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 several 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 plenty of 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. To optimize your workflow, consider your purpose by 1) identifying your task to understand what you are trying to do and 2) defining the scope of the task to determine how you are going to do it. For some tasks, coding your own recipe with existing tools is practical, while other tasks benefit from new tools.

If the scope of your task is simple or reasonable given your knowledge and skills, using an R package may not be appropriate. There are advantages of coding in base R to implement a unique solution for your problem. In particular, when you code from scratch, you know precisely what you are running; thus, your script may be easier to decipher and maintain. Conversely, packages require reliance on shared code with features of which you may not be aware.

Packages are valuable when a task has a broad scope or is beyond the scope of what you desire to code in base R; a task that is narrow in scope is not necessarily simple. While there are 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. In general, the more reasonable it is for a given task to be abstracted away from its context, the more plausible it is that someone has generalized its themes, developed efficient algorithms, and organized them in an intuitive way to share with other R users. Extensive tasks justify sophisticated frameworks 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 `tidyr` [2,3] (See Table 1). Nevertheless, there are indeed packages for seemingly singular tasks, which you may favor over coding from scratch. Some R packages are small and have a specific use conducive to tasks that require highly specialized functions. For example, there is a package for converting English letters to numbers as on a telephone keypad [4].

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 do not) need. 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? Develop a list of domain-specific keywords that relate what you are trying to do to how one may go about doing it to narrow your search. Identify the type of inputs you have and envision working with them; contemplate the desired outputs and corresponding format. For instance, suppose you are using Bioconductor packages in analyses and have data you would like to visualize; you must consider that the inputs will be of a certain class—namely, S3 or S4 objects—that impose restrictions when creating graphics. The data visualization package you use should address such limitations.

Rule 2: Spend time searching; find and collect options

There are three main categories that distinguish various methods for finding R packages: learning, collaboration, and searching. Firstly, packages are essential to venturing beyond base R and thus quickly become an integral aspect of advancing your R skills. In other words, when learning R, you are typically introduced to some of the most commonly used packages. This does not, however, imply that such packages are necessarily the most basic. Online tutorials, courses, and textbooks tend to be reliable resources for acquiring knowledge about packages that are particularly versatile and reliable. We recommend ——— for interactive learning and ——— textbooks ("[x] with R" series) such as the RStudio books for further reading.

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.

Table 1

```r
library(kableExtra)
library(knitr)
```

```r
# general packages data
gen_pkgs <- data.frame(
  Package = c("readr[note]",
              "dplyr[note]",
              "tidyr",

              "broom[note]",
              "purrr[note]",
              "caret",
              "keras",

              "ggplot2[note]",
              "kableExtra",
              "rmarkdown"),

  Description = c("read rectangular data (e.g., csv, tsv, and fwf)",
                  "grammar of data manipulation",
                  "create tidy data",

                  "tidy model output",
                  "functional programming tools",
                  "train classification and regression models",
                  "R interface to a neural network library",

                  "data visualization",
                  "tables",
                  "reports"),

  Documentation = c("readr",
                    "dplyr",
                    "tidyr",

                    "broom",
                    "purrr",
                    "caret",
                    "keras",

                    "ggplot2",
                    "kableExtra",
                    "rmarkdown")
)
```

```
# general packages table
kable(gen_pkgs, format = "latex", booktabs = TRUE) %>%
  # scale
  kable_styling(latex_options = "scale_down") %>%
  # separate rows by category
  pack_rows("Data Manipulation", 1, 3) %>%
  pack_rows("Statistical Modeling", 4, 7) %>%
  pack_rows("Data Visualization", 8, 10) %>%
  # column wrap
  column_spec(1, width = "10em") %>%
  column_spec(2, width = "20em") %>%
  # bold column names
  row_spec(0, bold = T) %>%
  add_footnote(c("See the tidyverse",
                 "See the tidyverse",
                 "See the biobroom analog in Bioconductor",
                 "See the tidyverse",
                 "See the tidyverse"),
                notation = "symbol")
```

Package	**Description**	**Documentation**
Data Manipulation		
readr[*]	read rectangular data (e.g., csv, tsv, and fwf)	readr
dplyr[†]	grammar of data manipulation	dplyr
tidyr	create tidy data	tidyr
Statistical Modeling		
broom[‡]	tidy model output	broom
purrr[§]	functional programming tools	purrr
caret	train classification and regression models	caret
keras	R interface to a neural network library	keras
Data Visualization		
ggplot2[¶]	data visualization	ggplot2
kableExtra	tables	kableExtra
rmarkdown	reports	rmarkdown

[*] See the tidyverse
[†] See the tidyverse
[‡] See the biobroom analog in Bioconductor
[§] See the tidyverse
[¶] See the tidyverse

```
  # trying to separate color; striped by group
#  row_spec(1:3 - 1, extra_latex_after = "\\rowcolor{gray!6}")
#  row_spec(0:3, extra_latex_after = "\\rowcolor{orange!6}") %>%
#  row_spec(4:6, extra_latex_after = "\\rowcolor{gray!6}") %>%
#  row_spec(7:11, extra_latex_after = "\\rowcolor{gray!6}")

# QUESTIONS
# Code font for package names in " "? \texttt{}?
# How do you repeat same symbol on multiple items with one footnote?
# How do you separate colors and stripe by group?
# Add title?
# Add caption?
# Cite packages in bib and add references in table?
# Embed url link to package documentation?
# How do you add link/reference to Table 1 in text in the template?
# How do you hide code for table in knitted pdf...include=FALSE errors?
# Title for column 2: description/purpose/usage?
# Length of description/purpose/usage for each package?
```

Supporting information

Do we need to include any supporting information?

Acknowledgements

[Acknowledgement of people who have helped]

[Funding acknowledgement]

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

3. Wickham H, Henry L. Tidyr: Tidy messy data [Internet]. 2020. Available: `https://CRAN.R-project.org/package=tidyr`

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

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`