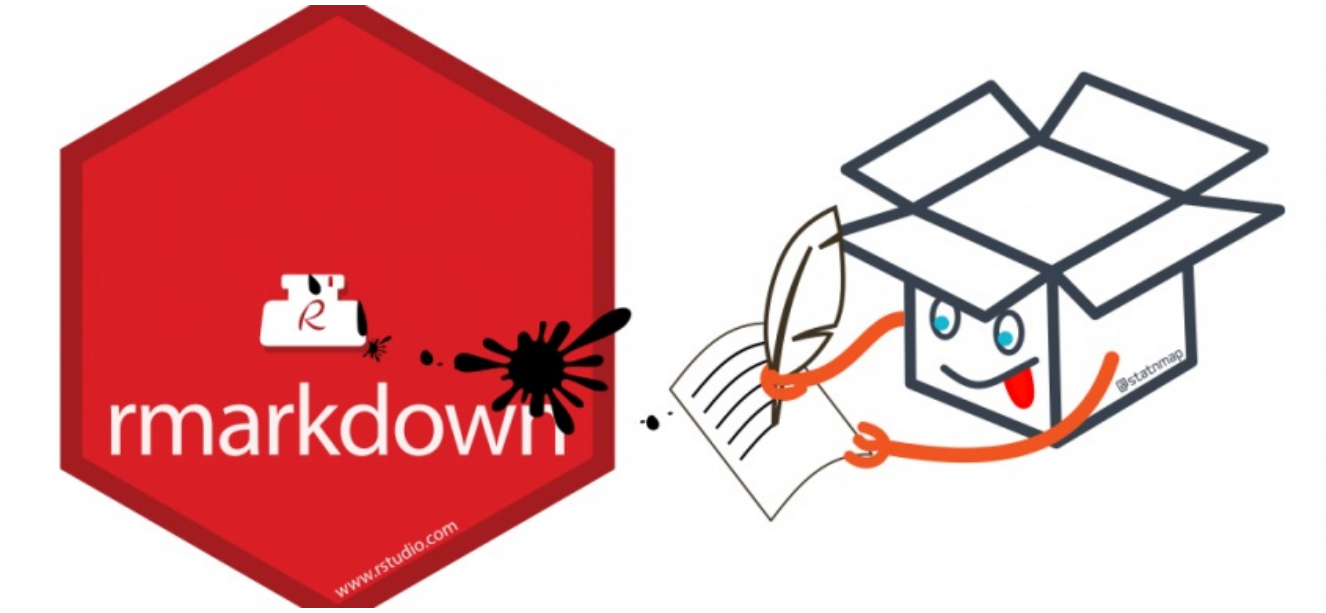
# (en) The R Task Force

rtask.thinkr.fr/blog/rmd-first-when-development-starts-with-documentation/

By: Sébastien Rochette

July 10, 2019




**Documentation matters ! Think about future you and others. Whatever is the aim of your script and analyses, you should think about documentation. The way I see it, R package structure is made for that. Let me try to convince you.**

At use'R 2019 in Toulouse, I did a presentation entitled: 'The "Rmd first" method: when projects start with documentation'. I ended up saying: Think Package ! If you are really afraid about building a package, you may want to have a look at these slides before. If you are not so afraid, you can start directly with this blog post. In any case, `Paco` the package should make this more enjoyable ! I hope so...


*This article is quite long. You can do a quick read if you skip the bullet points. Use the complete version while you are developing a real project.*

# Why documentation matters ?

## Prevent code sickness

- You never took back a R script of yours, 6 months later? Did you perfectly remember how worked every parts of the code? Functions and their parameters?
- You never had to improve/debug the code of somebody else?
- You were never asked to add functionnalities in a Shiny application originally written by a master trainee and then ponctually modified by the supervisor to “just” add one or two functionnalities?

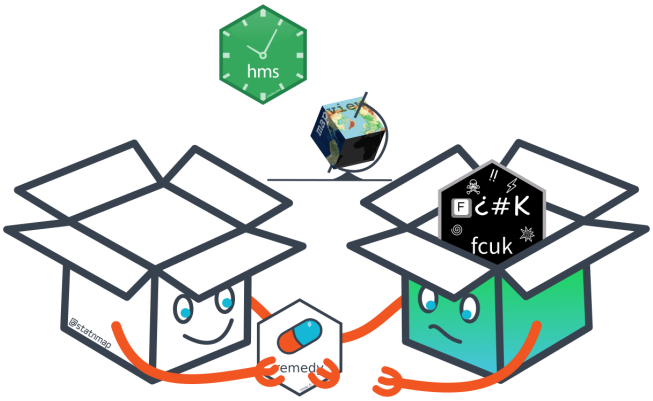
> If not, you are lucky. For now…

In any case, **think about future you and think about future users** or developpers of your work. Never think this is a one shot. One day or another, this one shot script will again be useful. It will be more useful if it is correctly documented.
Multiple reasons would make documentation necessary:

- You share your scripts with your young padawan and you do not really have time to spend with him/her to explain each step. You prefer spending your time for more constructive discussions around the project, than on the use of a specific function.
- A colleague has to run your analysis when you are on a sick leave. Sick leave is generally not something you planned.
- You want to share your work with the World ! One day or another, you will be asked to share your scripts because it can interest people out of your team. Or maybe you

think that this can be valuable to share your code with a R-community that gives you so much every day...

> Don't provoke (to much) sickness to people who will have to understand and use your code… Give them the remedy, give them a documentation.




## If you read this post, you are good enough to create a package

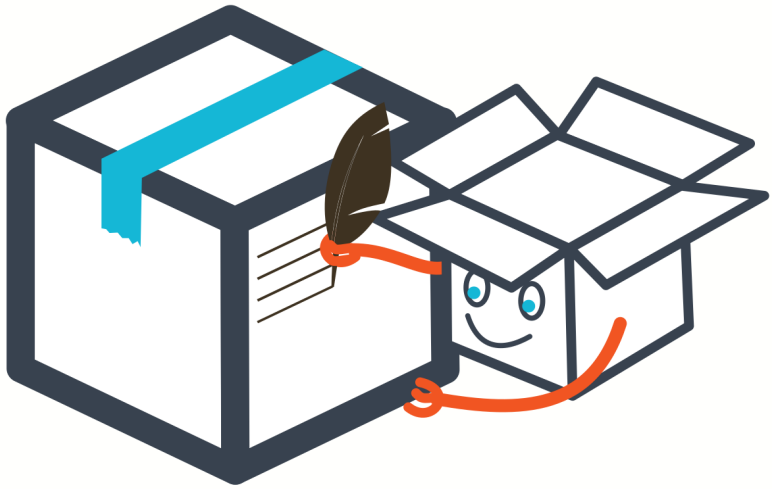
You are not here by chance. You are here because one day, you wrote a line of code in the R language. Thus, **trust me, you know enough to create a package**.

When we tell our students/trainees, after their two first days, that the next step is to build a package, the first reactions are:

> I just started, I am not developer who can create a package.
> I do not want to share my work on the CRAN with the World, not even on Github.

Our answers are:

> First "copy-paste" of code requires a function, first function requires a package.
> You can create a package in six minutes and I'll show it to you now: Create a package in a few minutes (text in French but video does not require audio).
> You do not know what you are capable of. Trust me, in the next two days of tutorial, you will be able to create your own package, at least for internal use.



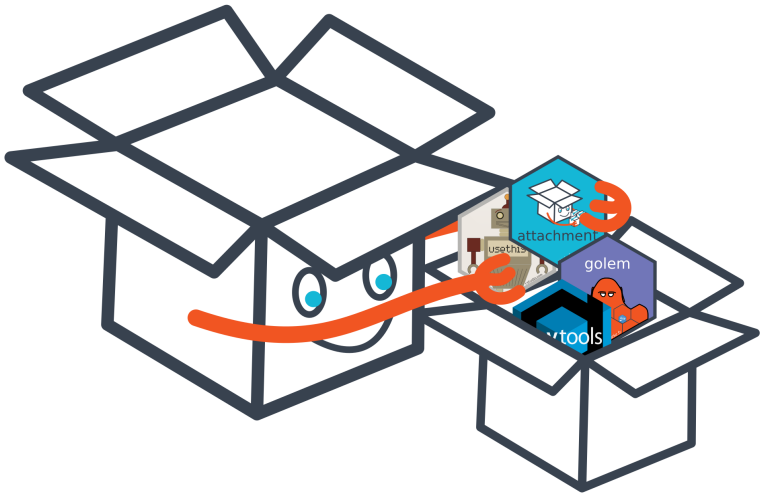
## A package forces standardized documentation

- Package forces standardized general description of your project
- Package forces standardized documentation of functions
- Package recommends to show reproducible examples for each function
- Package allows integration of user guides (vignettes)
- Standardized structure of a package and its `check` are supposed to conduct to a re-usable code

There are different places in a package where to write some code, some text, some information, ... Remembering the complete procedure of package development may be painful for beginners, even for advanced developers. Hence, some think they need to build a hundred packages before being able to get all of it. That's not true.

> The 'Rmd first' method may reduce the fear of package building by developing as if you were not in a package

Moreover, **some useful packages are here to help** you in all package development steps, among them {devtools}, {usethis}, {attachment}, {golem}, {desc}, ...




## Develop as if you were not in a package

In my presentation at use'R 2019 in Toulouse, I started developing my code in a "classical" R project. If you looked at this presentation, you know we will end up with a package. Here, we will directly start by building a package skeleton. Just a few files to think about before developing. You will see that everything is about documentation.
In this blog post, we will start to create a small data analysis in a project named `my.analysis.package`.

## These steps looks like package development...

Your project needs to be presented. What is the objective of this work? Who are you? Are others allowed to use your code? We also keep a trace of the history of the package construction steps.

### Create new **"R package using devtools"** if you are using Rstudio

- Or use `usethis::create_package("my.analysis.package")`.
- *A constraint is that your project name can only consist of letters, numbers and dots. It must start with a letter; and it cannot end with a dot.*

### Create a file named `dev_history.R`, for instance at the root of the project

- This file stores the documentation of your workflow. **This makes the creation and development of your package reproducible**
- Do not forget to hide it from package build using `usethis::use_build_ignore("dev_history.R")`
- Use it to store all your calls to `usethis::`, `devtools::`, `attachment::` and co...
- Have a look at what we used in this file for {attachment} development or what code {golem} proposes for your Shiny Application development workflow

## Fill in the project `DESCRIPTION` file

- Give a title to your analysis
- Write a description sentence. This sentence should finish with a point `.`. If you need to write the description on multiple lines, you need to indent the text with two extra spaces at the beginning of each new line.

## Define the license

- You may think about licensing all of your work. You never know who is going to read and re-use it in the future.
- If you do not know what to choose for now, you can choose a proprietary license. Use `License: file LICENSE` in the `DESCRIPTION` file and then create a file called `LICENSE` (without extension) at the root of the project, containing for example:

```
Proprietary
Do not distribute outside of ThinkR.
```

## Add the following lines in your `dev_history.R` script

Execute these lines each time you see: in this blog post. *Including now*.


```
# Document functions and dependencies
attachment::att_to_description()
# Check the package
devtools:check()
```

This should result in **`0 errors, 0 warnings, 0 notes`**


# Work with your data in a Rmd document almost like classical project development

**The Rmd file is your sandbox** before being the userguide of your work. Use it to try, explore, test your data and analyses. Once you are ready, follow the guide to clean up the content and store everything at the right place.


## Create a small dataset

A small dataset will help in the documentation process

> If you are building this package for a specific analysis for your work or for a client, you may think you want to you use your entire dataset. Directly manipulating a big dataset can be painful during code writing. You may face long computation time while debugging. **Do not neglect the power of a small reproducible example !** It may not cover all future possible bugs, but this will accelerate development and help documenting the code.

- As a start, you can store the dataset `my-dataset.csv`, in a folder named `inst/example-data/`. This will later allow to find this data using `system.file("example-data/my-dataset.csv", package = "my.analysis.package")`.
- There are different ways to store data inside a package. The proposed one is an easy way to start. When you are mature enough, you may have a look at chapter on External Data in "R Packages" book to decide on your way of storing the data.

## "Rmd first": **Here starts the "Rmd first" method**.

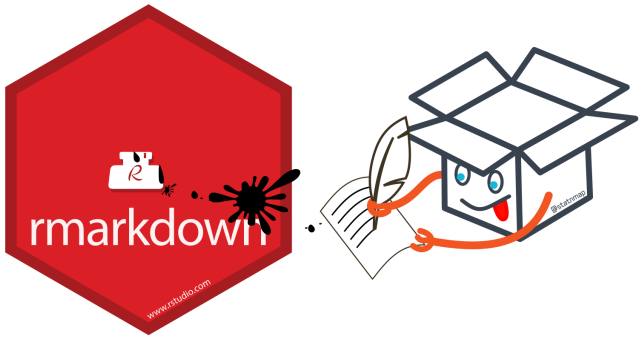
Not really really first, but almost first as we haven't started code development yet. So, let us create a vignette. This is a classical RMarkdown file stored in a specific place and that will be shown in a specifically formatted HTML output.

- `usethis::use_vignette("aa-exploration")`
- If you **choose a filename of your vignettes starting with two letters**, this will be used to show them in the correct order in the documentation. Use the same in the `VignetteIndexEntry` field of the *YAML* of your vignette, but you can remove them in the title. *I recommend using letters instead of numbers because of later use of {bookdown} with package {chameleon}.*
-


## It may now be necessary to install your package.

This will make your dataset available for the vignette.

You can use `devtools::install()`. *Add it in your* `"dev_history.R"` *to execute when needed.*




## The “Rmd first” method looks like classical analysis

The workflow of an analysis often starts with data reading, cleansing, filtering, exploring, ... These steps may be presented in a Rmarkdown file, so that you can create a report of your work and make it reproducible to others.
The **“Rmd first” assumes that any project can start like data analysis**. Whatever project you are doing, you will need some data, some reproducible examples to build it. Below is a tiny example of a Rmd build.

### Load necessary packages at the top of your vignette

> If you need more packages along the development, always go on top to add them so that future users won’t be surprised in the middle of the analysis

### Write a sentence about **what you plan to do** in the next chunk

> Say what you’ll do, do what you said (Diane Beldame, ThinkR)

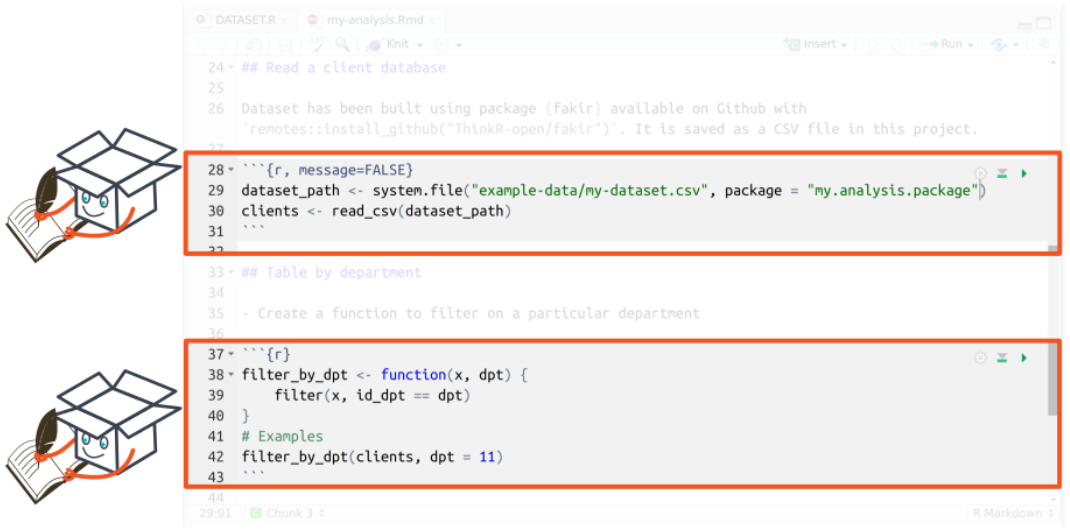

## Do what you said

Load your data

If you installed the package, data path is available using

```
# Get path
datapath <- system.file("example-data/my-dataset.csv", package = "my.analysis.package")
# Read data with {readr} for instance
clients <- readr::read_csv(datapath)
```

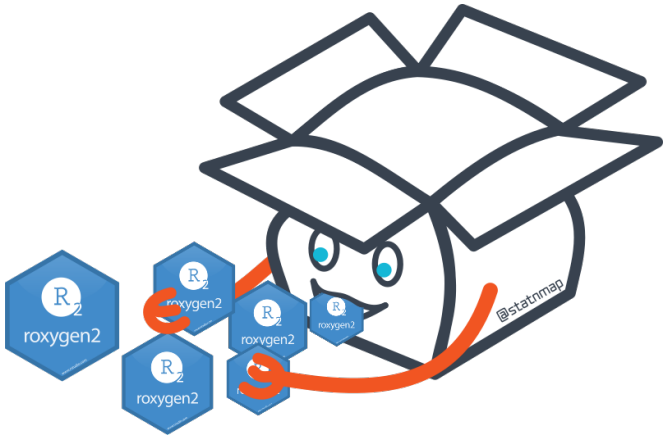
- Write the code you want, apply it on your dataset
- When your code does what you want it to do, transform it as a function




## **Document the function** while you are still in the Rmd

- Use roxygen syntax
- Document the parameters
- Add an example of use: **you already have an example because you wrote your Rmd as a reproducible example**

```
#' Filter by department
#'
#' @param x dataframe with column named id_dpt
#' @param dpt department number
#'
#' @return a filtered dataframe
#' @export
#' @examples
#' dataset_path <- system.file("example-data/my-dataset.csv", package = "my.analysis.package")
#' clients <- read_csv(dataset_path)
#' filter_by_dpt(clients, dpt = 11)
filter_by_dpt <- function(x, dpt) {
  filter(x, id_dpt == dpt)
}
```



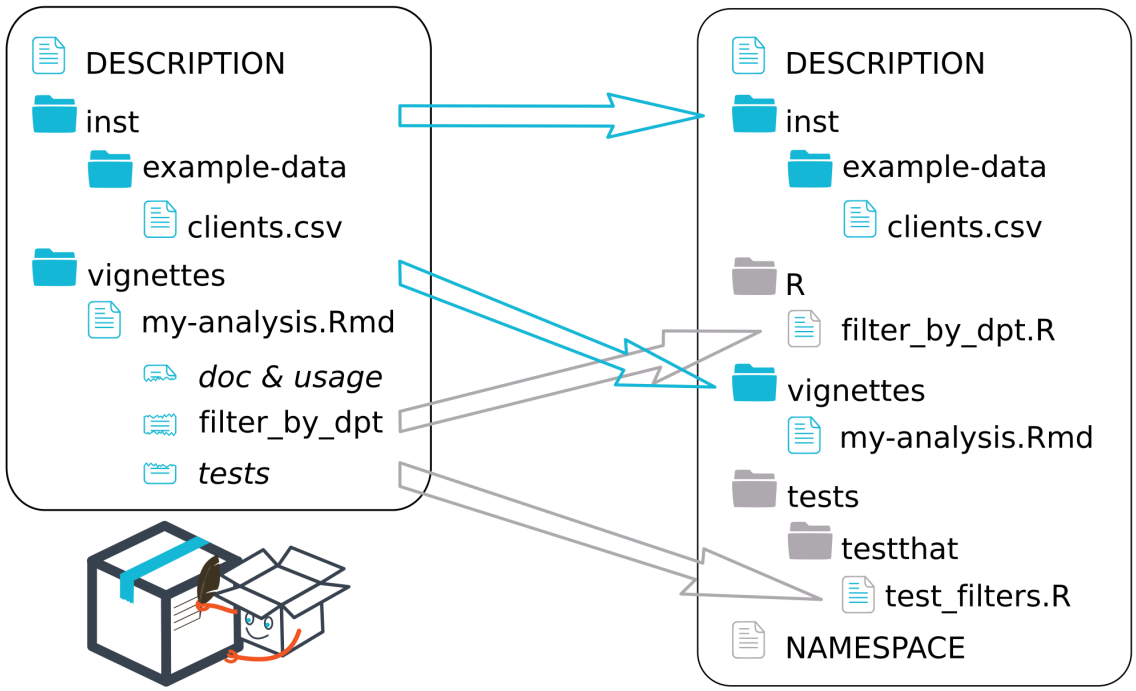

## **Create a test** while you are still in the Rmd

- **You already have a reproducible example to build your test**
- You can try package {exampletestr} to build a test from your function example. *This requires to realise the "Rmd cleansing" first (see below).*

```
dataset_path <- system.file("example-data/my-dataset.csv", package = "my.analysis.package")
clients <- read_csv(dataset_path)
output <- filter_by_dpt(clients, dpt = 11)
testthat::expect_is(output, "data.frame")
testthat::expect_equal(nrow(output), 10)
```


# "Rmd cleansing": Make it a proper a package

**Everything you need to make a proper package is already set.** We will now clean the Rmd. This means we will move parts of the code of the Rmd to the appropriate directory of the package. In the vignette, we will only keep the explanations and the code to use the main functions.




- `DESCRIPTION` is already written
- `inst/` folder does not change
  Again, later you may think about other ways to store your data (See External Data in "R Packages" book)
- `vignette/` folder does not change
  The Rmd file itslef will be cleaned
- Function `filter_by_dpt()` needs to be moved (cut-paste)
  - Create a `R/` folder and a R script file named `filter_by_dpt.R`
    You can use `usethis::use_r("filter_by_dpt")` . *Add it in* `dev_history.R`
  - Cut and paste the code of the function as well as the roxygen skeleton
  - Clean calls to `library` in your vignette that were used for the function and not useful anymore
- Tests needs to be moved (cut-paste)
  Create a `tests/testthat/` folder and a R script file named `test_filters.R`
  You can use `usethis::use_test("test_filters")` . *Add it in* `dev_history.R`
- `NAMESPACE` will be automatically built in the following part

## Polish the package

Your package is set. It is now time to run functions to create documentation in the package format and check if your package correctly follow construction rules.

- Create documentation files. The ones that will appear when you call the help of a function. Fill (automatically) `NAMESPACE` file.
  Run `attachment::att_to_description()` (that contains `devtools::document()` )

- **List all dependencies** of your package in the `DESCRIPTION` file.
    To help you not forget anything, **use** `attachment::att_to_description()`.
- regularly



## Publish your documentation

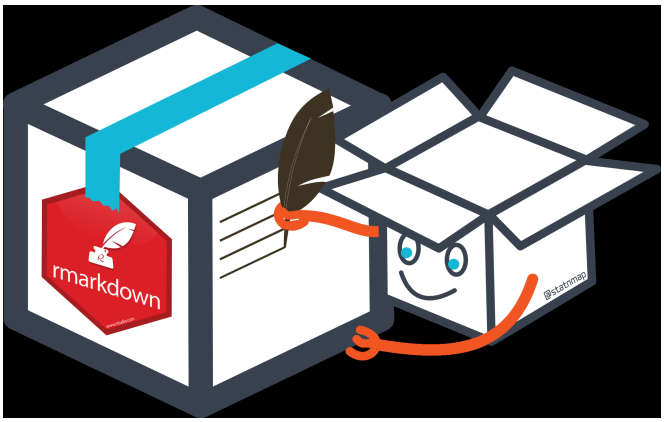
As you created a package, you can use {pkgdown} to present the entire documentation in the same HTML site. The 'Rmd first' forced you to create vignettes that can also be presented as a HTML (or PDF) book, delivered as shareable userguides or as your analysis report.

- Use {pkgdown} to **present your documentation**.
    - This contains vignettes, the Readme (if created) as a presentation of your study, fonctions and executed examples of the functions with outputs.
    - You can present it to your clients as the vignette is the output of your analysis
    - You can use `chameleon::build_pkgdown()` to build and keep the pkgdown site inside your package, so that your clients can call it offline with `my.analysis.package::open_pkgdown()`
- Transform your **vignettes into a real userguide** (or analysis report) using {bookdown} with `chameleon::build_book()`. You keep it inside the package so that your clients can call it offline with `my.analysis.package::open_book()`, or you can deliver the book itself.
    This is the reason why I named my vignettes starting with two letters like `aa-my-analysis.Rmd`
- Combine with a {testdown} report to show your clients that you worked properly with unit tests for each function.


## Everything is a package

A package is not only for archives stored on the CRAN that must be shared with the entire world. A package is a directory containing specific files and folders, organised so that anyone can find documentation, code and tests in a precise place. Hence, every project can be a package, so that tools built around packages will allow/force you to work in a reproducible way with re-usable project. Think package:

- Package of data analysis for a client using 'Rmd first' method allows documentation of function for future analyses and creates a userguide and/or analysis report with {pkgdown}/{bookdown} for the client
- Shiny App with {golem} and Rmd first allows documentation of internal functions for future debugging and improvements, as well as presentation of intermediate steps through static reports.
- Package is a package. While using 'Rmd first', you write documentation along with your code.

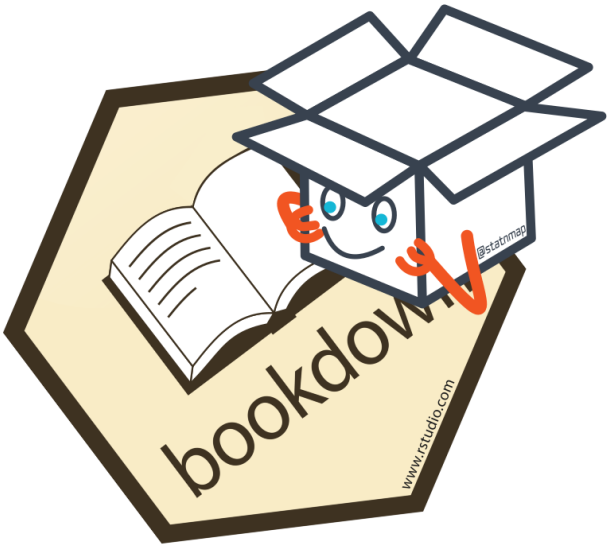



## Ressources

- If you already wrote your analysis in a Rmd file and if you are now convinced that it would be more reusable while inside a package, I recommend to read Emily Riederer blog post entitled "RMarkdown Driven Development (RmdDD)". She presented the

same kind of 'Rmd first' approach and details the steps to clean your Rmd to make it a package.

- All information you need to know about package development may be found in Hadley Wickham and Jenny Bryan "R packages" bookdown
- Building a Shiny Application inside a package using {golem}
- Recommendations for a "Workflow to build [documented] Big Shiny Apps" using {golem} are in this Colin Fay bookdown
- Package {chameleon} to customise and highlight your HTML outputs
- Package {testdown} to present the outputs of your unit tests in a bookdown




Enjoy writing documentation!
Future you and people around you will thank you for that!