EpiTeam R Style Guide

Epiconcept

2023

Contents

1 Introduction

The Epidemiology team aims to harmonise its coding style across projects.

This document describes the style that we aim to apply. It is based on https://style.tidyverse.org/index.html, initially derived from Google's original R style https://google.github.io/styleguide/Rguide.html.

The most important thing about a style guide is that it provides consistency. Making code easier to read will also allow to focus on the content.

Some decisions described in this guide are driven by the wish to keep our R code as simple as possible. Non-R users should be able to read and understand our R code.

If you have any suggestion to this guide, please feel free to share them to the team at strap@epiconcept.fr

2 Syntax

2.1 Assignment

Proper assignment `<-` is preferred over the equal sign. Never use right assignment:

```
# Good
results <- mean(iris$Sepal.Length)

# Bad
results = mean(iris$Sepal.Length)
mean(iris$Sepal.Length) -> results
```

Variable name and assignment should be on the same line:

```
# Good
iris_sepal <- iris %>%
  dplyr::select(Species, Sepal.Length, Sepal.Width) %>%
  dplyr::arrange(-Sepal.Length)

# Bad
iris_sepal <-
  iris %>%
  dplyr::select(Species, Sepal.Length, Sepal.Width) %>%
  dplyr::arrange(-Sepal.Length)
```

2.2 Naming

2.2.1 Variables

Use `CamelCase` for variable names (similarly to ECDC data warehouse format).

```
# Good
DayOne

# Bad
first_day_of_the_month
djm1
```

2.2.2 Globals

Global variables and constants should be in `UPPERCASE` with `snake_case`:

```
# Good
COUNTRY_CODE
VACCINE_BRAND
```

2.2.3 Functions

Use a special `camelCase` for function names (first letter in lower case). Start your function name with a verb as discussed in the following section Function name.

```
# Good
doNothing <- function(x, y) {
  return()
}

# Bad
```

```r
do_nothing <- function(x, y) {
  return()
}
```

2.2.4 Files

All file names should be in lower case: datafile, scripts, log files, image files, other output files. Some OS are not case sensitive and this could lead to unexpected issues. The extension can mix case to respect R standards: `.R` extension for scripts, `.RData` extension for data. If needed `use_snake` case to separate words

```r
# Good
do_nothing.R

# Bad
doNothing.R
```

2.3 Packages

Explicitly qualify package name (i.e., namespaces) for all rarely used external functions. Some different packages are using the same function names and this may lead to unexpected error messages.

```r
# Good
IrisSepal <- iris %>%
  dplyr::select(Species, Sepal.Length, Sepal.Width) %>%
  dplyr::arrange(-Sepal.Length)

# Bad
IrisSepal <- iris %>%
  select(Species, Sepal.Length, Sepal.Width) %>%
  arrange(-Sepal.Length)
```

2.4 Pipe %>%

Pipes can be used but with parsimony. Don't use them for short assignments (one line)

```r
# Bad
Species <- iris %>%
  dplyr::pull(Species)

# Good
Species <- iris$Species
```

Try to avoid to use them for highly repeated assignments. Store intermediate results if this can help to check your code or if you can give meaningful names (if result is a step in your analyse)

Don't use the in place pipe %<>%(it requires another package and is not widely used)

```r
# Good
iris <- iris %>%
  dplyr::select(Species, Sepal.Length, Sepal.Width) %>%
  dplyr::arrange(-Sepal.Length)

# Bad
iris %<>%
  dplyr::select(Species, Sepal.Length, Sepal.Width) %>%
  dplyr::arrange(-Sepal.Length)
```

3 Functions

3.1 Function name

A function must reflect an action. Therefore, strive to use verbs for function names.

Ideally, try to be consistent and use `camelCase` for function name, and `lower case` for its arguments.

3.2 Return

Use explicit returns. Do not rely on R's implicit return feature. It is better to be clear about your intent to return() an object.

```
# Good
addValues <- function(x, y) {
  return(x + y)
}

# Bad
addValues <- function(x, y) {
  x + y
}
```

3.3 Call

Please always specify the name of the arguments when calling a function.

4 Code documentation

There are two types of comments. Those which help to structure the code (header, section break) and those which give information about your code.

4.1 Structure

Add sections to your script to structure your scripts and make it easier to navigate from chunk to chunk (shortcut: `Ctrl + Shift + r` for PC or `Cmd + Shift + R` for Mac). Use subheadings as much as necessary.

This also allows the use of the RStudio tree view for navigating your code (see tree button).

```
# 0 - Script information ------------------------------------------------

# 1 - R packages --------------------------------------------------------

# 2 - Import data -------------------------------------------------------

## 2.1 - Case-based -----------------------------------------------------

## 2.2 - Aggregated -----------------------------------------------------
```

Use a standard header when you create a R script (see snippets `epihead`) .

It is useful to mark the end of a script because it's help to know which script has run correctly.

```
# END of main.R ---------------------------------------------------------
```

4.2 Comments

Add comments to your code! Uncommented code could be very difficult to maintain Insert simple comments (no section) before every chunk of code

```
# Renaming of the variable
```

They should mainly explain why you are doing something. Comments are there to help people who don't know your project to understand your objectives. You don't need to explain R syntax except for tricky constructions.

```
# Bad
# Renaming of the variable

# Better
# Renaming of the variable according to the data dictionary
# stored in whatever_data_disctionnary.csv


# Bad
# Checking data

# Better
# Checking for incorrect date format in vaccination dates
# to build list of inconsistencies (sent to country)
```

5 Tests

6 NEWS

7 GitHub

8 Conclusion

But remember... this is just a guide, it is not there to cause us more problems than it solves ! Guide is there to help as much as possible, it's not an absolute ! If following those rules seems difficult, then we could re-evaluate them.

Have fun !