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?

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`