CorrelAid Style guide

Bitte ergänzen und editieren wie ihr es für sinnvoll haltet. Der Styleguide sollte sich sowohl an R als auch an Stata NutzerInnen richten.

Good coding style is like using correct punctuation. You can manage without it, but it sure makes things easier to read. As with styles of punctuation, there are many possible variations. The following guide describes the style we want to use for our projects.

It is based on Hadley Wickham's R Style Guide which is based on Google's R style guide, with a few tweaks.

Good style is important because code will usually have multiple readers and in our case also multiple coders. In that case, it's a good idea to agree on a common style up-front. Since no style is strictly better than another, working with others may mean that you'll need to sacrifice some preferred aspects of your style.

The formatR package, by Yihui Xie, makes it easier to clean up poorly formatted code. It can't do everything, but it can quickly get your code from terrible to pretty good. Make sure to read the notes on the wiki before using it.

Organization and Layout

If everyone uses the same general ordering, we'll be able to read and understand each other's scripts faster and more easily.

- All files begin with a common header structure (see below)

```
# CorrelAid
# Projekt: Projektname
# Kurztitel für Skript, der beschreibt was das Skript tut
# Coders: Dieter Daten <dieter.daten@gmx.de>, Gitte Hub <gitte.hub@uni-siegen.de>
# Zuletzt verändert am: 11.11.2015
```

- Scripts are structured as follows
  - Header (see above)
  - Verbose file description comment, including purpose of program, inputs, and outputs
  - **R** source() and library() statements
  - Function definitions
  - Code

Notation and naming

File names

- File names for scripts should be meaningful and end in `.R`.

  # Good

  fit_models.R utility_functions.R

  # Bad

  foo.r stuff.r

- If files need to be run in sequence, prefix them with numbers:

  0_download.R 1_parse.R 2_explore.R

Pay attention to capitalization, since you, or some of your collaborators, might be using an operating system with a case-insensitive file system (e.g., Microsoft Windows or OS X) which can lead to problems with (case-sensitive) revision control systems.

- Never use filenames that differ only in capitalisation.

Data

- Always save data in formats that are readable by both R and Stata: preferably `.csv` (If you use `.dta` always use saveold to save in Stata format).

Object names

> "There are only two hard things in Computer Science: cache invalidation and naming things."
>
> — Phil Karlton

- Variable and function names should be lowercase.
- Use an underscore (`_`) to separate words within a name.
- Generally, variable names should be nouns and function names should be verbs.

- Strive for names that are concise and meaningful (this is not easy!).

Although standard R uses dots extensively in function names (`contrib.url()`), methods (`all.equal`), or class names (`data.frame`), it's better to use underscores. For example, the basic S3 scheme to define a method for a class, using a generic function, would be to concatenate them with a dot, like this `generic.class`. This can lead to confusing methods like `as.data.frame.data.frame()` whereas something like `print.my_class()` is unambiguous.

"'{r, eval = FALSE} # Good day_one day_1

Bad

first_day_of_the_month DayOne dayone djm1 "'

- Where possible, avoid using names of existing functions and variables. This will cause confusion for the readers of your code.

```
{r, eval = FALSE} # Bad T <- FALSE c <- 10 mean <- function(x)
sum(x)
```

Figures

- Code that produces a figure is kept in its own `.R`-file.
- File names for code and figure that it produces share the same name.
- Save figures as `.png` (for html) and `pdf` (for PDF)
- Always save figures in an object so that it can be called up by other scripts.

Tables

- Code that produces a table is kept in its own `.R`-file.
- File names for code and table that it produces share the same name.
- Save table as `.md` (for markdown), as `.html` (for html), as `.tex` (for PDF)

knitr

- When using knitr to produce output never include substantive code in the knitr document.
- Only use `print()`, etc. to display output produced by scripts you sourced in the knitr document.

Loading & Saving files

- The working directory is always the main folder of the repository.
  - knitr is an exception here as it automatically sets the directory containing the `.Rmd` or `.Rnw` file as working directory. File paths in knitr documents have to be specified accordingly.
- Always comment out `setwd()` in your code if you, for convenience, want to include the path to your local directory in the code.
- Always comment out code that loads or reads data from the hard drive. This includes loading and saving data, figures or tables.

This makes running these commands less convenient but has a number of advantages. If you by accident run a complete script it will not make any unwanted changes to the files on your hard drive (no need to download the repo again). If you source scripts this avoids errors arising from a sourced script trying to load a file it does not find (for example when using knitr).

Syntax

Spacing

- Place spaces around all infix operators (`=`, `+`, `-`, `<-`, etc.).
- The same rule applies when using `=` in function calls.
- Always put a space after a comma, and never before (just like in regular English).

“‘{r, eval = FALSE} # Good average <- mean(feet / 12 + inches, na.rm = TRUE)

Bad

average<-mean(feet/12+inches,na.rm=TRUE) “‘

- There's a small exception to this rule: `:`, `::` and `:::` don't need spaces around them.

“‘{r, eval = FALSE} # Good x <- 1:10 base::get

Bad

x <- 1 : 10 base :: get “‘

- Place a space before left parentheses, except in a function call.

“‘{r, eval = FALSE} # Good if (debug) do(x) plot(x, y)

Bad

if(debug)do(x) plot (x, y) “‘

Extra spacing (i.e., more than one space in a row) is ok if it improves alignment of equal signs or assignments (`<-`).

```
{r, eval = FALSE} list( total = a + b + c, mean = (a + b + c) / n
)
```

- Do not place spaces around code in parentheses or square brackets (unless there's a comma, in which case see above).

“‘{r, eval = FALSE} # Good if (debug) do(x) diamonds[5,]

Bad

if (debug) do(x) # No spaces around debug x[1,] # Needs a space after the comma x[1 ,] # Space goes after comma not before “‘

Curly braces

- An opening curly brace should never go on its own line and should always be followed by a new line.
- A closing curly brace should always go on its own line, unless it's followed by `else`.
- Always indent the code inside curly braces.

“‘{r, eval = FALSE} # Good

if (y < 0 && debug) { message(“Y is negative”) }

if (y == 0) { log(x) } else { y ^ x }

Bad

if (y < 0 && debug) message(“Y is negative”)

if (y == 0) { log(x) } else { y ^ x } “‘

- It’s ok to leave very short statements on the same line:

```
{r, eval = FALSE} if (y < 0 && debug) message("Y is negative")
```

Line length

- Strive to limit your code to 80 characters per line. This fits comfortably on a printed page with a reasonably sized font. If you find yourself running out of room, this is a good indication that you should encapsulate some of the work in a separate function.

Indentation

- When indenting your code, use two spaces. Never use tabs or mix tabs and spaces.
- The only exception is if a function definition runs over multiple lines. In that case, indent the second line to where the definition starts:

```
{r, eval = FALSE} long_function_name <- function(a = "a long
argument", b = "another argument", c = "another long argument")
{ # As usual code is indented by two spaces.  }
```

Assignment

- Use `<-`, not `=`, for assignment.

```
{r} # Good x <- 5 # Bad x = 5
```

Functions

- Should be verbs, where possible.
- Only use `return()` for early returns.
- Strive to keep blocks within a function on one screen, so around 20-30 lines maximum.

Organisation

Commenting guidelines

- Comment your code. Each line of a comment should begin with the comment symbol and a single space: `#` for **R**.
- Comments should explain the why and what.
- Short comments can be placed after code preceded by two spaces, `#` in **R** and then one space.
- Use commented lines of `-` and `=` to break up your file into easily readable chunks.

“‘{r, eval = FALSE} # Load data ——————————

Plot data ————————————————

“‘

Don’ts

- **R** The possibilities for creating errors when using `attach()` are numerous. Avoid it.