R Coding Best Practices

Florian M. Hollenbach

17 August 2020

Cheatsheets & R help

You can find lot's of help for packages that are maintained by the Rstudio crowd on the Cheatsheet page: https://rstudio.com/resources/cheatsheets/. For more advanced coding, I recommend the Hadley Wickham book, in part because it is free (but you should buy a copy if you can afford it): http://adv-r.had.co.nz/.

In general, there are lots of resources online and a lot of questions are answered on Stack Overflow. For example, just found on Twitter today, here is a resource Joscelin Rocha Hidalgo put together on GitHub: (https://github.com/Joscelinrocha/Learning-R-resources/wiki)[https://github.com/Joscelinrocha/Learning-R-resources/wiki].

Some best practices for writing r-code

(To Be Continuously Updated)

Try to adhere to the r-studio style guide as much as possible

- If possible and sensible, follow the r-studio style guide: http://adv-r.had.co.nz/Style.html. It will make your code more readable.

Comment, Comment, Comment

- You should add comments to almost every line in your code. This will help others to understand your code but also help your future self when you come back to the file one, two, or ten years from now.

No use of `rm(list = ls())` at beginning of files

- Do not empty the workspace with `rm()`.
- `rm()` keeps many previous settings .
- For example, all the packages that were already loaded remain or options like `options(stringsAsFactors = FALSE)` remain set.
- *Instead*: always start with a fresh r-session.

No use of `setwd()`

- Instead: *use the here package*, see: https://github.com/jennybc/here_here
- If you use r-studio, create an R-project for the project in the main folder.

- When loading `library(here)` within the project in rstudio, the main folder will become the basis from which to declare any paths.
- If you use other editors, you can create a .here file in the main folder.
- Use `here()` to declare paths from main folder.
- This means no more changing of paths, etc.
- The code will run on different machines or at different points in time. Even if we rename the main folder at some point or move it to a different location.

No massive package loading

- Instead: *only load those packages that are actually used in the file.*
- Also, pay attention to function conflicts when loading, i.e., which functions are masked when loading a new package.
- For example, if tidyverse is loaded the lag functions within plm, lfe will not work!
- Instead: *use the conflicted package*, see: https://github.com/r-lib/conflicted
- The conflicted package will alert/throw an error when using any function that has conflict.
- Also, when loading a package that has a conflicted function, immediately after loading the package declare which package to use the function from, e.g.: `conflict_prefer("filter", "dplyr") ### use filter from dplyr`.

Tables

- For Tables I now recommend using the *modelsummary* package https://vincentarelbundock.github.io/modelsummary/index.html. It is great to work with and has lots of functionality. Also, the package author (Vincent Arel-Bundock) is super helpful and responsive.