Introduction à R

Thibaut Goetsch & Bineta FAYE

2023-09-07

1. R, Rstudio, Rmarkdown ??

- R est un logiciel libre et gratuit principalement dédié aux analyses statistiques et aux représentations graphiques. Il est distribué par le CRAN (Comprehensive R Archive Network). C'est un site web où l'on peut trouver et télécharger du matériel concernant le logiciel R (code, documentation, bibliothèques...)
- Rstudio est une interface facilitant l'utilisation de R. Elle est également gratuite. L'interface RStudio est divisée en 4 fenêtres :
  - Console où on peut entrer et exécuter des commandes
  - Environnement, History où on peut visualiser les objets construits
  - Files, Plots... où on peut visualiser les répertoires et fichiers de l'espace de travail, les graphiques, intaller des packages...
  - R script où on conserve les lignes de commandes ainsi que les commentaires sur le travail effectué.

Pour travailler sur Rstudio on commence généralement par ouvrir un script puis entrer les commandes dans le script ensuite regarder les sorties dans la console (en cliquant sur le bouton run) sans oublier de sauver le script.

- Rmarkdown est un package qui permet de créer des rapports (au format pdf,word,... o) ou des pages web html qui comportent du code R. On peut également ecrire du latex dedans comme par exemple:

$$\bar{X_n} = \sum_{i=1}^{n} x_i/n$$

2. Installation

1) Installer R

https://pbil.univ-lyon1.fr/CRAN/

2) Installer R studio (interface utilisateur)

https://posit.co/download/rstudio-desktop/

3. Packages

Installation d'un package (1 fois par ordinateur, puis à chaque mise à jour de R ou pour mise à jour du package)

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

Chargement d'un package (avant chaque utilisation des fonctions du package, au début du script) :

```
library(dplyr)
```

En cas de fonction homonyme on peut indiquer le package source grâce aux "::" :

```
dplyr::filter()
```

4. Objets en R

4.1 Variables et affectations

Une variable au sens statistique correspond à une caractéristique pouvant prendre différentes valeurs selon les individus (statistiques, pas forcément une personne mais une unité statistique). Dans notre base de données elle sera donc une série de valeurs. On distingue dans un premier temps les variables :

- Quantitatives : des nombres issus d'une mesure avec lesquelles on peut effectuer des opérations mathématiques (numeric dans R). Exemple : poids, dosage sanguin,
- Qualitatives : du texte issue d'une catégorisation pour lequel on peut compter les individus dans chaque catégorie et calculer des pourcentages (character ou plus souvent factor dans R). Exemple : genre, gravité d'une maladie
- Logiques : issues d'un test et ne pouvant prendre que les valeurs vrai/faux (logical dans R et TRUE/FALSE ou T/F). Correspondent à des variables qualitatives binaires. Exemple : présence d'un symptôme, d'une maladie, appartenance à un groupe

En programmation et donc dans R, une variable peut aussi avoir une valeur unique. On s'en sert alors pour stocker une valeur. Une variable au sens large dans R peut donc s'entendre comme l'association entre un nom et une valeur/série de valeurs. R associe un nom avec un emplacement dans la mémoire et en appelant ce nom il nous renvoie le contenu (ou non) de cet espace. Il faut donc choisir un nom parlant, et qui n'est pas déjà utilisé par R pour ses fonctions. R renverra toujours une erreur si vous appelez une variable qu'il ne connait pas.

On peut créer une variable soit en lui assignant une valeur soit en la déclarant simplement. Elle apparait alors dans l'environnement de travail en haut à droite ainsi que ce qu'elle contient si elle n'est pas vide.

```
# Assigner une valeur
x1<-8
# ou
x2=8
x3<-"Moyenne"
x4<-TRUE
x5<-NULL
```

On peut inspecter le contenu d'une variable de plusieurs manières en l'appelant simplement, ou en demandant d'afficher son contenu plus explicitement

```
x1;x2;x3;x4;x5
print(x3)
```

On peut examiner son type et sa structure grâce à différentes instructions

```
class(x3)
str(x3)
```

Un aperçu de ces éléments est visible dans l'environnement en haut à droite.

Une variable vide prend la valeur NULL et une variable manquante prend la valeur NA (Non applicable) (NaN signifie Not a Number, et Inf infini mais ils se rencontrent moins souvents). NULL n'est pas 0 mais une variable de dimension 0 (0 est bien **un** nombre), et les opérations le contenant donnent un résultat de même nature (numeric(0) character(0) ou logical(0) sont le type de variable suivi de sa longueur). NA aussi renvoie des NA, et beaucoup de fonctions demandent de spécifier le comportement à avoir quant aux données manquantes, qui sont fréquentes en statistiques.

```
1/NA
1/NULL
```

4.2 Vecteurs

Les vecteurs correspondent à des séries de valeurs. On les crée également soit en leur assignant une valeur soit en les déclarant juste.

```
a<-c(1,12,33,75)
a
b<-1:10
b
c<-seq(1,10)
c
d<-seq(1,10,2)
d
e<-rep(1,10)
e
f<-rep(c(0,1),10)
f
g<-rep(c(0,1),each=10)
g
h<-paste0("a",1:10)
h
```

On peut accéder à certaines valeurs de cette série par leur(s) numéro(s) entre crochets, ou bien en retirer par le signe moins.

```
h[1]
h[-1]
h[1:5]
h[seq(2,10,2)]

b[1]<-"a"
b
```

Notez bien qu'un vecteur correspond à une variable et ne contient donc que des valeurs d'un même type.

Avec une série de valeurs, on préfère souvent utiliser des variables de type facteur pour les variables qualitatives dans R. Celles-ci sont des nombres entiers renvoyant chacun au nom d'une catégorie (les niveaux de notre facteur, ou levels). On peut changer l'ordre des niveaux.

```
i<-factor(h)
i
j<-factor(h,levels=h)
j
k<-relevel(j,"a10")
k
l<-factor(k,levels=h)
```

```
l
```

Les variables peuvent être transformées d'un type à l'autre quand cette transformation a du sens pour R.

```
as.numeric(i)
as.factor(f)
as.character(j)
as.logical(f)
as.numeric(c(TRUE,FALSE))
```

On peut citer les variables de type date à titre d'information, qui se présentent comme du texte structuré. Il faut indiquer à R comment l'interpréter(day/month/year souvent) pour qu'il situe la date (peut s'étendre à l'heure, la minute, la seconde). Celles-ci permettent de localiser un évènement dans le temps ou de calculer des durées.

```
var.date<-as.Date("30/05/1997",format="%d/%m/%Y")
var.date
var.date2<-as.Date("08/06/1997",format="%d/%m/%Y")
var.date2

difftime(var.date2,var.date,units="days")
```

4.3 Listes

Les listes ne sont pas des variables contrairement aux vecteurs. Elles ne sont pas des séries de valeurs mais des séries d'objets (des variables, ou d'autres listes). Ces objets peuvent être de n'importe quelle nature indépendamment de leurs voisins.

```
a<-list(a,b,c,d,e,f,g,h)
a
b<-list(1:10)
b
c<-as.list(1:10)
c
d<-list(list(1,2),3,list(list(4,5)))
d
```

On appelle un élément d'une liste grâce aux doubles crochets

```
a[[1]]
```

4.4 Matrices

Les matrices sont des tableaux de valeurs qui ont la même propriété que les vecteurs de ne pouvoir contenir qu'un seul type de valeurs, le plus souvent numérique.

```
a<-matrix(1:9,nrow=3,ncol=3)
a
b<-matrix(1:9,nrow=3,ncol=3,byrow = TRUE)
b
c<-matrix(1:3,nrow=3,ncol=3)
c
d<-cbind(1:3,1:3,1:3)
d
e<-rbind(1:3,1:3,1:3)
e
```

On accède à un ou plusieurs éléments d'une matrice entre crochets en donnant leurs coordonnées séparées par une virgule [ligne,colonne]

```
a[1,2]
```

4.5 Data frame

Les bases de données sont des listes de variables qui portent toutes sur les mêmes individus, donc de même taille, celle de notre échantillon. Si nous n'avons pas le même nombre de valeurs pour chaque variable, c'est que certaines elles sont manquantes. Certaines bases comme iris sont accessibles :

```
str(iris)
summary(iris)
head(iris)
View(iris)
```

Les variables d'une base de données peuvent être appelées de différentes manières. Notez que ça ne donne pas toujours le même résultat.

```
iris$Sepal.Length
iris$Sepal.Length[1]
iris[[1]]
iris["Sepal.Length"]
str(iris["Sepal.Length"])
iris[["Sepal.Length"]]
str(iris[["Sepal.Length"]])
iris[1,1]
iris[1,"Sepal.Length"]
```

Grâce aux indices on peut croiser les informations plus simplement avec une base de données, puisqu'il n'y a toujours qu'un seul ordre à nos séries de valeurs pour toutes les variables

```
iris$Sepal.Length[iris$Sepal.Width>2]
```

Une base de données est le plus souvent chargée d'un tableau de donnée mais peut aussi être construite manuellement

```
BDD<-data.frame(Var1=1:10,Var2=rep("a",10),Var3=factor(rep(c("a","b"),10)))
str(BDD)
BDD2<-data.frame(Var1=1:10,Var2=rep("a",10),Var3=factor(rep(c("a","b"),length.out=10)))
str(BDD2)
as.data.frame(a)
```

Tous les objets peuvent être nommés (chaque élément d'un vecteur par exemple) mais pour une base de données c'est automatique, les colonnes sont nommées par défaut et les lignes numérotées.

```
colnames(iris)
rownames(iris)
```

5. Tests logiques, Conditions & Boucles

- Tests logiques

On teste ici si deux éléments sont égaux, différents,...

```
a<-"blue"
b<-"green"
a==b
a!=b
```

```
#################
x<-8
y<-2
x>y
x<y
#################
x>y & a!=b  #and
x<y | a==b  #or
```

- Conditions

if .... **else** On les utilise pour éxécuter un bout de code selon une condition.

- Syntaxe

```
if(condition){
    instrs...
}else{
    instrs...
}
```

- Exemple

```
if (a!=b){
    cat(a,"different de ",b)
    } else
    print("égalité :) ")
```

- Boucles

Comme c'est le cas pour tous les langages de programmation, les boucles **for** et **while** sont les plus utilisés.

Syntaxe

```
for(item in vect){
    instr1
    instr2...
}
```

Exemple

```
for(i in 1:5) {print(1:i)}
```

Syntaxe

```
while(cond){
    instr...
}
```

Exemple

```
i=1
while(i<=5)
{print(1:i)
    i=i+1}
```

Application: Ecrire un programme que renvoie tous les entiers pairs entre 1 et 1000.

Ecrire programme qui trouve tous les nombres premiers de 2 à 10 000 (facultatif)

6. Fonctions

Une fonction est un sous-programme, c'est-à-dire une portion de code qui est exécutée lorsqu'on l'appelle.

- Syntaxe:

```
nom_de_ma_fonction <- function(arguments) {
    instrs....
}
```

*Exemple

```
cube<- function(x){
    cub<-x^3
    return(cub)
}
cube(4)
```

Application:

Ecrire une fonction qui prend en entrée un entier n et qui retoure la liste des entiers pairs de 1 à n.

Ecrire une fonction qui trouve tous les nombres premiers de 2 à n (facultatif)

7. Data

Les données sont souvent le matériel de départ d'une étude statistique. Elles sont généralement d'abord stockées dans des fichiers (txt, xls, csv).

Une des premières étapes du travail est alors d'amener ces données dans R et de les affecter à un objet de type dataframe.

Il est par conséquent très important connaître et maîtriser les opérations qui permettent de réaliser ces importations.

7.1 Import, Manipulation, Export

- Import

Depuis un fichier externe

Les fonctions read.table et read.csv sont les fonctions standards de R pour importer des données à partir de fichiers .txt ou .csv.

```
read.table("chemin/nom_fichier.txt",sep=",",header = TRUE)
read.csv("chemin/nom_fichier.csv",sep=",",header = TRUE)
```

Exercice: chargez le jeu de données test.csv,mettez le dans df et faites head(df)

Depuis un package R

Dans certains packages de R sont aussi stockées des jeux de données. Pour les importer, il suffit d'attacher le pachage pour accéder au jeu de données.

Par exemple: On peut accéder au jeu de données TitanicSurvival du package carData contenant l'information sur l'état de survie, le sexe, l'âge et la classe de 1 309 passagers lors de la catastrophe du Titanic en 1912.

```
library(carData)
TitanicSurvival
ttn<-TitanicSurvival
```

- Manipulation

Les commandes basiques

Manipulez!!!

Executez toutes les commandes metez en commentaires ce qu'elles font

```
dim(ttn)
ttn
class(ttn)
```

```
class(ttn$age)
levels(ttn$sex)
summary(ttn$age)
```

```
ttn$age #vecteur age
ttn[,3]
ttn["age"]#
```

```
ttn[ttn$sex=="female",]
```

```
ttn[ttn$sex=="female" & ttn$survived=="no",]
```

```
ttn[ttn$passengerClass=="1st"| ttn$passengerClass=="2nd",2:3]
```

```
ttn[ttn$age>25,1:3]
```

stats

```
mean(ttn$age)
mean(ttn$age[!is.na(ttn$age)])
median(ttn$age[!is.na(ttn$age)])
quantile(ttn$age[!is.na(ttn$age)])
summary(ttn$age)
```

tidyverse

Tidyverse est une collection de packages R conçus pour la science des données.

On peut citer entre autre:

tidyr qui aide à créer des données ordonnées, où chaque colonne est une variable, chaque ligne est une observation et chaque cellule contient une valeur unique.

ggplot2 qui est lui spécialisé sur la création de jolies graphiques

dplyr qui fournit un ensemble de "verbes" qui aident à résoudre les défis de manipulation de données les plus courants :

- select() : sélectionner des colonnes (variables)
- filter() : filtrer des lignes (individus)
- arrange() : ordonner des lignes
- mutate() : créer des nouvelles colonnes (nouvelles variables)
- summarise() : calculer des résumés numériques (ou résumés statistiques)
- group_by() : effectuer des opérations pour des groupes d'individus

Ici nous allons nous focaliser principalement sur **dplyr**. Vous pouvez bien evidemment exporer les autres package de tidyverse ici https://www.tidyverse.org/. Pour l'intallation il suffit d'installer la collection d'un seul coup:

```
install.packages("tidyverse") ##
```

Vous pouvez consulter le cheatsheet de dplyr ici https://github.com/rstudio/cheatsheets/blob/main/data-transformation.pdf

```
head(ttn)
```

Exécutez et dites ce que font les commandes suivantes:

```
ttn|>select(survived,sex)
```

```
ttn|> filter(sex=="female")
```

```
ttn|> filter(survived=="yes")
```

```
ttn|> filter(sex=="female" & survived=="yes")|> summarize(n=n())
```

```
ttn|> filter(sex=="male" & survived=="yes")|> summarize(n=n())
```

```
ttn|>arrange(sex)
```

```
ttn|>arrange(survived)
```

```
ttn|>arrange(age)
```

```
ttn|>arrange(desc(age))
```

```
ttn|>mutate(age_mois=age*12)
```

```
ttn|> group_by(sex)|> summarize(n=n())
```

Donner le nombre de survivant en fonction du genre en utilisant les "verbes" précédents

```
ttn|>filter(!is.na(age))|>
    summarize(
            mean(age))
```

Que renvoie la commande

```
ttn|>
    group_by(survived)|>
    summarize(
            mean(age))
```

Et maintenant?

```
ttn|>filter(!is.na(age))|>
    group_by(survived)|>
    summarize(
            mean(age))
```

- Export

I s'agira ici d'enregistrer sur sa machine, un dataframe qu'on a sous R.

```
write.table(ttn,"titanic.txt")
write.csv(ttn,"titanic.csv")
```

7.2 Visualisations

- plot()
- hist()
- barplot()

- points() ...

Tapez **help(plot)**

```
x<-c(1:30,40:60)
y<-x-5
```

```
plot(x,pch=18,col="blue")
points(y,col="red")
```

```
plot(x,y)
```

```
hist(x)
```

Ecrire et représenter la fonction $f(x) = \frac{x^2}{ln(10)}$ sur [-50,50]

- Exercice

On reprend le jeu de données Titanic...

```
plot(ttn)
```

Tapez la commande suivante et interpretez

```
plot(ttn$age,pch=18)
```

Rpresentez l'histogramme de la variable age

```
plot(ttn$sex)
```

Age des femmes en rouge et celui des hommes en bleu Creer d'abord un vecteur **couleur** que prend "red" si sex="female" et "blue" sinon.

Mettez côte à côte les histogrammes de l'age pour les hommes et pour les femmes.