- INWT
- Blog
- INWT's guidelines for R code

INWT's guidelines for R code

25.01.2018 14:05
by Mira Céline Klein

> *"It turns out that style matters in programming for the same reason that it matters in writing. It makes for better reading.“* **Douglas Crockford in JavaScript: The Good Parts**

Why do we need yet another style guide?

"The reason to care about a style guide is just one thing: We want that our source code is not only interpretable by a computer but also by a human.“

If you start to write not only for yourself but also for others, you do not want to lose your readers. You have to start to care about format, style and clarity. The format and partially the style can be addressed by sticking to a style guide. However, your code is not becoming prose just like that. If an author does not care if her code is readable, even a style guide won't help. On the contrary, if you do care about readability, format and style are the easiest things to get started with and it will have an immediate impact.

When you begin to care about readability, here is a short list of benefits:

- Your colleagues will like you and are happy to pick your work up, where you left off. Well, not quite, but we can get there.
- Errors are easier to spot.
- When you review your own source code you will fewer and fewer ask yourself: Who wrote this? And what does it mean?

When you decide to stick to any style guidelines, be aware that it is not that important which one you pick. What is important is that you and your team can agree on sticking to it. There are two popular style guides for R. One is the style guide from Google and the other one can be found in Advanced R by Hadley Wickham.

We deviate only on some specific rules from these guides. Furthermore, we have decided to make them mandatory. Every time code is committed into a git repository, a CI tool will automatically check for any style violations. Another component is a template for a typical script which can be found below. We also implemented most of these rules and some more ideas in a dedicated package INWTUtils which can be found in its GitHub repository.

Naming convention

In contrast to the two style guides mentioned before, we decided to use lowerCamelCase for most identifiers. Underscores and dots are not allowed in usual object names. There is one case where you could deviate from this convention: By naming GLOBAL_VARIABLES – they are only used in scripts and only defined at the top – in upper case with underscores, they can be found at a glance.

If you want to go a step further, you can even check your R code for deviations from style rules with automated checks. The lintr package contains useful functions to run such checks. You can apply these checks to a package or a single file. While doing so, you can choose exactly which rules you want to test. Our own checks are:

- Use <- for assignment, not =
- Spaces should be after all commas, but not before
- Operators like +, -, =, … should be surrounded by spaces
- The line length should not exceed a certain number of characters, e.g., 80 or 100
- Spaces should be used instead of tabs
- Object names should not be too long
- Left parentheses should have space before, except for function calls
- Remove empty blank lines at the end of the file

We also wrote a few of our own checks which can then be easily integrated with the package:

- Function arguments without default should be listed before arguments with default
- Avoid double spaces (except for indents)
- Don't use *setwd* or *source* in package functions – this can lead to unexpected side effects
- Remove whitespaces at the end of lines
- We want to force ourselves not to use internal functions from one of our own packages (via *pkg:::foo*). This would be a hint that we should rather write a documentation for that function and export it. This rule applies only to scripts, not to package functions.

If you want to get started, we recommend you to install the package *lintr* and read the help for *lintr::lint*.

In the beginning, it may be hard to follow these style rules, especially if they have been chosen by someone else and are not in line with your personal preferences. But eventually, it will help you to write cleaner code. In addition, you may adapt a consistent coding style to your team, which makes cooperation a bit more comfortable.

The structure of a script

Header

It's a good idea to start a script with a header including the purpose, the files it needs as input, and the output it produces. In addition, the header should list the author's name and email address. This can also be an incentive to take on ownership: You may look at your code more carefully if your name is written above it.

Cleaning up the workspace

The first step is to clean up your workspace, e.g., removing all objects. Otherwise, it could happen that your code uses an object from the workspace which will not be available anymore at a later point.

Cleaning up the search path

The search path contains all packages that are attached, that is, whose functions you can access directly. Some packages may be attached and you are not aware of them anymore. If you want to keep control, it's

packages you need for the analysis via *library* and execute source statements.

Contents

Now you can start with the actual analysis. First, declare your global variables, if any. By declaring all global variables at the top of your code you will always find them quickly and not overlook any of them. To obtain a clear structure, you can divide your code into numbered sections (e.g. “loading data”, “data preparation”, “model estimation”, etc.). This may seem quite trivial, but it helps to get an overview very quickly. Also, modern editors may provide the possibility to collapse regions of code. This can further help to browse a file.

```
################################################################################
# This script is a template for the general structure of R scripts. This first #
# header should describe what the script does, which input files it needs and  #
# which output it produces. This header also contains name and e-mail adress   #
# of the author.                                                               #
#                                                                              #
# Author: Mira Céline Klein                                                    #
# E-mail: mira.klein@inwt-statistic.de                                         #
################################################################################


# 00 Preparation ---------------------------------------------------------------

# Clean up workspace
rm(list = ls(all.names = TRUE))

# Clean up search path
INWTUtils::rmPkgs()

# Load packages
library(INWTUtils)

# Define global variables
A_GLOBAL_VARIABLE <- 4 # Upper case and underlines for global variables


# 01a Load data ----------------------------------------------------------------

exampleVector <- 1:3 # lowerCamelCase for ordinary object names


# 01b Prepare data -------------------------------------------------------------



# 02 ... -----------------------------------------------------------------------
```

What articles you might also be interested in:

- A meaningful file structure for R projects

Go back

Hauptstraße 8
Meisenbach Höfe,
Entrance 3a
10827 Berlin

+49 30 1208231-0

+49 30 1208231-99

info@inwt-statistics.de

E-mail

Message

SEND

- Training
- Blog
- Imprint
- Data protection

- HomeData Science Solutions: INWT Statistics
- Business CasesBusiness Cases
- PortfolioINWT Statistics - Portfolio
- Meet the TeamINWT Team - concentrated data science expertise
- JobsINWT Statistics - Data Science Jobs in Berlin
- TrainingData Science Training with R
- BlogBlog: Data Science
- Start a Project!Data Science Project