Package as Analysis

Joshua H. Cook

2019-03-10

Contents

Welcome

This is a manual on how to use the standard R package framework for data analysis. Though potentially more work, especially at the start, the purpose of using the R package framework is to maintain a clear and reproducible analysis.

[This book is currently in progress, though any feedback is welcome.]

Why

Why bother with maintaining a package framework while also doing data analysis? It is a great question, especially when one considers the complexity and fluidity of an analysis and the rigidity of the R package framework. The answer is that some rigidity is needed - but just the basics. That is what the R package framework provides. There is a place for everything, though sometimes getting it to work (ie. build and pass checks) requires a few extra steps.

During my own analyses, I found things were getting much too disorganized and decentralized. What would start out as exploratory would morph into a subdirectory graph more complex than the oringinal parent analysis. Perhaps for more organized people, this is unnecessary, but for those of us who want order and aren't sure how to get it, this framework offers a great place to start.

The final (more abstract) reason for using the R package framework is to battle the current issue of reproducibility. Reproducibility is the cornerstone of science - if a finding it true, anyone should be able to replicate it. However, the scientific community has been dealing with an astounding amount of irreproducibility, most famously documented by the Open Science Collaboration. If the analysis is organized as an R package, though, an analysis can be re-run entirely by anyone else familliar with R. Thus, whether they are collaborators and competitors, anyone should be able to follow the analysis a scientist publishes.

Advantages

There are many advantages to using this framework. Here are just a few, though I am sure you will find there are many others:

- Because this is a standard framework, other will be able to navigate the directories and files adeptly.
- The implementation of tests on functions and subroutines will make bugs easier to find and increase overall confidence in the validity of the analysis
- This is a seamless mixture of scripts and markdown files for the separation of functions and analysis
- Complete documentation of functions makes returning to code later much easier!
- The analysis can take advantage of normal R package tools such as Travis-CL and Codecov integration, pkgdown, and devtools (build checks, documentation, etc.).

Examples

[coming soon] Allele-specific *KRAS* copy number alteration

0.1 About

About this Book

[TODO]

About the Author

I am a classically-trained biologist-turned computational biologist. I graduated with degrees in Molecular Biology and Biochemistry, and Chemistry from the University of California, Irvine in 2017. My research focused on investigating the patterns and mechanisms of dissemination by which *Toxoplasma gondii*, an obligate, intracellular parasite, infects a human host Cook *et al.*, 2018. I started my graduate studies at Harvard Medical School in 2018, and after rotating in a chemical biology lab and a *Vibrio cholerae* lab, I finally decided to study cancer using computational biology. Since then, and continuing still today, I have been learning computer programming and statistics, trying to catch up to my peers. Consequently, I have fallen in love with R, especially because of the Tyidyverse and tidy data.

Resources

The best resource for making R packages is R Packages by Hadley Wickham.

There are some useful R packages you will want to have installed:

- `devtools` - will do most of the development building and checking
- `roxygen2` - makes all of the documentation
- `usethis` - for preparing all of the pieces and tools you want to include
- `testthat` - for running tests
- `kintr` - for compiling all of the Rmarkdown files

These can be installed using the following code.

```
install.packages(c("devtools", "roxygen2", "usethis", "testthat", "knitr"))
```

License

Chapter 1

Getting Started

Setting up the package is mostly automated and is well documented in R Packages by Hadley Wickham. If you are running the analysis on your local machine, I would recommend using RStudio (which you likely already do), but this is possible to do on a remote computing cluster (which is how I work). I begin by going through the steps of setting up the basic package framework, which is the same for local or remote work. Following that, there is a section for how I work remotely. This can be skipped if you only work locally or if you already have a system you enjoy (though I highly recommend the system I currently use). I finish off with a few extras that I recommend using, but are not necessary.

1.1 R Package Set Up

1.2 Remote

If you conduct work remotely, I'm going to assume that you have ssh set up and running. Otherwise, there are plenty of resources available, and you should review the material available by your system admin.

Though I prefer Rstudio for normal package development, I spare my computer the pain of performing complex and heavy computation, opting instead to off-load it to the Harvard Medical School Research Computing Cluster. Therefore, I use SublimeText3 as by text editor and send code to the remote comuting node over ssh using iTerm2 as my terrminal. Finally, I use SSH File System (SSHFS) to "mount" the remote directory to my local directory.

1.2.1 SublimeText3 Set-Up

Here are the handful of SublimeText3 (ST3) packages I use for R coding, followed by any particular notes on their use:

- LSP - "Gives Sublime Text 3 rich editing features for languages with Language Server Protocol support"
- MarkdownEditing - "Markdown plugin for Sublime Text. Provides… more robust syntax highlighting and useful Markdown editing features for Sublime Text."
- R-IDE - "[A]iming to utilize the use of language server + better support R Markdown + better support of R packaging + …"
- SendCode - "Send code and text to macOS and Linux Terminals, iTerm, ConEmu, Cmder, Tmux, Terminus; R (RStudio), Julia, IPython."

LSP and R-IDE handle syntax and completion in ST3. It isn't a great system, so if you know of a better set-up in ST3, please let me know. MarkdownEditing and R-IDE combine to make RMarkdown feasible.

The SendCode package essentially copies, pastes, and runs my code written in ST3 to the terminal when I press `command` + `return`. This way, I can type in ST3 and run in the terminal without using the mouse.

Before moving on, I made this snippet to quickly add a code chunk.

```
<snippet>
        <content><![CDATA[
```{r ${1:chunk_name}}
$0
```
]]></content>
        <!-- Optional: Set a tabTrigger to define how to trigger the snippet -->
        <tabTrigger>rchunk</tabTrigger>
        <!-- Optional: Set a scope to limit where the snippet will trigger -->
        <scope>text.html.markdown.multimarkdown, text.html.markdown</scope>
        <description>create a Rmd code chunk</description>
</snippet>
```

1.2.2 Using SSHFS

1.3 Extras

Chapter 2

Framework

Include a subsetion on each main peice of package:

- what is it for
- things to be wary of
- specific use for data analysis

Chapter 3

Example: Allele-specific *KRAS* CNA

[in progress]