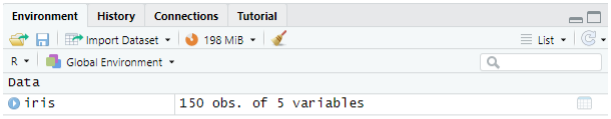
1. Les bases de données peuvent être directement incluses dans R ou dans les packages.
2. Les bases de données peuvent être créées dans l'environnement sauvées dans l'environnement R. Ces fichiers ont une extension .RData
3. Les bases de données peuvent être issues de fichiers externes. Ces fichiers peuvent avoir différentes extensions, les plus courantes étant .csv et .txt.

4.2.1 Bases de données issues de la base de R ou des packages

Le code suivant permet d'importer le jeu de données "iris" disponible de base dans R.

```
data(iris)
```

Un objet `iris` apparait dans l'environnement du projet comme le montre la figure suivante.



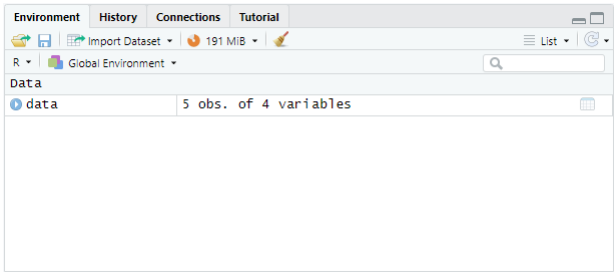
Si les données sont dans un package, le package doit être importé au préalable.

4.2.2 Bases de données issues dans un fichier .RData

Les fichiers `.RData` sont spécifique au language R et peuvent contenir plusieurs objets en un seul fichier. Pour importer des données, il faut utiliser la fonction `load()`

```
load("04-data.RData")
```

Tous les objets importés sont chargés dans l'environnement de travail.



4.2.3 Bases de données issues de fichiers externes

Les données à analyser sont souvent disponibles dans un fichier externe sous différents formats tels que .csv ou .txt. Pour importer ces données, il existe une fonction par type de fichier (`read.csv()` et `read.table()`). Lorsqu'une de ces fonctions est utilisée, le contenu est stocké dans une dataframe. Il est nécessaire de spécifier le chemin d'accès entre votre logiciel et votre fichier à télécharger.

Vous pouvez le faire de deux manières: 1. En utilisant la fonction `setwd()` et en spécifiant à l'intérieur la direction complète qui va permettre au logiciel de retrouver votre document dans vos fichiers. 1. En créant un projet dans lequel vous stockez vos documents. Ces fonctions acceptent des arguments qui permettent de s'adapter à la nature de fichier à importer. Parmi ces arguments, il y en a trois principaux:

1. `header` qui est une valeur logique (`TRUE` ou `FALSE`) pour la présence d'un en-tête avec les noms de variables. Cet argument est mis par défaut à `TRUE` pour la fonction `read.csv()` et à `FALSE` pour la fonction `read.table()`.
2. `sep` qui est le caractère dont les champs sont séparés. Cet argument est mis par défaut à `,`.
3. `dec` qui est le séparateur décimal. Cet argument est mis par défaut à `.`.

Le seul argument obligatoire est le chemin d'accès au fichier à lire. Il n'est pas nécessaire de spécifier le chemin complet si le fichier à lire se trouve dans le dossier du projet. Si ce n'est pas le cas, vous devez spécifier le chemin d'accès complet à partir de ce qui a été fait avec la fonction `setwd()`. Pour charger une base de donnée nommé `04-data.csv` dont les valeurs sont séparées par des virgules, il suffit d'écrire la ligne suivante:

```
csv_data <- read.csv("04-data.csv")
csv_data
```

```
##      Nom Age Note.1 Note.2
## 1   Marc  18    5.0   5.50
## 2   Anne  20    6.0   4.00
## 3 Marie   21    4.5   4.75
## 4   Jean  17    3.5   5.00
## 5 Sophie  18    5.0   4.75
```

Pour charger une base de donnée nommé `04-data.txt` dont les valeurs sont séparées par des points-virgules, il suffit d'écrire comme dans la ligne suivante en n'oubliant pas de sécifier le caractère de séparation avec l'argument `sep=";"`.

```
txt_data <- read.table("04-data.txt", sep = ";", header = TRUE)
txt_data
```

```
##      Nom Age Note.1 Note.2
## 1   Marc  18    5.0   5.50
## 2   Anne  20    6.0   4.00
## 3  Marie  21    4.5   4.75
## 4   Jean  17    3.5   5.00
## 5 Sophie  18    5.0   4.75
```

Il est également possible de charger une base de données issues d'un fichier externe en l'important depuis le menu `File > Import Dataset > From Text` puis de sélectionner le fichier dans vos dossiers.

Chapter 5

ANOVA à mesures répétées

5.1 ANOVA à mesures répétées à un facteur de classification

L'ANOVA à mesures répétées est utilisée lorsque les mêmes sujets sont mesurés plusieurs fois dans différentes conditions ou à différents moments. Pour effectuer l'ANOVA à mesures répétées, nous allons utiliser la fonction `anova_test()` du package `rstatix`. Dans ce chapitre, nous allons traiter de l'ANOVA à mesures répétées avec un et deux facteurs de classification. Nous présentons en détails la question du format, du traitement et de la description des donnés pour l'ANOVA à un facteur de classification. Les mêmes démarches sont à suivre pour l'ANOVA à deux facteurs pour laquelle nous présentons uniquement la marche à suivre de l'ANOVA en tant que telle et son interprétation.

5.1.1 Format, traitement et description des données

Afin d'utiliser cette fonction, les données doivent être au format long, à savoir le format dans lequel chaque ligne représente une mesure individuelle avec une colonne identifiant le sujet. La base de données avec laquelle nous travaillons contient trois variables, à savoir `participants` qui correspond à la variable permettant d'identifier les sujets de l'étude, la variable `condition` qui correspond aux temps de mesure, et la variable `score` qui est la variable indépendante mesurée. La variable `condition` est donc la variable inta-sujet et la variable `score` la mesure répétée. Dans le traitement des données, il est important de contrôler que la variable des sujets et la variable intra-sujet soient des facteurs, et que la variable à mesures répétées soit numérique. Si ce n'est pas le cas, il est important de les transformer. La fonction `unique()` permet d'extraire les éléments uniques afin de savoir combien de participants sont présents dans la base de données ou combien de temps de mesure il y a.

```r
library(dplyr) #manipulation des données
```

```
## Warning: le package 'dplyr' a été compilé avec la version R 4.3.1
```

```
##
## Attachement du package : 'dplyr'
```

```
## Les objets suivants sont masqués depuis 'package:stats':
##
##     filter, lag
```

```
## Les objets suivants sont masqués depuis 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(rstatix)
```

```
## Warning: le package 'rstatix' a été compilé avec la version R 4.3.3
```

```
##
## Attachement du package : 'rstatix'
```

```
## L'objet suivant est masqué depuis 'package:stats':
##
##     filter
```

```r
data <- read.csv("score.csv", header = TRUE, sep = ";", dec = ",")
str(data)
```

```
## 'data.frame':    15 obs. of  3 variables:
##  $ participants: int  1 1 1 2 2 2 3 3 3 4 ...
##  $ condition   : chr  "T1" "T2" "T3" "T1" ...
##  $ score       : num  12.1 14.1 15.3 11.8 13.2 14.7 15.2 16.7 11.7 10.3 ...
```

```r
data$participants <- as.factor(data$participants)
data$condition <- as.factor(data$condition)
str(data)
```

```
## 'data.frame':    15 obs. of  3 variables:
##  $ participants: Factor w/ 5 levels "1","2","3","4",..: 1 1 1 2 2 2 3 3 3 4 ...
##  $ condition   : Factor w/ 3 levels "T1","T2","T3": 1 2 3 1 2 3 1 2 3 1 ...
##  $ score       : num  12.1 14.1 15.3 11.8 13.2 14.7 15.2 16.7 11.7 10.3 ...
```

```r
unique(data$participants)
```

```
## [1] 1 2 3 4 5
## Levels: 1 2 3 4 5
```

```r
unique(data$condition)
```

```
## [1] T1 T2 T3
## Levels: T1 T2 T3
```

5.1.2 Effectuer et interpréter l'ANOVA

L'ANOVA s'effectue grâce à la fonction `anova_test()` en utilisant les arguments suivants:

1. **dv**: variable dépendante numérique
2. **within**: variable intra-sujet sous forme de facteur
3. **wid**: variable sous forme de facteur permettant d'identifier les sujets

Afin de pouvoir interpréter une ANOVA, des hypothèses doivent être vérifiées. Il s'agit de l'hypothèse de normalité des données, que l'on teste grâce à la fonction `shapiro_test()` avant d'effectuer l'ANOVA, et l'hypothèse de sphérécité des données, que l'on contrôle directement dans le résultat de l'ANOVA.

L'hypothèse nulle du test de la normalité des données est que les données suivent une loi normale. Si cette hypothèse n'est pas rejetée, alors l'ANOVA à mesures répétées peut être effectuée. L'hypothèse nulle du test de sphérécité considère l'égalité des variances des différences des groupes. Si l'hypothèse nulle n'est pas rejetée, alors aucune correction est nécessaire. Au contraire, une correction doit s'appliquer.

La fonction `get_anova_table()` permet d'obtenir le résultat de l'ANOVA. Nous proposons d'utiliser l'argument `correction = "auto"` car il permet d'effectuer la correction nécessaire si l'hypothèse de sphérécité des données est rejetée.

```
data %>%
  group_by(condition) %>%
  shapiro_test(score)
```

```
## # A tibble: 3 x 4
##   condition variable statistic     p
##   <fct>     <chr>        <dbl> <dbl>
## 1 T1        score        0.973 0.894
## 2 T2        score        0.969 0.872
## 3 T3        score        0.896 0.386
```

```
res <- anova_test(data, dv = score, within = condition, wid = participants)

print(res$`Mauchly's Test for Sphericity`)
```

```
##      Effect     W     p p<.05
## 1 condition 0.876 0.819
```

```
res %>% get_anova_table()
```

```
## ANOVA Table (type III tests)
##
##      Effect DFn DFd     F     p p<.05   ges
## 1 condition   2   8 0.446 0.655       0.077
```

```
res %>% get_anova_table(correction = "auto")
```

```
## ANOVA Table (type III tests)
##
##      Effect DFn DFd     F     p p<.05   ges
## 1 condition   2   8 0.446 0.655       0.077
```

Enfin d'interpréter le résultat de l'ANOVA, il est nécessaire d'analyser la table de l'ANOVA:

1. **Effet**: la variable intra-sujet
2. **DFn**: degrés de liberté du facteur intra-sujet
3. **DFd**: degrés de liberté de l'erreur intra-sujet
4. **F**: statistique F de Fisher
5. **p**: significativité du test
6. **ges**: taille de l'effet généralisée aussi appelée êta carré

La significativité du test permet de conclure sur la présence ou l'absence d'un effet significatif de la condition sur le score, donc de l'impact des temps de mesure sur le score. La taille d'effet permet d'évaluer l'importance de l'effet. Elle doit être analysée selon la grille d'analyse correspondante.

5.1.3 Test post-hoc

Si l'ANOVA est significative, il est important d'effectuer un test post-hoc pour savoir quelles conditions diffèrent. Le test post-hoc est effectué avec la fonction `pairwise_t_test()` pour comparer les données pair à pair avec une correction de Bonferroni. L'interprétation se fait comme dans le cas d'un test de Student à mesures répétées.

```
pairwise_t_test(data, score ~ condition, paired = TRUE,
                p.adjust.method = "bonferroni")
```

```
## # A tibble: 3 x 10
##   .y.   group1 group2    n1    n2 statistic    df     p p.adj p.adj.signif
## * <chr> <chr>  <chr>  <int> <int>     <dbl> <dbl> <dbl> <dbl> <chr>
## 1 score T1     T2         5     5    -1.06      4 0.347     1 ns
## 2 score T1     T3         5     5    -0.472     4 0.661     1 ns
## 3 score T2     T3         5     5     0.419     4 0.697     1 ns
```

5.2 ANOVA à mesures répétées à deux facteurs de classification

5.2.1 Données

Les mêmes manipulations préalables que pour l'ANOVA à mesures répétées avec un facteur de classification sont à effectuer. La base de données avec laquelle on

travaille contient des scores répétés à trois reprises pour dix individus qui sont répartis dans deux groupes, soit expérimental, soit contrôle.

```
data <- read.csv("diet.csv")

str(data)
```

```
## 'data.frame':    72 obs. of  4 variables:
##  $ id       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ treatment: chr  "ctr" "ctr" "ctr" "ctr" ...
##  $ time     : chr  "t1" "t1" "t1" "t1" ...
##  $ score    : int  83 97 93 92 77 72 92 92 95 92 ...
```

```
data <- data %>%
  mutate(id = as.factor(id),
         treatment = as.factor(treatment),
         time = as.factor(time),
         score = as.numeric(score))

str(data)
```

```
## 'data.frame':    72 obs. of  4 variables:
##  $ id       : Factor w/ 12 levels "1","2","3","4",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ treatment: Factor w/ 2 levels "ctr","Diet": 1 1 1 1 1 1 1 1 1 1 ...
##  $ time     : Factor w/ 3 levels "t1","t2","t3": 1 1 1 1 1 1 1 1 1 1 ...
##  $ score    : num  83 97 93 92 77 72 92 92 95 92 ...
```

```
unique(data$id)
```

```
##  [1] 1  2  3  4  5  6  7  8  9  10 11 12
## Levels: 1 2 3 4 5 6 7 8 9 10 11 12
```

```
unique(data$time)
```

```
## [1] t1 t2 t3
## Levels: t1 t2 t3
```

```
unique(data$treatment)
```

```
## [1] ctr  Diet
## Levels: ctr Diet
```

5.2.2 ANOVA

La présence de deux facteurs de classification change l'argument `within` de la fonction `anova_test()`. En effet, il n'y a plus une mais deux variable intra-sujet. Dans le tableau de l'ANOVA à mesures répétées avec deux facteurs de classification, il y a trois effets:

1. L'effet principal du facteur temps qui teste si les scores changent significa-

tivement entre les trois temps de mesure. L'hypothèse nulle est l'absence de différence entre les temps.
2. L'effet principal du groupe qui test si l'appartenance à un groupe influence significativement le score. L'hypothèse nulle est l'absence de différence entre les groupes.
3. L'interaction entre les deux facteurs intra-sujet qui teste si l'évolution des scores dans le temps diffère selon le groupe d'appartenance. L'hypothèse nulle stipule qu'il n'y a pas d'interaction entre les deux facteurs

```
data %>%
  group_by(time, treatment) %>%
  shapiro_test(score)
```

```
## # A tibble: 6 x 5
##   treatment time  variable statistic      p
##   <fct>     <fct> <chr>        <dbl>  <dbl>
## 1 ctr       t1    score        0.828 0.0200
## 2 Diet      t1    score        0.919 0.279
## 3 ctr       t2    score        0.868 0.0618
## 4 Diet      t2    score        0.923 0.316
## 5 ctr       t3    score        0.887 0.107
## 6 Diet      t3    score        0.886 0.104
```

```
res <-  data %>%
  anova_test(dv = score, wid = id, within = c(time, treatment))

print(res$`Mauchly's Test for Sphericity`)
```

```
##           Effect     W     p p<.05
## 1           time 0.469 0.023     *
## 2 time:treatment 0.616 0.089
```

```
res %>% get_anova_table(correction = "auto")
```

```
## ANOVA Table (type III tests)
##
##           Effect  DFn   DFd      F        p p<.05   ges
## 1           time 1.31 14.37 27.369 5.03e-05     * 0.049
## 2      treatment 1.00 11.00 15.541 2.00e-03     * 0.059
## 3 time:treatment 2.00 22.00 30.424 4.63e-07     * 0.050
```

Dans notre cas, on observe que les deux facteurs sont significatifs, tout comme l'interaction entre eux. Comme l'interactione est significative, il est nécessaire de poursuivre l'analyse en effectuant une ANOVA à mesures répétées à un facteur pour la première variable pour chaque niveau de la deuxième variable, puis de faire des comparaisons multiples si l'ANOVA est significative.

Si l'interaction devait ne pas être significative, il est nécessaire de faire l'analyse des effets principaux directement.