Let's build an R-package together!

Daniel Heimgartner

April 11, 2025

1 WHY?

What is an R-package?

- You are free to organize your code however you want!
- But: There are good and bad practices (e.g., stuff like `setwd()`, `read.csv("/foo/bar.csv")`)
- A package is nothing else then a standardized way to organize *information* (not necessarily code!)
- An (R) package is a organized repository/folder structure with a *DESCRIPTION* and *NAMESPACE* file.
- It is nothing to be afraid of!

What are the benefits?

- Code can be easily shared and is installable (i.e., loadable via `library(Rpackage)`)
- Code runs everywhere
- Code, data, documentation, paper! is organized in a comprehensible way (i.e., in a way that humans and computers understand and *expect*)
- If you conform to this structure you get a lot of additional benefits/tools for free (e.g., nicely formatted doc/man pages, easily generate a webpage, etc.)

What will I learn?

- How to scaffold the package structure
- Where and how to put your analysis scripts
- How to write useful helper functions in separate source file (which can easily be used in other scripts)
- How to attach data to the package
- How to document objects (package, data, functions)
- How to write your paper inside R, using the vignette
- How to share the code with your colleagues
- Where to learn more

2 LET'S GET STARTED

`usethis::create_package()`

This will scaffold the R-package. Let's run

```
usethis::create_package("groupRetreat", rstuio = TRUE)
```

and navigate into the `groupRetreat` folder. What did we get?

If you know git, run `usethis::use_git()`

```
.
├── DESCRIPTION
├── NAMESPACE
├── R
└── groupRetreat.Rproj
```

`usethis::use_data_raw()`

The `data-raw` folder is where the untouched raw data goes. But also all the scripts that manipulate this raw data and generate some other output (e.g., data for the analysis). I usually put all my scripts there since it allows me to easily store intermediate `objects`, attach them to the package and use them at a later stage (e.g., a model fit as returned by `fit()`). Let's get concrete

You can also put raw data from a `.csv` *or some other file format. It is convention to name the raw data the same as the script which manipulates it, i.e.,* `useful_data.csv` *in this case...*

```
usethis::use_data_raw("useful_data")
```

This will create the `data-raw` folder and initialize the `useful_data.R` script. An example what to put there

```
## code to prepare `useful_data` dataset goes here
## e.g., fetch data from the qualtrics server
useful_data <- iris
## clean it
useful_data <- useful_data %>%
    rename(species = Species, sepal_length = Sepal.Length, sepal_width = Sepal.Width,
           petal_length = Petal.Width, petal_width = Petal.Length)  # classic mistake ;)
## this is the magic line!
usethis::use_data(useful_data, overwrite = TRUE)
```

Run the script! What happened?

This will generate the `data` folder and the R-data `useful_data.rda`. You will run this function a lot! It essentially simulates a `library(groupRetreat)` call (i.e., loads the package is it was installed).

See why this convention is helpful! Any person familiar with the R-package workflow expects that an object part of a package was generated in the `data-raw` *folder, in a script with the same name.*

`devtools::load_all()`

```
devtools::load_all()
groupRetreat::useful_data
```

Maybe we have a function that we use in multiple scripts (during data preparation in `data-raw`*) or want other users to benefit from...*

Everything in the `R` folder gets sourced!

It is convention to name the source file the same as the function name. I.e., a function `foo()` *lives in* `foo.R`*! It's just a convention, you can put multiple functions into the same source file if you like...*

```
foo <- function(msg = "Are you sleeping?") {
    cat(msg, "\n")
  }
```

Again, "test it"

```
devtools::load_all()
foo()
```

```
Are you sleeping?
```

What did we learn so far?

- What an (R) package is and (hopefully) we are all convinced that it is pretty easy and useful!
- How to organize raw data and generate analysis-ready data.
- How to "outsource" functions that we frequently use or want other users to be able to consume

Sometimes it is also reasonable to write a function only for organizational sake and not clutter the source file. Usually if you write a function such as `make_america_great_again()` *you can forget about the implementation details...*

3 THE DOCUMENTATION GAME

The above gets you already quite far. Did you ever wonder, how these fancy R-helppages are generated? Documentation and code should live together (if possible). We can use the **roxygen2** package to easily document code using a special comment syntax (starting with `#'`). Move back to the `R/foo.R` file, place the cursor inside the function body and press `Shift+Ctrl/Cmd+P`. Type "Insert a rox" and hit enter. This scaffolds the *roxygen skeleton*.

Documentation as "code"

Anything you can achieve in RStudio is available through this command palette (as in most code IDEs)...

Let's explain what this function does

```
#' For Orientation Only
#'
#' This function does nothing useful, but it could!
#'
#' @param msg a message to `cat` to the console.
#'
#' @returns NULL
#' @export
#'
#' @examples
#' foo("No, I am not!")
foo <- function(msg = "Are you sleeping?") {
    cat(msg, "\n")
  }
```

`devtools::document()`

This *parses* the roxygen skeleton and generates the `man` folder (manpages are the helppages) as well as the `.Rd` documentation (which is R's markup, similar to LaTeX, for documentation). But we don't have to know about this – all we need to do is write the magic comments and run `devtools::document()` to update it. I will update this document and show you – I promise!

TODO
How to document data?

4 PAPER VIGNETTE

`paperPackage::scaffold()`

What is reproducible research? Anyone should be able to generate your results on the fly!

TODO

`devtools::install_github("d`

5 SHARING IS CARING

`devtools::build()`

This builds the source package (`.tar.gz` – a compression file format similar to `.zip`) which can be installed by ...

```
install.packages("../groupRetreat.tar.gz")
```

You can also upload the code to GitHub and then people can install it via `devtools::github_install()`. *See https://github.com/dheimgartner/Rpackage*

6 TAKE AWAY

Let's recap the whole game by recalling what these functions do (do you remember?)

- `usethis::create_package()`
- `usethis::use_data_raw()`
- `devtools::document()`
- `devtools::load_all()`
- `devtools::build()`

usethis *and* **devtools** *provide many more helpful functions for package development!*

7 NEXT STEPS

- Learning by doing – try it for your next project/paper!
- This gets you going but it only scratches the surface (e.g., how to properly manage external dependencies)
- Tip: Run `devtools::check()` and learn from the errors, warnings and notes...
- Please learn some git for versioning (and necessarily some bash)

Resources

bash, git (GitHub is "just" the remote repository – a website/app, but some very nice features for project management, e.g., the issue tracker), gitbash for windows

TODO

REFERENCES

Cameron AC, Trivedi PK (2013). *Regression Analysis of Count Data*. 2nd edition. Cambridge University Press, Cambridge. 5

A TO CITE OR NOT TO CITE

A nice book: Cameron and Trivedi (2013)

B R CODE

Wow this is some real code!

```
f <- function() {
    cat("Hello, world!\n")
  }
f()
```

```
Hello, world!
```

C TALKING code

```
usethis::create_package()
```

```
x <-1
```

D MATHEMATIK, MATHEMATIK — IMMER DIESE MATHEMTIK

$$a^2 + b^2 = c^2 \tag{1}$$