Introduction to R for your data analysis and visualizations

Contents

Préambule

Qu'est-ce que `R` ?

- `R` est un logiciel permettant de faire des statistiques et produire des graphiques ;
- `R` est également un langage de programmation, c'est ce qui fait sa spécificité ;
- `R` est gratuit, disponible sur toutes les plateformes (Windows, Mac, Linux...) ;
- `R` est un logiciel open-source, chacun peut collaborer et enrichir le logiciel avec des nouvelles librairies.

Nous vous invitons à installer `R` sur vos ordinateurs personnels. Vous trouverez çi-dessous des liens utiles :

- un lien pour télécharger `R` : `http://cran.univ-lyon1.fr/`;
- un lien pour télécharger `RStudio` (environnement plus pratique, mais il faut avoir installé R avant) : `http://www.rstudio.com/products/rstudio/download/`;
- un lien vers le projet `R` (en anglais) : `http://www.r-project.org/`.

Note : `R` est basé sur un interpréteur `Scheme`.

1 Découverte de R et RStudio

1.1 Description de l'interface classique

Lancez `RStudio`. La fenêtre qui s'ouvre se décompose en 4 fenêtres :

1. En haut à gauche, la fenêtre de `script`, dans laquelle on écrit les programmes (fichiers avec une extension .R), commentaires (après le symbole `#`)... Lorsqu'on souhaite exécuter une ligne de commandes, ou un bloc de lignes de commandes, on sélectionne les inscructions à éxécuter et on les envoie vers la console avec la combinaison de touches `Ctrl + Entrer`.

2. En bas à droite, la fenêtre de `la console`. Dans la `console`, on tape les commandes à la suite du symbole `>`, elles sont exécutées après avoir tapé `Entrer`. Dans la `la console`, le symbole `>` signifie que `R` est prêt à travailler. Si ce symbole n'est pas affiché, c'est que `R` n'a pas fini de calculer le résultat de la commande précédente.

3. En haut à droite, une visualisation de l'`environnement` crée : variables définies, tables de données chargées etc...

4. En bas à droite, une fenêtre avec différents onglets pour visualiser l'aide, les graphiques etc...

1.2 Description de l'interface mode Notebook

1. A gauche, vous avez accès à une grande fenêtre dans laquelle vous pouvez écrire :

   - du texte, en mode markdown. Profitez du mode texte pour indiquer les titres, expliquer votre design expérimental et vos choix méthodologiques, expliquer vos conclusions après chaque étape, etc...
   - des morceaux de code R. Vous pouvez donc découper pas à pas vos scripts et calculs.
   - les résultats des bouts de code.

2. En haut à droite, une visualisation de l'`environnement` crée : variables définies, tables de données chargées etc...

3. En bas à droite, une fenêtre avec différents onglets pour visualiser l'aide, les graphiques etc...

Il y a deux avantages à utiliser le mode Notebook. Le premier avantage est l'imbrication texte-code-résultats, qui vous permet d'expliquer votre analyse et vos choix méthodologiques au fur et à mesure. C'est le vecteur idéal pour rendre compte d'une analyse avec vos collègues. L'autre avantage est que votre notebook peut être exporté vers différents formats : page html, document pdf, ou (on ne sait jamais) document Word.

A titre d'exemple, la correction du sujet de cette semaine vous sera fournie en mode notebook, et à partir de la semaine prochaine **tous vos travaux de TP devront être faits en mode Notebook**.

1.3 Répertoire de travail

`R` utilise un répertoire de travail à partir duquel il va lire les fichiers de données, sauvegarder les scripts, les environnements ou les figures. Pour connaitre votre répertoire de travail, utilisez la commande `getwd()`. Pour modifier votre répertoire de travail, utilisez la commande `setwd("path")`. Si vous êtes sous `RStudio`, vous pouvez réaliser cette opération *via* la menu `Session > Set Working Directory > Choose Directory`.

1.4 Sauvegarder son travail

Lorsqu'on travaille sous `R`, on a la possibilité de sauvergarder :

- Le script avec les commandes, commentaires etc : c'est la manière la plus simple de travailler car en sauvegardant le script et les données initiales, on peut toujours reproduire des résultats (si on n'a pas utilisé de méthodes non-déterministes ou aléatoires, bien sûr). L'extension des scripts `R` est simplement `.R`.
- Le script avec les commandes + tout l'envirronnement de travail. Cette solution présente un gros inconvénient : c'est plus volumineux, mais en contrepartie est très utile si les calculs sont très longs car on peut directement stocker le résultat sans avoir à ré-éxécuter ces calculs. L'extension des environnements `R` est `.Rdata`.

Vous êtes supposés sauvegarder régulièrement votre travail sans qu'on vous le rappelle. Merci !

1.5 Versionner son travail R

Si `git` est installé sur votre poste, vous pouvez versionner vos analyses R ! Pour cela, allez fouiller dans les menus `Create a project > Version Control > Git > ...`. Vous verrez apparaître une icône `Git` dans le bandeau supérieur qui vous permettra de faire vos `commit`, `push` etc...

2 Uploading a dataset

2.1 The read.table() function

This functions allows to import a dataset contained in a text file or a spreadsheet. The columns are the variable studied, and the lines are the individuals of the population (exactly one individual per line). The supported format is very simple : columns are separated with a comma, a space or a tabulation (or anything you want if you specify it with the `sep` argument).The resulting object in R is called a `data.frame`.

```
# reading the table ...
> df <- read.table("data-individuals.txt", header=TRUE, sep="\t")

# looking at the table that has been read (to detect potential errors)
# with at least one of these three commands
> head(df)
> str(df)
> summary(df)
```

Comment #1 If the command does not work, in 99% of the cases it is because the filename is not correct ; or the `working directory` is not correct.
What is the `working directory` ? This is the directory where `R` reads and writes files. To indicate your `working directory`, use the menu `Session > set working directory > choose directory`, or the command `setwd()`.

Comment #2 When you have loaded a dataset, always verify the import is correct. To do that, you can use one of the command `head`, `str` or `summary`.

2.2 Exploring the table

To obtain basic information about the table, you can use the functions :

```
nrow(df)    # number of rows in the dataset
ncol(df)    # number of columns
colnames(df)     #  the names of the columns
dim(df)          # dimension of the table
summary(df)      # a nice summary of the dataset
```

The `summary()` command is very convinient and gives a nice summary of the global dataset. More precisely, for each column of the dataset :

- if the variable is qualitative, it indicates the levels and the ditribution in each category.
- if the variable is quantitative, it indicates the mean, the median, the fisrt and third quartiles, the max and the min.

3 Base-R

3.1 Columns selections

To select one or several column(s), two solutions are possible:

1. The first is to use the name of the column (if there are headers of course)
2. The second method is to give the number of the column (be careful, number starts at 1).

```
#     1.  selecting a col with its name
#         the output is a vector
df$Weight
df$Major

#     2.  selecting a col with its indices (indices start at 1)
#         the output is a vector
df[,3]

#     3.  selecting multiple col with their names
#         the output is a data.frame
df[, c("Weight","Major")]

#     4.  selecting multiple col with their indices (indices start at 1)
#         the output is a data.frame
df[, c(3,5)]
```

3.2 Line selections

To select one or several line(s), two solutions are possible:

1. The first method is to give the number(s) of the line(s) (be careful, number starts at 1).
2. The second one is to indicate a condition (useful to sub-select data)

```
#     1.  selecting a line with its indices (indices start at 1)
#         the output is a list
df[8,]

#     2.  selecting multiple lines with their indices (indices start at 1)
#         the output is a data.frame
df[c(1,3,5,7),]
df[90:100,]

#     3.  selecting one or multiple lines given a boolean condition
#         the output is a data.frame
df[which(df$Gender == "Mrs"),]
df[which(df$Gender == "Mrs" & df$Height > 170),]
df[which(df$Major %in% c("Eco","Sciences")),]
```

The last instruction (selection by condition) is the basis of everything in base-R. In some programs, some programmers simplify the command by writing it a little bit differently :

```
# instead of writing :
df[which(df$Weight < 52),]
# they write :
df[df$Weight < 52,]
```

Even if the results are equivalent with our dataset, the two commands are not equivalent. Actually, when the dataset is containing missing data (`NA`), these elements will not be selected with the `which` instruction, while they will be without. So be careful if your dataset might contain missing values !

3.3 Functions for one or two vector(s)

A small list here some useful commands :

```
# functions for one vector of quantitative data
mean(df$Height, na.rm=TRUE)
sd(df$Height, na.rm=TRUE)     # the standard deviation
median(df$Height, na.rm=TRUE)
quantile(df$Height, na.rm=TRUE)
quantile(df$Height, na.rm=TRUE, probs=seq(0,1,by=0.1))

# functions for two vectors of quantitative data
cor(df$Height, df$Weight, method="pearson")
cor(df$Height, df$Weight, method="spearman")

# functions for one vector of qualitative data
table(df$Major)

# functions for two vectors of qualitative data
table(df$Major, df$Glasses)

# to count...
length(which(data$Height > 180))
```

3.4 Exercises

1. Upload the dataset "data-individuals-large.txt". Be careful with the `read.table()` options...
2. Calculate the average weight, and the standard deviation of weights.
3. What is the average weight of men wearing glasses?
4. Calculate the body mass index (bmi), for all the individuals in the table (the formula is given below), and add it as a new column in the data.frame.

   $$bmi = weight/height^2$$

   (the size is in m, the weight in kgs).
5. How many persons have a body mass index under 19 ?

4 Tidyverse R

4.1 What is the tidyverse ?

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures. These packages are :

- `tidyr`: The goal of `tidyr` is to help you create tidy data. Tidy data is data where :
  1. Every column is variable.
  2. Every row is an observation.
  3. Every cell is a single value.
- `dplyr`: a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges
- `ggplot2`: system for declaratively creating graphics, based on The Grammar of Graphics.
- `tibble`: modern reimagining of the data.frame, keeping what time has proven to be effective, and throwing out what is not. Tibbles are data.frames that are lazy and surly: they do less (i.e. they don't change variable names or types, and don't do partial matching) and complain more (e.g. when a variable does not exist).
- `purrr` : enhances R's functional programming toolkit by providing a complete and consistent set of tools for working with functions and vectors.

4.2 Line(s) selection with the dplyr verb `filter`

```
# a simple selection of lines
df %>%
  filter(Major == "Art/Letters")
```

Comment #3 This is the first appearance of the `%>%` operator. This is a pipe (meaning what is on the left side of the `%>%` operator is taken as an input for what is on the right side of the `%>%`. We use it a lot in the tidyverse, and the classical way to write pipelines is to write the `%>%` operator at the end of the line (and the right side of the pipeline on the next line).

```
# if you have several criteria
df %>%
  filter(Major == "Art/Letters") %>%
  filter(Height >= median(Height))

# is not equivalent to
df %>%
  filter(Major == "Art/Letters",
         Height >= median(Height))
```

4.3 Columns(s) selection with the dplyr verb `select`

```
# a simple selection of columns
df %>%
  select("Gender","Glasses","Weight")
```

4.4 Exercises (selection verbs)

1. Upload the dataset "data-individuals-large.txt". Be careful with the `read.table()` options...
2. Create a **new** data.frame containing only the men wearing glasses, and keeeping information only about gender, weight and glasses.
3. What is the average weight of men wearing glasses?

4.5 Adding new columns or modifying columns with the dplyr verb `mutate`

The same verb `mutate` is used for both modifying a column, or to add a new one. If you use it with an already existing column name, it's a modification, while if you use it with a new column name, it will create it. The verb `mutate` can be seen as a mapping.

```
# Modyfying a column (1)
# example : transforming the size from cms to meters
df <- read.table("data-individuals.txt", header=TRUE, sep="\t")
head(df)
df <- df %>%
  mutate(Height = Height/100)
head(df)

# Modyfying a column (2) with a conditional expression
# example : "yes"/"no" for the seeblings instead of its size
head(df)
df <- df %>%
  mutate(Brothers.Sisters = ifelse(Brothers.Sisters>0, "Yes","No"))
head(df)
```

```
# Adding a new column in the dataset
# example : adding the body mass index
df <- read.table("data-individuals.txt", header=TRUE, sep="\t")
head(df)
df <- df %>%
  mutate(Height = Height/100) %>%
  mutate(bmi = Weight / Height**2)
head(df)
```

4.6 Renaming a column with the dplyr verb `rename`

Renaming columns is very useful (especially before joining tables). To rename a column, a simple verb `rename` exists (although we could have used `mutate` and pipe it with a `select` to obtain a column renaming).

```
# Renaming a column
# example : indicating the unit for heights and weights
df <- read.table("data-individuals.txt", header=TRUE, sep="\t")
head(df)
df <- df %>%
```

```
  mutate(Height = Height/100) %>%
  rename(height.cms = Height,
         weight.kgs = Weight) %>%
  mutate(bmi = weight.kgs / height.cms**2)
head(df)
```

4.7 Sorting the dataset with the dplyr verb arrange

```
# Sorting by increasing order
# example : the bmi are sorted
head(df)
df <- df %>%
  arrange(bmi)
head(df)

# Sorting by decreasing order
# example : the bmi are sorted
head(df)
df <- df %>%
  arrange(desc(bmi))
head(df)

# Sorting by increasing order with several columns
# example : the bmi are sorted
head(df)
df <- df %>%
  arrange(Major, bmi)
head(df)
```

4.8 Exercises (window verbs)

1. Upload the dataset "data-individuals-large.txt". Be careful with the `read.table()` options...

2. Create a new data.frame containing only the men wearing glasses, and indicating their height, weight, body mass index and another column indicating if the student should be on diet or not. The diet is required iff the bmi is under 18 (not enough) or above 25 (too much).

4.9 Summarizing values with the dplyr verb summarise (or summarize)

The verb `summarise` (or `summarize`, both spellings are OK) is used when you want an entire column to be summarised into a single value.

```
head(df)
df %>%
  summarise(
    ave.size = mean(height.cms, na.rm=TRUE),
    sd.size = sd(height.cms, na.rm=TRUE)
  )
```

It is very often used with the verb `group_by` that creates subgroups.

```
head(df)
df %>%
  group_by(Gender, Major) %>%
  summarise(
    ave.size = mean(height.cms, na.rm=TRUE),
    sd.size = sd(height.cms, na.rm=TRUE)
  )
```

4.10 Exercises (summarising verbs)

1. Upload the dataset "data-individuals-large.txt". Be careful with the `read.table()` options...
2. Create a new data.frame containing, for all the persons having aswered the gender question, the median weight and the number of art students, seperately for both men and women.

4.11 Joining tables with the dplyr verb `left_join`

Let's consider we have two tables that were found in `https://www.gapminder.org/data/`. They contain the children mortality rate (death of children between 0 and 5 y.o. per 1000 born), and the fertility rates (children per woman), for 193 countries.
As the datasets are not tidy, we will use the verb `pivot_longer` from tidyr.

```
df_child_mort <- read.table("child_mortality.csv", header=TRUE, sep=",",
                                                   check.names = FALSE)
head(df_child_mort)
# ooch, this is not tidy !

df_child_mort <- df_child_mort %>%
  pivot_longer(
    c(-"country"),
    names_to = "year",
    values_to = "child.mort.rate")
head(df_child_mort)

df_fertility <-  read.table("children_per_woman.csv", header=TRUE, sep=",",
                                                   check.names = FALSE)
head(df_fertility)
# ooch, this is not tidy !

df_fertility <- df_fertility %>%
  pivot_longer(
    c(-"country"),
    names_to = "year",
    values_to = "children.per.woman")
head(df_fertility)
```

Now to join these two tables, we have to be careful to the country name and the year...

```
df_joined <- df_child_mort %>%
  left_join(df_fertility, by=c("country","year"))
head(df_joined)
```

Comment #4 We have used a left join, but other joining verbs exist (right join, inner join, full join).

4.12 Exercises (global)

1. Upload the datasets "children per women", "child mortality" and "income" in 3 datasets.
2. Create a new data.frame joining everything.
3. Add a new column "Century" that indicates the century (19,20 or 21).
4. For each country, compute the average children per woman and the average income depending on the century.

5 Introduction to ggplot2

5.1 Introduction

An effective chart is one that:

1. Conveys the right information without distorting facts.
2. Is simple but elegant. It should not force you to think much in order to get it.
3. Aesthetics supports information rather that overshadow it.
4. Is not overloaded with information.

The list below sorts the visualizations based on its primary purpose. First, please load the ggplot2 library :

```
library(ggplot2)
```

5.2 Correlation with scatterplots

Loading the dataset :

```
# Dataset
data("midwest", package = "ggplot2")
colnames(midwest)
```

A simple scatterplot with the verb `geom_point`...

```
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_point()
plot(gg)
```

You can modify axis delimitation with the modifyier `xlim` and `ylim` :

```
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_point() +
  xlim(c(0,0.03)) +
  ylim(c(0,5e5))
plot(gg)
```

You can change titles and labels with the modifyier `labs` :

```
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_point() +
  labs(subtitle="Area Vs Population",
       y="Population",
       x="Area",
       title="Scatterplot",
       caption = "Source: midwest")
plot(gg)
```

You can "shake" points (useful when they overlap) by replacing the verb `geom_point` with the verb `jitter` :

```
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_jitter() +
  xlim(c(0,0.03)) +
  ylim(c(0,5e5))
plot(gg)
```

You can color points based on another qualitative variable by passing this instruction to the verb. For example :

```
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_jitter(aes(col=state)) +
  xlim(c(0,0.03)) +
  ylim(c(0,5e5))
plot(gg)
```

The same with a color depending on a quantitative data (so the color represents a gradient) :

```
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_jitter(aes(col=percollege)) +
  xlim(c(0,0.03)) +
  ylim(c(0,5e5))
plot(gg)
```

Of course, color scales can be modified, for example :

```
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_jitter(aes(col=percollege)) +
  scale_color_gradient(low="blue", high="red") +
  xlim(c(0,0.03)) +
  ylim(c(0,5e5))
plot(gg)
```

You can modify the dot size by passing this instruction to the verb. For example :

```
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_jitter(aes(col=percollege, size=popdensity)) +
  scale_color_gradient(low="blue", high="red") +
  xlim(c(0,0.03)) +
  ylim(c(0,5e5))
plot(gg)
```

5.2.1 Exercise 1...

With the dataset "data-individuals.txt", propose a scatterplot giving the weight depending on the height.

If you have finished, add color depending on the gender and dot size depending on the bmi.

5.2.2 Adding a regression...

You can easily add a regression to your scatterplot by adding a second verb `geom_smooth` :

```
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_point(aes(col=state, size=popdensity)) +
  geom_smooth(method="loess", se=T) +
  xlim(c(0, 0.1)) +
  ylim(c(0, 500000)) +
  labs(subtitle="Area Vs Population",
       y="Population",
       x="Area",
       title="Scatterplot",
       caption = "Source: midwest")
plot(gg)
```

5.2.3 Exercise 2

With the dataset "data-individuals-large.txt", propose a scatterplot giving the weight depending on the height. If you have finished, add color depending on the gender and dot size depending on the imc.
Add a regression line with standard error. You can try to change regression to "lm" : what happened ?

5.3 Simple histogram

The verb changes to `geom_hist`.

```
# Histogram on a Continuous (Numeric) Variable
g <- ggplot(midwest, aes(poptotal)) +
   geom_histogram( binwidth = 100000,
                   col="white",
                   size=.1) +
  xlim(c(0,2e6)) +
  ylim(c(0,30)) +
  labs(title="Histogram with Fixed Binning")
plot(g)
```

5.3.1 Enrichment with colors depending on another variable

```
# Histogram on a Continuous (Numeric) Variable
g <- ggplot(midwest, aes(poptotal)) +
  scale_fill_brewer(palette = "Spectral") +
  geom_histogram(aes(fill=state),
                   binwidth = 100000,
                   col="white",
                   size=.1) +
  xlim(c(0,2e6)) +
  ylim(c(0,30)) +
  labs(title="Histogram with Fixed Binning")
plot(g)
```

5.3.2 Exercise 3

With the dataset "data-individuals-large.txt", propose a histogram describing the imc distribution. If you have finished, add titles and color depending on the gender. Don't hesitate to play with the bin width.

5.3.3 Histogram for a categorical variable

The verb changes to `geom_bar` (and **not** `geom_hist`).

```
df <- read.table("data-individuals.txt", header=TRUE, sep="\t")
# Histogram on a Categorical variable
g <- ggplot(df, aes(x=Major)) +
  geom_bar() +
  labs(title="Histogram on Categorical Variable",
       subtitle="Gender repartition depending on Major") +
  coord_flip()
plot(g)
```

5.3.4 Enrichment with colors depending on another variable

```
df <- read.table("data-individuals.txt", header=TRUE, sep="\t")
# Histogram on a Categorical variable
g <- ggplot(df, aes(x=Major)) +
  geom_bar(aes(fill=Gender), width = 0.5) +
  labs(title="Histogram on Categorical Variable",
       subtitle="Gender repartition depending on Major") +
  coord_flip()
plot(g)
```

5.3.5 Exercise 4...

With the dataset "data-individuals-large.txt", propose a barplot describing the glasses repartition depending on the major.

If you have finished, add titles

5.4 Simple boxplot

The verb changes to `geom_boxplot`.

```
g <- ggplot(midwest, aes(y=percasian, x=state)) +
  geom_boxplot() +
  labs(title="Boxplot",
       subtitle="Percentage of asian people per state")
plot(g)
```

5.4.1 Exercise 5...

With the dataset "data-individuals.txt", propose a boxplot the size as a function of the major. If you have finished : add titles, flip your boxplot and change colors (one color per major).

5.5 Violin plot (variant of boxplot)

A violin plot is a kind of boxplot showing a more precise distribution of the data (carfeul, a local fit for normalization ios included). The verb changes to `geom_violin`.

```
g <- ggplot(midwest, aes(y=percollege, x=state)) +
  geom_violin(aes(col=state)) +
  scale_fill_brewer(palette = "Spectral") +
  labs(title="Boxplot",
       subtitle="Percentage of asian people per state")
plot(g)
```

5.5.1 Exercise 6...

With the dataset "data-individuals-large.txt", choose one qualitative variable and one quantitative variable and propose an appropriate violin plot...

5.6 Facetting...

Facetting may be used with all geom types. For example with the graph:

```
gg <- ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_jitter(aes(col=percollege)) +
  xlim(c(0,0.03)) +
  ylim(c(0,5e5))
```

```
plot(gg) + facet_grid(~state)
```

or :

```
plot(gg) + facet_grid(state~.)
```