Thesis Template using R and knitr

Alexis Sardá-Espinosa

December 4, 2016

Dedication

To me

Abstract

Abstract goes here

Declaration

I declare that..

Acknowledgements

I want to thank...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

I did my thesis in **R**, and it took me a while to get everything working the way I wanted. The example here is a skeleton that could serve as a starting point to write not only theses, but any kind of document using LaTeX and **R**.

The main files have a `Rnw` extension and are transformed into `tex` files by using the `knitr` package.[1] However, anything that doesn't depend on **R** can be written directly in `tex` files, making it flexible.

I didn't put too much effort into the layout shown here, it is only meant to serve as a starting point with some examples. Other templates can also be used.

[1] `http://yihui.name/knitr/`

Chapter 2

Organization

2.1 Compilation Scripts

There are two main files used for compilation: `compile-linux` for Linux users, and `compile-windows.bat` for Windows users. They are both used in the same manner, so in the examples below I will simply refer to the scripts as `compile`.

The first thing that is needed is of course LaTeX. There are many resources available on how to install it, so I will not dwell on that.

The next thing is **R** itself. Again I will not go into too much detail, just make sure that the `Rscript` executable is available in your path. If you have code that will use parallel computation, the default **R** package that is used is `doParallel`, although this can be modified in the file `Thesis.R` (see Section 2.2).

If you simply double click on the `compile` script, it will run `pdflatex`, `bibtex` and `pdflatex` twice (for citations to work). This will apply any changes made to any `tex` file, but will **not** detect any changes made in `Rnw` files, regardless of whether it's **R** code or not. Running the scripts from the command line with no arguments is equivalent.

The other option is to run the scripts from the command line followed by a single argument, either `TRUE` or `FALSE`; this will *knit* the main `Thesis.Rnw` file and apply all changes made, including **R** code. If you use `FALSE`, everything is done sequentially. With `TRUE`, a number of parallel workers equal to the number of detected CPU cores are registered first, so that **R** code can run in parallel. This is only useful if your code is specifically made to run in parallel (e.g. with the `foreach` package).

2.2 Main Files

If the `compile` scripts are called to update the **R** code, they basically call `Rscript` on `Thesis.R`. The latter registers the parallel workers if necessary and calls `knitr` on `Thesis.Rnw`, which creates `Thesis.tex`. It also loads **R** utilities on each worker (see Section 2.2.1).

The master file is `Thesis.Rnw`, which contains the main `tex` layout as well as the defaults for `knitr` (more on that in Section 2.3).

2.2.1 Utilities

By "utilities" I mean the common **R** functions and libraries that will be used. I have made a basic distinction between cached and uncached utilities. This is related to `knitr`'s cache function, so you should definitely read about that first if you don't know how it works. Uncached utilities should be the ones that will be used by `knitr` itself or in the chunk options.

The files are located in the `Utils` folder and have obvious names...

2.3 Default Functionality

2.3.1 Knitr Cache

You can see the defaults for `knitr` in `Thesis.Rnw`. By default, all chunks are cached and hidden from the output (with `echo = FALSE`). Additionally, auto-dependency detection is turned on by default; this has worked well for me.

Uncached utilities are always loaded along with `knitr`'s defaults. The rest of the used libraries and functions can be included in the cached utilities file.

Cache is created in `Data/cache/`.

2.3.2 MD5 Hashes

I've included a function called `md5` that I've used in some chunk options. Since you can add any non-NULL value to chunk options in order to create its cache (again, read `knitr`'s information on cache if you haven't), you can use MD5 hashes to create chunks that depend on specific files, so that if the files change, the chunk will be updated automatically. There is an example of this in the source code for Section 3.1.

It is a semi-custom function because the default `md5sum` doesn't throw an error if it doesn't find the file, in which case you might not notice if you're actually caching a valid file or not.

2.3.3 Workspace Cleaning

I also include a function called `clean_workspace` that removes everything except the uncached utilities. It can optionally reload cached utilities and ignore variables if desired.

I run this at the end of each child `Rnw` file. I do this because I'm not sure how `knitr`'s auto-dependency detection works (my bad), and I wanted each child file to be as independent as possible so that things wouldn't break in one section if I changed something from the previous section. Check the source code for the examples, and note that this cleaning is always uncached.

2.3.4 Dynamic Loading of Utilities

I was using several libraries, and it was annoying to wait for all of them to load if all I wanted to do was update content without changing any **R** code (recall that changes made to `Rnw` files won't be detected unless you re-knit everything). My solution was to use a `knitr` hook (see `knitr`'s website).

The hook is a function called `utils_hook`, located in the uncached utilities file. If you see the defaults in `Thesis.Rnw`, you'll notice every chunk will run the hook by default, which loads cached utilities before actually running the chunk. Since the hooks are only executed if the chunk needs to be updated (i.e. its cache was invalidated or non-existent), utilities are only loaded whenever I actually have to re-run **R** code. Each chunk that needs to be updated will re-load everything, of course, but that's unimportant (i.e., re-loading a library has no effect).

Notice that chunks that are manually uncached should set the `load.utils` hook option to NULL to avoid loading things unnecessarily. See the source code for workspace cleaning chunks for examples.

2.3.5 Tables

I used the `xtable` package to create tables from data frames. I created a (cached) function called `make_table` that has some defaults so that I didn't need to write everything every time. You can see the defaults in the source code and note that any default can be overridden via the ellipsis.

The `results` chunk option should be set to `"asis"` so that everything is detected correctly. See the source code for Section 3.3.

2.3.6 Random Seeds

I set one overall seed in the initial chunk where knitr is configured. I've added another hook (`seeds_hook`) that saves the random seed for every chunk (by default) and restores it whenever it is run, in order to maintain reproducibility. The seeds are saved in the `Data/seeds.RData` file. It should detect new chunks, and the chunk labels are used for naming. If you want to disable this for a given chunk, add `rand.seed = NULL` to the chunk options (see, e.g., the last chunks in the files in the `Examples` folder).

Note that when adding/modifying chunks after initial caching, it could be that some or all chunks end up with new random seeds, especially if some chunks are not cached. If there is a chunk whose reproducibility is crucial, set its seed manually inside the chunk. Alternatively, manually open the seeds file, delete the seeds you want to update, and manually save it again.

2.4 Considerations

2.4.1 File Paths

Be careful with paths, for example you should specify the correct figure path in both `knitr` and LaTeX options; see the source code for Section 3.2.

Additionally, you should know that each child `Rnw` file is run with its own directory as the working directory. However, the paths you give to `knitr` or the ones used in chunk options are always with respect to the root directory, i.e. where `Thesis.Rnw` is located. See the source code for Section 3.1 for an example.

2.4.2 Chunk Names

I believe all chunk names need to be unique.

Chapter 3

Examples

3.1 Basics

The caret package is one that can take advantage of parallel execution. Below, I train a random forest on a sample dataset [Lichman(2013)].

```
chess_rf <- train(label ~ ., data = chess,
                  method = "rf",
                  tuneGrid = data.frame(mtry = 8:15),
                  trControl = trainControl(method = "oob"))

best <- chess_rf$bestTune$mtry
acc <- round(chess_rf$results$Accuracy[chess_rf$results$mtry == best],
             3)
```

Using a random forest with the Chess dataset and OOB error estimate, best value of mtry was 15 with an accuracy of 0.994.

Try changing something in the chunk (to invalidate the chunk's cache) and compiling the thesis with/without parallel support to see the difference in run time.

With the rand.seed hook in place, each chunk should have an associated random seed after you compile everything for the first time. The chunk below is not cached, but its seed is restored everytime you compile, so it should always give the same result. If you add chunks with randomness before it and delete the seeds.RData file to update it, it will probably get a different seed assigned.

```
rnorm(1)

## [1] -0.1180655
```

3.2 Figures

Plots can be created directly in **R** and the results are added automatically to the `tex` file. See Fig. 3.1 for an example of a single plot.

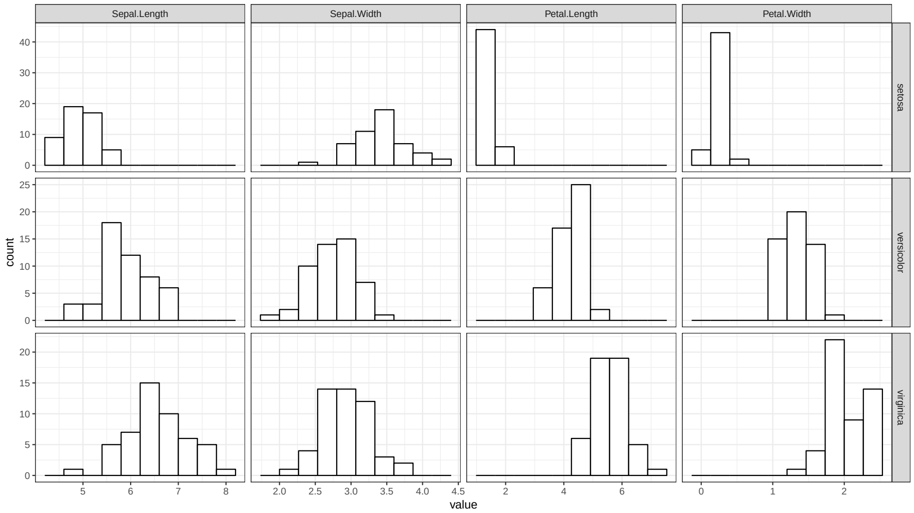
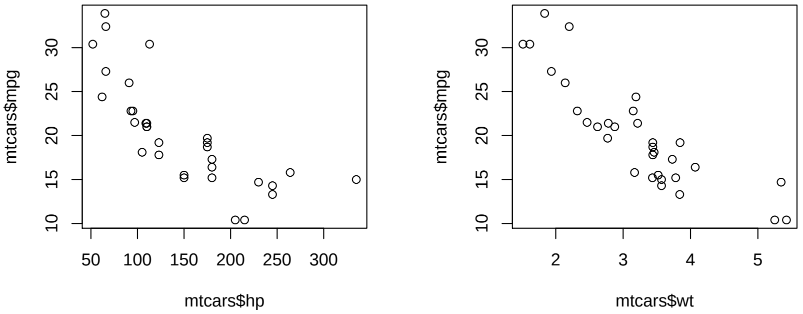


Fig. 3.1: Histograms for the iris dataset. Look at the list of figures to see difference between short and long captions.

You can create several plots per chunk and have them in the same main figure. See Fig. 3.2 for an example of two plots: Fig. 3.2a and Fig. 3.2b.



(a) MPG vs HP

(b) MPG vs Weight

Fig. 3.2: Plots using `mtcars` dataset.

3.3 Tables

Tables can also be created in **R** and inserted automatically. Check Table 3.1.

Table 3.1: First rows of the CO2 dataset. Look at the list of tables to see the difference between short and long table caption.

	Plant	Type	Treatment	conc	uptake
1	Qn1	Quebec	nonchilled	95.00	16.00
2	Qn1	Quebec	nonchilled	175.00	30.40
3	Qn1	Quebec	nonchilled	250.00	34.80
4	Qn1	Quebec	nonchilled	350.00	37.20
5	Qn1	Quebec	nonchilled	500.00	35.30
6	Qn1	Quebec	nonchilled	675.00	39.20

Something Useful I found was that you can use the LaTeX package `makecell` to insert linebreaks in table headers. See the source code for Table 3.2.

Table 3.2: Same table as before but with linebreaks.

Plant Something	Type Yes	Treatment No	conc whatever	uptake ok
Qn1	Quebec	nonchilled	95.00	16.00
Qn1	Quebec	nonchilled	175.00	30.40
Qn1	Quebec	nonchilled	250.00	34.80
Qn1	Quebec	nonchilled	350.00	37.20
Qn1	Quebec	nonchilled	500.00	35.30
Qn1	Quebec	nonchilled	675.00	39.20

Bibliography

[Lichman(2013)] M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

[R Core Team(2015)] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. URL `https://www.R-project.org/`.

[RStudio Team(2015)] RStudio Team. *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA, 2015. URL `http://www.rstudio.com/`.

Appendix A

First Appendix

In case it's needed.