Ten simple rules for selecting an R package

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

1 Department of Statistics, Colorado State University, Fort Collins, Colorado, United States of America
2 Department of Mathematics, Colorado State University, Fort Collins, Colorado, United States of America
3 Department of Environmental & Radiological Health Sciences, Colorado State University, Fort Collins, Colorado, United States of America

* Corresponding author: Brooke.Anderson@colostate.edu

Abstract

R is an increasingly preferred software environment for data analytics and statistical computing among scientists and practitioners. Packages markedly extend R's utility and ameliorate inefficient solutions. We outline ten simple rules for finding relevant packages and determining which is optimal for your desired use.

Author summary

Write the author summary here. Do we want to include and author summary?

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

Disclaimer?

Do we need to include a disclaimer in the margin like the one from [1] that states: "**Competing Interests**: The authors have no affiliation with GitHub, nor with any other commercial entity mentioned in this article. The views described here reflect their own views without input from any third party organization."

- RStudio
- ROpenSci

Funding acknowledgment?

Do we need to include a funding acknowledgment in the margin as in the examples?

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. There are many analogies in computing that draw comparisons between programming and culinary arts: recipe structures, coding cookbooks, and the like. To conceptualize packages, imagine you are the chef, 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 and solve problems with interesting techniques.

Are there R packages for wrangling and cleaning data frames, designing interactive apps for data visualization, or performing dimensionality reduction? Yes! How do you find an R package that will help you train regression and classification models, assess the beta diversity of a population, or analyze gene expression microarray data? The answer is not as simple; there are tens of thousands of R packages. 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 or solve a problem of interest 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.

Rule 1: Consider your purpose

There are often several different ways to accomplish a task or arrive at a solution while programming, albeit some ways are more elegant and efficient than others. While there are certainly ways to cook up an algorithm using for loops and conditionals in base R, a relevant package may accomplish the same goal in a more reproducible manner with less code and fewer bugs. If you define your purpose by making observations about and considering limitations of your current toolbox before you start searching for new tools, you will be more likely to recognize what you do (and don't) need.

Think about what you are doing (or trying to do) and how you might like to do it. Identify the type of inputs you have and envision working with them; contemplate the desired outputs and corresponding format. Which existing functionalities in base R could be improved in context of your problem? Which new functionalities would you like to add to base R to expand what you can do? You may have a unique or particular task that requires highly specialized functions. Some R packages are simple and have a very specific use. For example, there is a package for converting English letters to numbers as on a telephone keypad [2]. On the other hand, you may be faced with an extensive task that justifies a broader framework with several functions that form a cohesive package. Numerous processes that involve data—although varying in application—are ubiquitous. Data manipulation is one such common task that has been streamlined by packages such as `dplyr` and `data.table` [3,4].

In general, the more reasonable it is for your given task to be abstracted away from its context, the more plausible it is that someone has generalized its themes, bundled them up, and branded them in a nice way—often *much* nicer than expected. Nevertheless, there are indeed packages for seemingly singular tasks, which you may be pleasantly surprised to discover. You need not invent the wheel before checking to see if it already exists, but before you sift through options, having some notion of what you need will narrow your search.

Rule 2: Spend time searching; find and collect options

Rule 3: Check how it's shared

Rule 4: Explore the availability and quality of help

Rule 5: Verify the credibility of the author(s)

Rule 6: Investigate the package development

Rule 7: Read, research literature, seek evidence of peer review

Rule 8: Quantify how established the package is

Rule 9: Put the package to the test

Rule 10: Develop your own package

Conclusion

Computational reproducibility is surfacing as a central axiom in academia as researchers identify the need for means by which they can implement transparent systems. As a corollary, former approaches are often found to be at odds with productivity and collaboration. The R programming language is increasingly used by researchers in computational biology and bioinformatics, a discipline among many that is generating extensive heterogenous and complex data that demands sophisticated tools and rigorous methods [5,6].

R packages are a defining feature of the language insofar as many are robust and easily learnable. Packages greatly enhance the user experience and enable you to be more efficient and effective at learning from data regardless of prior experience. However, the sheer quantity and potential complexity of available R packages can undermine their collective benefits. Finding and choosing packages, particularly for beginners, can be daunting and convoluted. R users often struggle to sift through the tools at their disposal and wonder how to distinguish appropriate usage. These ten simple rules for navigating the shared code in the R community are intended to serve as a valuable page in your computing cookbook—one that will evolve into intuition yet remain a reliable reference. May searching for and selecting proper tools no longer spoil your appetite and dissuade you from discovering, trying, creating, and sharing new recipes.

Supporting information

Do we need to include any supporting information?

Acknowledgements

Do we need to include any acknowledgements?

List of 10 rules

(currently in no particular order and not precisely worded)

1. **Consider your purpose**

- features
- functions
- organization
- package description
- compare similar options
- particular task
- general purpose
- improve base R (efficiency, elegance)
- expand base R (do more)
- What do you want to use the package to accomplish?
- What functionalities of base R could be improved?
- What functionalities would you like to add to base R to expand what you can do?
- How do you want to work with inputs and what are the desired outputs?

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

- internet searches (keyword "... in R")
- textbooks ("[x] with R" series)
  - RStudio books
- tutorials
- courses
- social media (#rstats)
- conferences (e.g., RStudio, useR!)
- consult collaborators
- CRAN task views (based on certain disciplines and methodologies)
- Research articles
  - Check which packages have been used in research in your field (provide suggestions for good Google Scholar search queries to identify papers that have used certain packages or that present a package) Alternatively, check the Methods and References sections of papers in your field.
  - Related to that, we could talk about how packages can be cited (the `citation` function produces one in the preferred format for any package). You can look up most packages in Google Scholar to see how many times it's been cited by look-
    ing at the "Cited by" link with the reference. See for example the first listing at https://scholar.google.com/scholar?hl=en&as_sdt=0%2C6&q=dplyr&btnG=
- Blogs
  - find posts with overviews of new packages
  - RStudio packages developed, maintained, or contributed to by the RStudio team categorized by purpose
  - RStudio webinars
  - ROpenSci packages site includes links to blogs, tutorials, and examples for using specific packages
  - R-bloggers
  - Revolutions daily blog

    - R Weekly has specific sections for new packages and updated packages

- People
    - Joe Rickert of RStudio (Ambassador at Large) used to regularly highlight interesting new packages (check to see if he still does).
        - what makes a great R package
        - active on Twitter (`@RStudioJoe`)
    - Mara Averick of RStudio (`tidyverse` developer advocate) advertises cool new R things; check if any focus on packages.
        - I cannot find any recent activity or posts by Mara beyond Twitter (`@dataandme`), but some of her sites have interactive info about cool packages and how to use their features.
        - There are also several videos and posts from her that relate to conferences and unconfs.

3. **Check how it's shared**

- check repository association
    - CRAN
    - Bioconductor
    - GitHub
    - GitLab (alternative to GitHub)
    - ROpenSci (runs its own repository, only includes ones it has peer-reviewed)
    - Self-hosted repositories (can be made with the `drat` package; see paper)
    - purpose of repositories: mechanisms of quality control that regularly check code and manage webs of dependencies
- alternative ways R packages can be shared (not repo)
    - zipped file
    - collaborators

4. **Explore the availability and quality of help**

- 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
- find ways to get help beyond initial documentation
- listservs
- online communities
- Stack Overflow (frequency of questions and answers on the topic)
- See if GitHub repo for the package seems responsive to Issues
- `Rcpp` is an example of high-quality help
    - associated book
    - maintainer, Dirk, is known to be responsive to user questions (listserv)
    - ample documentation including examples to get started

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
- number of citations in Google Scholar

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
- How interoperable it is with other packages that you want to use?
- some packages do what they do really well, but it is hard to use them with the tidyverse or other outside packages
  - S3 or S4 objects that make it hard to work them into a pipeline where their functions are not the last step
- packages that help with interoperability
  - `broom` and `biobroom`: make it easier to put numerous statistical functions into a larger tidyverse workflow

    - Max Kuhn's `caret` package for machine learning—adds a layer that lets you use the same interface to work with machine learning algorithms from lots of different packages that otherwise all have slightly different interfaces for calling the algorithm and working with the results.

10. **Develop your own package**

- necessity
- innovative idea
- standardize
- automate
- novel approach or method
- unique and specialized purpose
- copying and pasting functions repeatedly
- collection of functions you use often
- desire to easily share code, data, documentation, and tests with others
- "personal R packages"
- suggested package development resources for readers:
    - R Packages: Organize, Test, Document, and Share Your Code by Hadley Wickham
    - Ten simple rules for making research software more robust (Taschuk & Wilson)

References

1. Perez-Riverol Y, Gatto L, Wang R, Sachsenberg T, Uszkoreit J, Veiga Leprevost F da, et al. Ten simple rules for taking advantage of git and github. PLoS computational biology. Public Library of Science; 2016;12.

2. Myles S. Phonenumber: Convert letters to numbers and back as on a telephone keypad [Internet]. 2015. Available: `https://CRAN.R-project.org/package=phonenumber`

3. Wickham H, François R, Henry L, Müller K. Dplyr: A grammar of data manipulation [Internet]. 2020. Available: `https://CRAN.R-project.org/package=dplyr`

4. Dowle M, Srinivasan A. Data.table: Extension of 'data.frame' [Internet]. 2019. Available: `https://CRAN.R-project.org/package=data.table`

5. Gentleman RC, Carey VJ, Bates DM, Bolstad B, Dettling M, Dudoit S, et al. Bioconductor: Open software development for computational biology and bioinformatics. Genome biology. Springer; 2004;5: R80.

6. Holmes S, Huber W. Modern statistics for modern biology [Internet]. Cambridge University Press; 2018. Available: `https://web.stanford.edu/class/bios221/book/index.html`