Ten simple rules for selecting an R package

Caroline J. Wendt [1,2], G. Brooke Anderson [3*]

1 Department of Statistics, Fort Collins, Colorado, 80523
2 Department of Mathematics, Fort Collins, Colorado, 80523
3 Department of Environmental & Radiological Health Sciences, 1681 Campus Delivery, Fort Collins, Colorado, 80523

* Corresponding author: Brooke.Anderson@colostate.edu

Abstract

Write the abstract here.

Author summary

Write the author summary here.

Text based on plos sample manuscript, see *`http://journals.plos.org/ploscompbiol/s/latex`*

Introduction

Explain what R is and how its package ecosystem works.

Points:

- Open source project, where many people contribute with their own extensions
- Large variation in the quality of different extensions (packages)
- That some R users, particularly new ones, struggle with finding and picking which packages to use.

Ideas of 10 things

Finding packages:

- CRAN task views
- Textbooks ("[x] with R"). May not be latest...
- Google searches, social media (#rstats)
- Conferences (and online streams of those). RStudio, UseR.

Picking a good package:

- On a public repository like CRAN or Bioconductor. Explain more about these repositories and what their standards are. Explain their role in the community. Give the alternative ways that R packages can be shared (GitHub, zipped file posted somewhere else). How these regularly check code and help with managing the web of dependencies.
- Quality of the documentation. Types of documentation (help files, vignette, packagedown website, bookdown book).

- Coverage by tests. Explain about unit testing and how it can help control quality.
- Peer review. ROpenSci. Associated with a peer reviewed paper. Associated with a book put out by a scientific publisher?
- Looking up package authors. Is there role in R development (RStudio, some big bio labs)? Is the work part of their work from an academic lab? Do they have a history of a lot of R development? GitHub profile. Google scholar profile. Also, is it a team of developers? Robust team?
- Evidence of established package. Lots of version. Clear NEWS providing explanations of changes. History of Issues and those being resolved.
- Exploring the code yourself. How open source framework provides this. GitHub mirror of CRAN if you don't want to download the zipped package file yourself.

Here are two sample references: [1,2].

Introduction

R is a language and environment for statistical computing and graphics that was developed by statisticians and is collaboratively maintained by an international core group of contributors. Unlike many popular proprietary languages (e.g., MATLAB, SAS, SPSS), R is highly extensible, free and open-source software; the user can access and thus change, extend, and share code for desired applications. Accordingly, a vibrant community of R users has emerged, many of which engage in the development of extensions to the functionality of base R software known as packages. A prominent contributor in the R community, Hadley Wickham, views functional programming as analogous to following a recipe; to conceptualize packages, imagine R is the kitchen and packages are the special gadgets which allow you to cook and bake new recipes. R packages are coding delectables that enable the user to perform practical tasks (e.g., wrangling and cleaning data frames, designing interactive apps for visualizing data, performing dimensionality reduction) and solve problems (e.g., training regression and classification models, assessing the beta diversity of a population, analyzing gene expression microarray data) with interesting techniques.

As a natural consequence of the open-source nature of R, there is variation in the quality of different packages among the numerous choices that exist. The advanced R user–having developed an intuition for their workflow–may tend to be relatively confident when searching for and selecting packages. By contrast, a common experience that characterizes learning R at the outset is the struggle to 1) find a package to accomplish a particular task and 2) choose the best package to perform that task. Even so, there remain obscure and complicated problems that morph selecting an R package into a barrier despite experience.

In coding as in life, we endeavor to make choices that optimize outcomes. Just as one may go about shopping for shoes, deciding which graduate program to pursue, or conducting a literature review, there is a science behind selection. We inform our decisions by assessing, comparing, and filtering options based on indicators of quality such as utility, association, and reputation. Likewise, choosing an R package requires attending to similar details. We outline ten simple rules for finding and selecting R packages so that you will spend less time searching for the right tools and more time coding delicious recipes.

List of 10 rules

(currently in no particular order and not precisely worded)

1. **Consider your purpose**

- What do you want to use the package to accomplish?
- features
- functions
- organization
- package description
- compare similar options

2. **Spend time searching; find and collect options**

- internet searches (keyword "... in R")
- textbooks ("[x] with R" series)
- tutorials
- courses
- social media (#rstats)
- conferences (e.g., RStudio, useR!)
- consult collaborators
- CRAN task views

3. **Check the repository association**

- purpose: mechanisms of quality control that regularly check code and manage webs of dependencies
- CRAN
- Bioconductor
- GitHub
- collaborators
- zipped file
- alternative ways R packages can be shared

4. **View the documentation**

- help files
- `help()`
- vignettes
- `DOCUMENTATION` file
- "cheatsheets" from RSudio
- RDocumentation (key word search, task views)
- websites (e.g., `packagedown`)
- `bookdown` books
- compare documentation completeness and resource quality

5. **Verify the credibility of the author(s)**

- team or single author (robust team?)
- associations (e.g., academia, industry, labs)
- expertise
- reputation
- experience (e.g., portfolio of packages, history of R development)
- role in R development (e.g., RStudio, regarded bio labs)

- profiles (e.g., GitHub, Google Scholar, Research Gate, Twitter)

6. **Investigate the package development**

- best practices
- unit testing (manage quality control)
- dependencies
- coverage by tests
- number of versions
- clarity of NEWS (explain updates and changes)
- GitHub Issues (history, resolution)

7. **Read, research literature, seek evidence of peer review**

- publications
- package itself
- papers about the package
- ROpenSci
- associations with books or publications from scientific publishers

8. **Quantify how established the package is**

- dependencies
- versions
- updates
- number of downloads
- popularity
- leaderboard
- ranking systems

9. **Put the package to the test**

- explore code
- interact with trial and error
- get a feel for using it in context of your goal
- open-source framework
- GitHub mirror of CRAN as an alternative to downloading zipped package file

10. **Develop your own package**

- necessity
- innovative idea
- novel approach or method
- unique and specialized purpose

References

1. Feynman R, Vernon Jr. F. The theory of a general quantum system interacting with a linear dissipative system. Annals of Physics. 1963;24: 118–173. doi:10.1016/0003-4916(63)90068-X

2. Dirac P. The lorentz transformation and absolute time. Physica. 1953;19: 888–896. doi:10.1016/S0031-8914(53)80099-6