Good practices in R

This page is based on a document created by Aurélien Nicosia (ULaval) in 2023 called "Bonnes pratiques de programmation en R." It has been updated.

Since R is "just" a programming language, it is technically possible to use a simple text editor to write R code and a terminal to run it. However, it is much more convenient—especially for beginners—to use a code editor such as RStudio to run your code. This allows you to run your code interactively, view your working environment, test your code, and more.

In addition, it is best to make sure you are working with the latest version of R and the packages you need. This reduces the risk of encountering bugs and allows you to take advantage of the latest features. One point to note, however, is that I would advise against changing the version of R or the packages once you have started working on something. This change could introduce bugs in the middle of the analysis (so always report the versions of the packages used for reproducibility). You can get a summary of the working environment with the `systemInfo()` and `packageVersion(pkg)` functions.

Regarding documentation, the `roxygen` package allows you to convert comments in your functions and packages into clean documentation.

A uniform syntax makes code much easier to read and understand. A style guide sets standards for uniform syntax. The *tidyverse* style guide (https://style.tidyverse.org) is recommended in R. Before presenting some style conventions, note that you can quickly change the formatting of a piece of R code in RStudio in the "Code -> Reformat Selection" menu.

Line breaks and indentation

A simple way to make your code more readable is to insert appropriate line breaks and indentations. For example, suppose we have the following character string:

```
text <- "Ceci est un example"
```

We want to correct two errors in this sentence: the word "example" written in English rather than French and the missing period at the end of the sentence. This can be done with the following instruction:

```r
paste0(gsub(pattern = "example", replacement = "example", x = text), ".")
```

This instruction contains one function call nested within another. It is much easier to read as follows:

```r
paste0(
  gsub(
    pattern = "example",
    replacement = "exemple",
    x = text),
  ".")
```

Assignment operator

In R, `<-` is used to assign a value to a variable, and `=` is used to pass values to arguments in a function call.

Naming conventions

The *tydiverse* style guide recommends using lowercase letters, numbers, and the underscore `_` to name variables and functions. Underscores are used to separate words in a name. Although other conventions may be found, these should be avoided. In any case, it is important to choose a convention and stick to it. In addition, it is best to avoid accents in variable names.

Code organization

When the code starts to get long, it becomes advantageous to separate it into several files. For example, you can have one file per part of the analysis (one for data cleaning, one for analysis, one for visualization, etc.). In addition, a data analysis does not usually consist solely of R code, e.g., C++ code files, data files, configuration files, etc. It is therefore recommended to create subfolders grouping files of the same type. RStudio projects are ideal for gathering all project-related files in one place. In addition, they make it easier to work on several projects simultaneously by managing the transition from one working directory to another.

A few things to do

1. Write your code in a script and save it frequently. This prevents you from losing track of important instructions because they were written directly in the console.
2. It is best to start every R work session with an empty work environment. To do this, disable automatic session image restoration in the settings. This allows you to be aware of the presence of different objects in the working environment.
3. Do not use the `load` function when the working environment is not empty. This prevents you from modifying an object in the working environment by overwriting it.

4. Do not use the `attach` function. This prevents you from modifying the file search path.
5. Save the graphics options and settings before modifying them.
6. Do not use `T` and `F` instead of `TRUE` and `FALSE`.