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 [2]. 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. That is, some instances do not warrant additional functions, data, and documentation. 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` [3,4] (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 [5].

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

Those who have used R packages may know that although leveraging existing tools can be advantageous, the initial challenge of finding a suitable package for a given task can obstruct potential benefits. Relatedly, new R users who are unfamiliar with the structure and syntax of the language may be hindered by the process of finding R packages because they do not know where to search, what to look for, nor how to sift through options. R packages are mentioned in a variety of places online, in print, and elsewhere. You can discover new R packages anytime you learn R-related topics, collaborate with other R users, or browse the internet.

Learn

Packages are essential to venturing beyond base R and thus quickly become an integral aspect of advancing your R skills. When learning how to program in R, you are typically introduced to some of the most common packages, which tend to have more general purposes (**Table 1**). In addition, reputable online tutorials, courses, and books are helpful resources for acquiring knowledge about packages that are versatile and reliable—many of which are short, accessible, and either affordable or offered at no cost to the learner. We recommend online R programming courses such as those through Coursera and Codeacademy for interactive learning and R book series including the RStudio books and Springer titles for further reading.

Collaborate

An inclusive and collaborative community is an overlooked, yet integral aspect of a software's success [6]. A defining feature of R is the enthusiasm of its users and contributors alike. The R community has a widespread internet presence across various platforms; however, members are markedly active on Twitter, a place where R users seek help, share ideas, and stay informed on `#rstats` happenings including releases of new packages [7]. Beyond social media, numerous featured pages and R blogs serve as another informal and up-to-date, yet more detailed venue for communicating and promoting R-related information (**Table ?**). Notably, Joseph Rickert, Ambassador at

Large for RStudio, writes monthly posts on the R Views blog highlighting exceptional new R packages in addition to special articles about recently released packages and lists of top packages within certain categories (e.g., Computational Methods, Data, Machine Learning, Medicine, Science, Statistics, Time Series, Utilities, Visualization).

A developer of an R package may intend for it to be private (exclusively for personal or professional use) or public (free and available for use by anyone) [8]. If your task is specific to a line of research, consult colleagues to see if they have relevant (private) code they would be willing to share. Alternatively, literature in your field may either introduce R packages developed to solve a unique data science problem or mention packages used during the research process. The former may be published in the *Journal of Statistical Software*, *The R Journal*, or *BMC Bioinformatics*, for example, and search queries that include `"R package"` along with domain keywords will narrow results. The latter requires identifying authors whom have used R in their analyses, hence packages may be mentioned in the Methods and/or References sections of the article. Accordingly, formatted citations for R packages can be obtained in R with `citation(package = "...")`. You can search for packages directly by name in Google Scholar: the `Cited by` link displays the number of times a package has been cited which connects to a page with those publications. Lastly, conferences are another collaborative environment wherein you can learn about R packages. There are two major annual R conferences, rstudio::conf for industry and useR! for academia; conferences in your field may foster connections with fellow scientists whom use R for similar tasks and help you collect information about packages related to your expertise. Talks and presentations at conferences are often recorded and made available online for playback at a later date.

Browse

Based on prior experience—not unlike solution-seeking for many tasks nowadays—you may think that finding R packages relies heavily on internet search queries. Indeed, search engines such as Google return ample pages related to anything `"...in R"`. However, this approach can lead to frustration and confusion when attempting to find a package for your purpose (see Rule 1). Instead, we recommend initially searching for packages in repositories such as the Comprehensive R Archive Network (CRAN), GitHub, or Bioconductor, all of which will be further discussed in Rule 3. In particular, CRAN Task Views are concentrated topics from certain disciplines and methodologies related to statistical computing that group R packages by the tasks they perform (e.g., Econometrics, Genetics, Optimization, Spatial). In the HTML version, you can browse alphabetized subcategories within each Task View and read concise descriptions to find tools with specific functions. Alternatively, you can access Task Views directly from the R console with `ctv::CRAN.views()`. To date, there are 41 Task Views that collectively contain thousands of packages which are curated and regularly tested. Moreover, CRAN Task Views provide tools that enable the user to automatically install all packages within a targeted area of interest. Ultimately, CRAN Task Views address several major user-end issues that have arisen due to the extensive amount of available packages by providing task-based organization, easy simultaneous installation of related packages, meta-information, ensured maintenance, and quality control [9]. Another place to find well-maintained tools that promote reusable software and reproducibility when working with scientific data in research applications is through rOpenSci packages. Packages are organized by name, maintainer, description, and status (i.e., activity, association, review), can be filtered according to purpose, and searched by name, maintainer, or keywords.

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

Alternative solutions can be sought when a package to solve your data science problem is nonexistent. An R package is the fundamental unit of shareable code; rather than exclusively being a user of packages, you can create them—more easily than you may think [10]. Just as there are numerous R packages for distinct tasks, the reasons why you might want to create a package are abundant: necessity, innovation, standardization, automation, specialty, containment, organization, sharing, collaboration, extensibility, etc.

Whatever your motivation, R packages are simply toolkits; you can create a package out of any collection of specialty functions. Packages need not be formal nor entirely cohesive. For instance, personal R packages (e.g., `Hmisc` and `broman`) are comprised of miscellaneous functions which the creator has developed and frequently uses [11,12]. Functions are necessary for efficiency and warranted when you repetitiously copy and paste your code while making slight modifications after each iteration [13]. The concept of personal R packages demonstrates a unique purpose for packages beyond the conventional. R packages are not solely reserved for specific tasks with comprehensive methods; rather, package development can help you learn how to apply proper coding techniques to writing functions and documentation with reproducibility and collaboration in mind [14].

Although you may not anticipate that anyone else will use your tools, following best practices for package development will yield more favorable outcomes. As a consumer of shared packages, you know the inherent benefits of robust software development relative to the quality of code, data, documentation, versions, and tests [15]. Similarly, creating a valuable package for personal use requires consideration for your future self and anticipation of distributing your code should the need arise. Consider using version control and take advantage of existing resources. Indeed, there are R packages that aid in package development (e.g., `devtools`, `usethis`, `testthat`, `roxygen2`, `rlang`, `drat`). There is no lack of effective organizational frameworks to reference in the open source R community; in fact, repositories for many exemplary packages are available on GitHub. We recommend consulting resources authored by expert R developers including R Packages by Hadley Wickham and Jennifer Bryan as well as the official manual, Writing R Extensions, from CRAN [10,16].

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 [17,18]. It follows that former approaches and traditional methods tend to be at odds with productivity and collaboration; much of the variability in science can be attributed to differences in workflow such that the absence of automation is deemed irresponsible [19]. The open source R language has become the dominant quantitative programming environment in academic statistics, enabling researchers to share workflows and reexecute scripts within and across subsets of the scientific community [19]. R is increasingly used by researchers in computational biology and bioinformatics, a discipline among many that is generating extensive heterogenous and complex data that demands standard tools and rigorous methods that beget reproducibility [20,21]. More broadly, as the R ecosystem—in which the life of modern data analysis thrives—rapidly evolves alongside the burgeoning R community, R is exhibiting sustained growth when compared to similar languages, particularly in academia, healthcare, and government [22].

R packages are a defining feature of the language insofar as many are robust and easily learnable. Some of the most prominent R packages are a result of the developer abstracting common elements of a data science problem into a workflow that can be shared and accompanied by thorough descriptions of the process and purpose. In this way, R packages have effectively transformed how we interact with data in the modern day in, perhaps, a more impactful manner than many revered contributions to theoretical statistics [19]. 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 (general packages)

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

```
# 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"),

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

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

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

Author = c("readr Wickham et al.",
           "dplyr Wickham et al.",
           "tidyr Wickham et al.",

           "broom Robinson et al.",
           "purrr Henry et al.",
           "caret Kuhn et al.",
           "keras Falbel et al.",

           "ggplot2 Wickham et al.",
           "kableExtra Zhu et al.",
           "rmarkdown Allaire et al."),

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

                  "broom",
                  "purrr",
                  "https://topepo.github.io/caret/index.html",
                  "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	Year	Author	Documentation
Data Manipulation				
readr*	read rectangular data (e.g., csv, tsv, and fwf)	readr	readr Wickham et al.	readr
dplyr†	grammar of data manipulation	dplyr	dplyr Wickham et al.	dplyr
tidyr	create tidy data	tidyr	tidyr Wickham et al.	tidyr
Statistical Modeling				
broom‡	tidy model output	broom	broom Robinson et al.	broom
purrr§	functional programming tools	purrr	purrr Henry et al.	purrr
caret	train classification and regression models	caret	caret Kuhn et al.	https://topepo.github.io/caret/index.html
keras	R interface to a neural network library	keras	keras Falbel et al.	keras
Data Visualization				
ggplot2¶	data visualization	ggplot2	ggplot2 Wickham et al.	ggplot2
kableExtra	tables	kableExtra	kableExtra Zhu et al.	kableExtra
rmarkdown	reports	rmarkdown	rmarkdown Allaire et al.	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? Do we want to link cheatsheets?
```

```
# 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. Team RC. The r project for statistical computing [Internet]. The R Foundation; 2020. Available: https://www.r-project.org/

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

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

6. Smith D. The r community is one of r's best features [Internet]. Revolutions. Microsoft; 2017. Available: https://blog.revolutionanalytics.com/2017/06/r-community.html

7. Ellis SE. Hey! You there! You are welcome here [Internet]. rOpenSci. NumFOCUS; 2017. Available: https://ropensci.org/blog/2017/06/23/community/

8. Rickert J. What makes a great r package? [Internet]. RStudio; 2018. Available: https://rstudio.com/resources/rstudioconf-2018/what-makes-a-great-r-package-joseph-rickert/

9. Zeileis A. CRAN task views. R News. 2005;5: 39–40.

10. Wickham H. R packages: Organize, test, document, and share your code. "O'Reilly Media, Inc."; 2015.

11. Harrell Jr FE, Charles Dupont, others. Hmisc: Harrell miscellaneous [Internet]. 2020. Available: https://CRAN.R-project.org/package=Hmisc

12. Broman KW. Broman: Karl broman's r code [Internet]. 2020. Available: https://CRAN.R-project.org/package=broman

13. Wickham H. Advanced r. CRC press; 2014.

14. Parker H. Personal r packages [Internet]. 2013. Available: https://hilaryparker.com/2013/04/03/personal-r-packages/

15. Taschuk M, Wilson G. Ten simple rules for making research software more robust. PLoS computational biology. Public Library of Science; 2017;13.

16. Team RC. Writing r extensions [Internet]. The R Foundation; 2020. Available: https://cran.r-project.org/doc/manuals/R-exts.html

17. Peng RD. Reproducible research in computational science. Science. American Association for the Advancement of Science; 2011;334: 1226–1227.

18. Goodman SN, Fanelli D, Ioannidis JP. What does research reproducibility mean? Science translational medicine. American Association for the Advancement of Science; 2016;8: 341ps12–341ps12.

19. Donoho D. 50 years of data science. Journal of Computational and Graphical Statistics. Taylor & Francis; 2017;26: 745–766.

20. 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.

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

22. Robinson D. The impressive growth of r [Internet]. Stack Overflow; 2017. Available: https://stackoverflow.blog/2017/10/10/impressive-growth-r/