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
- GitHub

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, datasets, documentation, or other objects. 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, S4 objects—which 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 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 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. Rickert also features 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 useful 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 and 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 tailored to 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]. Relatedly, CRANberries is a hub of information about new, updated, and removed packages from the CRAN network. Another place to find well-maintained tools is through rOpenSci packages. These R 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

Packages can be shared through a variety of platforms; a single package is often available in multiple places. While repositories are the primary way in which developers share packages for both public and private use, there are alternatives. Usually in lieu of making packages accessible to everyone via internet repositories, some developers share their code in zipped files or directly with collaborators. As far as R packages are concerned, a repository is essentially a warehouse for tools before they enter your kitchen; in computing terms, a repository is analogous to a cloud because it is a central location in which data is stored and managed. There is growing concern about whether there are too many R packages and a simultaneous call for quality control [10]. Some repositories impose vetting mechanisms that tame unwieldy aspects of the R ecosystem such as those that regularly check the code that underlies packages and manage corresponding webs of dependencies. The traditional repositories for R packages are CRAN, Bioconductor, and GitHub; however, there are lesser-known remote repositories that have unique properties.

A plurality of what we know about statistical methods and algorithms is wrapped up in R packages—written and documented in various ways by R users. The CRAN package repository is the most established and thereby main source from which you can install R packages. You can simply use the `install.packages` function to install a package from CRAN which is automatically saved to your computer in a designated package library. When you would like to use a particular package, you load it to your R session via the `library` function from base R. Due to its longevity and historical role, Rickert asserts that "CRAN is the greatest open-source repository for statistical computing knowledge in the world" [8]. Evidently, much of what we know about statistics is housed in CRAN where people share such information. The R Foundation manages CRAN and imposes strict regulatory practices for the selection and maintenance of the packages they host. A package must pass a series of stability tests in accordance with the CRAN Repository Policy before obtaining publication privileges [11]. As such, CRAN only considers packages that make a substantial contribution to statistical computing and graphics. CRAN maintainers actively monitor source and contributed packages to ensure they are compatible with the latest version of R and modify or remove packages that do not uphold publication quality.

The Bioconductor project was motivated by a need for transparent, reproducible, and efficient software in computational biology and bioinformatics to integrate computational rigor and reproducibility with research on biological processes [12]. The R environment and its package system is fundamental to the implementation of Bioconductor's interoperable and object-oriented (S4) infrastructure. Bioconductor software is in the form of coordinated, peer reviewed R packages. More specifically, Bioconductor boasts a modularized design wherein data structures, functions, and the packages that contain them have distinct roles that are accompanied by thorough documentation. Accessibility is pillar of the Bioconductor project, thus all forms of documentation (e.g., courses, vignettes, interactive documents) are curated for individuals with expertise in adjacent disciplines and minimal experience with R (see Rule 4) [12]. Similar to CRAN, Bioconductor has strict criteria for package submissions: a package must be relevant to high-throughput genomic analysis, interoperable with other Bioconductor packages, well documented, supported in the long-term, exclusive to Bioconductor, and comply with additional package guidelines [13]. Bioconductor packages facilitate the analysis and comprehension of biological data and help users solve problems that arise when working with high-throughput genomic data such as those related to microarrays, sequencing, flow cytometry, mass spectrometry, and image analysis. As a subset of the R community, Bioconductor has a supportive and innovative community which hosts annual meetings and conferences.

The rapid uptick in package development and subsequent inter-repository dependencies has sparked an ongoing debate as to whether regulated repositories—including the aforementioned CRAN and Bioconductor—are preferable to other distribution platforms, namely public version control systems such as GitHub [8,14][15]. While there are practical downsides to their restrictive practices, the benefits of exclusive repositories are evident. Nevertheless, there are considerable upsides to hosting a package on GitHub [8,14]. GitHub is a popular multi-purpose development platform (whereas CRAN and Bioconductor are not) that is also effective for distributing R packages. An increasing number of packages are hosted on GitHub during the development stages; if developers choose not to distribute their package through GitHub, the stable release versions of such packages are often published on CRAN or Bioconductor [16]. GitHub grants R users open-access to package code, a timeline of help resources (see Rule 4), a direct line of communication to developers, and permits discovery of up-and-coming packages (see Rule 2). You can install the latest version of packages from GitHub via `devtools::install_github`; however, the decentralized nature of GitHub is not conducive to a tool that automatically locates and installs corresponding dependencies [17]. For developers, GitHub provides a convenient means by which anyone can share and contribute public or private code without barriers to entry. Authors collaborate within a version-controlled system to develop and distribute packages including those with dependencies that are not on CRAN or Bioconductor. Further, the `drat` package enables developers to design individual repositories and suites of coordinated repositories for packages that are stored in and/or distributed through GitHub [18]. Both R users and package developers benefit from interactive feedback channels through GitHub Issues and the Star rating system.

Most novice R users will rarely encounter packages that are not shared through the abovementioned platforms. rOpenSci runs a repository that promotes reusable software and reproducibility when working with scientific data in research applications [19]. The repository only includes packages that have undergone their open review process that is harmonious with GitHub infrastructure. Further, GitLab is git-based version control and collaborative cloud for package production and deployment. It is an alternative to GitHub for production of large-scale packages that require continuous integration and continuous deployment for testing data and code to ensure a stable end product. There are platforms strictly for the development, rather than distribution, of R packages such as R-Forge and Omegahat, which are beyond our scope [20].

Rule 4: Explore the availability and quality of help

There has been a call for the development of centralized resources in statistical computing: those that enable a common understanding of software quality and reliability (i.e., software information specified in publications and domain-specific semantic resources) and a single metadata resource for statistical software [10]. No such resources have been consolidated to serve these purposes and given the decentralized nature of today's information society, it is questionable whether they will emerge. Current sources of information related to R packages are dispersed and seemingly boundless in quantity. On one hand, this allows users to explore diverse solutions and discover new tools; on the other, not knowing where to find help can lead to inefficient and ineffectual roundabouts. Clearly, not all package resources share the same level of quality and the fact that there are many resources in aggregate, does not imply that every package is associated with the same availability of resources. While all R packages warrant some minimal standard of documentation, beginners and users of complex packages merit more exhaustive information.

You can access information about R packages along with an index of help pages from

the console via `help(package = "...")`. Package information will vary; ideally, packages should have thorough documentation, but at minimum, every R package should include a `DESCRIPTION` file with metadata. The `DESCRIPTION` is a succinct record of the package's purpose, dependencies, version, date, license, associations, authors, and other technical details. The help pages feature information about the structure of functions within the package and contain executable examples to demonstrate the relationship between various inputs and outputs. If the `DESCRIPTION` and help pages alone leave you wanting, the package likely does not have further (quality) documentation and therefore should not be your first choice, if comparable options exist. In short, if the developer cannot initially communicate how their tool works, then you may not want to use it in your kitchen.

Fortunately, plenty of R packages include additional documentation beyond mere descriptions. The documentation that accompanies functions within packages is critical; the fact that someone can read the documentation at a later date and use it in their own work enables extensibility. RDocumentation is a searchable website, package (`install.packages("RDocumentation")`), and JSON API for obtaining integrated documentation for packages that are on CRAN, Bioconductor, and GitHub. This is a subsidiary reason why packages shared on these platforms tend to be superior (see Rule 3). RDocumentation may include: an overview, installation instructions, examples of usage, functions, guides, and vignettes. Most software documentation is rather technical and extraneous to new users whereas a vignette is a practical type of documentation in the form of a tutorial. A vignette is a detailed, long-form document that describes the problems an R package can solve, then illustrates applications through clear examples of code with coordination of functions and explanations of outcomes. Packages can have multiple vignettes; you can view or edit a specific vignette or obtain a list of all vignettes for a package of interest via the `vignette` function.

Some packages are branded quite well and include a comprehensive and cohesive set resources. Implicitly, this indicates that the authors are at least serious about their package development, which may lead you to infer that they know what they (and their package) are doing. Exemplar documentation that extends far beyond the minimum signifies an exceptional package. For instance, some packages have websites and/or books. One popular method that developers use to publish books about their package is through `bookdown`, a relatively new extension of R Markdown that is structured in such a way that integrates code, text, links, graphics, videos, and other content in a format that can be published as a free, open, interactive, and downloadable online book [21]. As a convenient example, the `bookdown` package itself has an online book that details usage of the package [22]. For reference, the `Rccp` is another package with notable documentation and first-rate help resources; the developers maintain both a main and additional website with a wealth of organized information about the package and resources including: examples, associations, publications, articles, blogs, code, books, talks, a mailing list, and links to other resources with `Rccp` tags. Aside from the documentation and resources from the developer, further information about some R packages is sometimes available in video tutorials, webinars, and code demonstrations (i.e., "demos"). Finally, keep in mind that RStudio creates Cheatsheets that provide concise usage information for popular packages through code and graphics organized by purpose. Cheatsheets can be accessed directly via the RStudio Menu (Help > Cheatsheets) or from the RStudio website on which you can subscribe to Cheatsheet updates and find translated versions.

Anticipate that you may run into complications beyond the scope of documentation while using a package. In which case, you will use resources that involve *asking* for help—should the occasion arise, you want to be assured that you will find a (satisfactory) answer. It was formerly the case that the only way to seek assistance was

through the discouraging R-help mailing list; since, the legendary R community has formed, with inclusion and creative problem solving as hallmarks of its presence online [23]. The modern R-help mailing list to which you can subscribe and send questions is moderated by the R Core Development Team and includes additional facets for major announcements about the development of R and availability of new code (R-announce) and new or enhanced contributed packages (R-packages) [24]. Certain packages have independent listservs; `statnet` is an example of a suite of packages that has its own community listserv. If a package has a development repository on GitHub, check the Issues to verify that the maintainer is responsive to posts and fixes bugs in a timely manner. In addition, you can search discussion forums such as Stack Overflow, Cross Validated, and Talk Stats to assess the activity associated with the package in question. Analyses of the popularity of comparable data analysis software in email and discussion traffic suggest that R is rapidly becoming more prevalent and is the leading language by these metrics [25,26]. When you encounter a problem, it is good practice to first update the package to see if the problem is due to a bug in a previous version—if the problem persists, seek help by finding or posting a reproducible example [27]. Overall, avoid using a package if the collective quality and quantity of resources is lacking.

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

Although R is grounded in statistical computing and graphics, there is variation in what people use R for, more variation in the skills of the people who use R, and thus even more variation in the extensions that people create for R. Just as research is a library of shared insight, open source software is an amalgam of shared tools. We care about who writes the articles we read, so too should we care about who creates the tools we use. Associations and reputation are often a proxy for quality; in this way, the process of evaluating and comparing R packages is no different than other decisions. In fact, as you become more immersed in the R community, you will find that name recognition is a crucial factor that determines why you blindly trust certain tools and hesitate to use others [28].

You can assess the credibility of R package developers through direct and indirect signals. Who made the package? Consider whether expertise in a certain domain is vital to the design and creation of the tool. Research the authors' associations (e.g., academia, industry, laboratories) and gauge the extent to which they have a primary role in R development through, for example, RStudio or esteemed biology laboratories. Further, you can learn more about their experience, active contributions to the R community, and history related to package development by exploring their profiles (e.g., GitHub, Google Scholar, Research Gate, Twitter, personal or package websites). If an author has such a history, peruse their portfolio of packages to see if any are highly regarded or recognizable. Frequent commits and effective resolutions of Issues for packages hosted on GitHub reveal the authors' commitment. If the package was developed by multiple authors, research each of them to evaluate the robustness of the team. By extension, these indicators of developer involvement and reputation will help you discern whether a package is worthy of your trust and time.

Rule 6: Investigate the package development

You need not be a software engineer to identify strong package development. Scientific software developers do not always adhere to best practices; indeed, these shortcomings are evident in the tools they create [29]. There are concrete ways to measure a tool's robustness beyond whether or not it works for those who did not create it. R packages

often depend on other R packages; you should check the reputations of such *dependencies* when selecting a package—quality packages will likely rely on a solid web of quality packages. What's more, like other types of software, well-maintained R packages have multiple versions corresponding to iterative releases to indicate that the package is compatible with dependencies and loyally updated (e.g., bug fixes, general improvements, new functionality) [1,30]. You can explore the version history of a package to see if it is up-to-date. As a user, there are two additional development protocols that you can further investigate to assess the underlying stability and utility of a package: unit tests and version control.

A responsible developer with a consistent and reproducible workflow will implement formal testing on their code to examine expected behavior via an automated process called unit testing [30,31]. Although inconvenient at the outset, the developer—and by extension, the package user—will benefit from unit testing, which results in fewer bugs, a well-designed code structure, an efficient workflow, and robust code that is not sensitive to major changes in the future [30]. To alleviate the burdens of unit testing, `testthat` is a popular, integrative R package that helps developers create reliable functions, minimize error, and visualize progress through automatic code testing [32]. Developers are also interested in quantifying the amount of code in their package that has been tested. Test coverage, a measurement of the proportion of code that has undergone unit testing, is an objective a metric for package developers, contributors, and users to evaluate code quality. Many developers use the `covr` package to generate reports and determine the magnitude of coverage on the function, script, and package levels [33]. Relatedly, developers who host their packages on GitHub, post status badges in the overview (`README`) section of the repository webpage. GitHub badges are a common self-imposed method to signal use of best practices and motivate developers to produce a product that is high in quality and transparency [34]. You may see, for example, license, dependency, or style badges, all of which are good indicators of package caliber; however, particular to this Rule, you should look for code coverage (`codecov`) badges which reveal the percentage of test coverage.

As we mentioned in Rule 3, version control has an essential role in package development and computational literacy more broadly [30,35]. Version control is like a time capsule for your workflow because it monitors and tracks changes to files as a project evolves, and stores them as previous versions to be recovered if necessary. In other words, "version control is as fundamental to programming as accurate notes about lab procedures are to experimental science" [35]. Git is a decentralized open-source version control system that is useful regardless of whether a project is independent or collaborative [36]. GitHub is an online user interface that functions as a collaborative platform for sharing and improving code. GitHub works in conjunction with Git to provide a powerful structured system to organize and manage components of a project for others and your future self. A growing number of scientists have research programs based in GitHub (e.g., laboratory repositories, large bioinformatics projects), which has become a revolutionary tool for productive team science and distributed development efforts [1,37]. As you may expect, Git coupled with GitHub is the version control duo of choice among serious R package developers [30]. Thus, if the package you are interested in using is among the thousands hosted on GitHub, this is evidence that the developer is at least committed to a logical, open, and reproducible workflow, suggestive of more time spent designing their tool.

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

Peer review is an important aspect of scientific research, not least because it establishes scholarly credibility. As a corollary, you can research information about an R package in different forms of literature and hence determine the extent to which it has been validated by the scientific community. Some journals publish articles about R packages themselves while others feature work that has used a particular package (see Rule 2). Clearly, these packages are technically sound and have made a substantial contribution to their fields and/or a common data science problem. Accordingly, in response to the rising number of researchers creating tools and software to work with their data, GitHub has granted developers the ability to obtain a Digital Object Identifier (DOI) for any GitHub repository archive so that code can be cited in academic literature [38]. If a package has such a DOI, you can explore the network of research associated with that package. Many R packages are associated with content in books and series from scientific publishers such as Springer. More directly, rOpenSci, a non-profit organization committed to promoting open science practices through technical and social infrastructure for the R community, is a unique example of an ecosystem of open-source tools with peer reviewed R packages (See Rule 3) [19].

Rule 8: Quantify how established the package is

When making comparisons, consulting the data is never a bad idea; numerical data associated with R packages will give you an impression of how regarded the tool is and whether or not it has stood the test of time. Since there are tens of thousands of R packages, you may be wondering how they stack up in terms of popularity. On GitHub, a large number of Stars, Forks, and Watchers associated with a package implies a substantial following and widespread usage [28]. Likewise, the number of Google Scholar citations is an explicit metric of a package's impact on scientific research and utility in research contexts (see Rule 2). RDocumentation (see Rule 4) is rich with stats on R packages. RDocumentation hosts a live Leaderboard with trends including the number of indexed packages and indexed functions, most downloaded packages, most active maintainers, newest packages, and newest updates. What's more, each package is assigned a percentile rank—featured on its RDocumentation page—that quantifies the number of times a package has been downloaded in a given month. A ranking algorithm computes the direct, user-requested monthly downloads by accounting for reverse dependencies (indirect downloads) so packages that are commonly depended upon, and hence frequently downloaded, do not skew the calculation [39]. You can research stats on corresponding dependencies for a more holistic picture. To further determine if a package is well-established in the R community, refer to the number of versions and updates (more is better) as well as the date of the most recent versions and updates (newer is better).

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 [30]. 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. The essence of an R package is that it is a self-contained piece of statistical knowledge that can be used in combination with other self-contained pieces of statistical knowledge of different shapes and sizes; the uniquely structured functions within a package help us implement that knowledge and weave it into novel scientific work.

Whatever your motivation, 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 [40,41]. Functions are necessary for efficiency and warranted when you repetitiously copy and paste your code while making slight modifications after each iteration [27]. 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 [42].

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 [29]. 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`) [17,43][44][45][46][18]. In the case of collaboration, the R project is compatible with distributed development—a feature that couples well with version control. 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 [30,47].

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 [48,49]. 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 [50]. 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 [50]. 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 [12,51]. 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 [25].

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 [50]. 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. Hornik K. Are there too many r packages? Austrian Journal of Statistics. 2012;41: 59–66.

11. CRAN repository policy [Internet]. The R Foundation; 2020. Available: https://cran.r-project.org/web/packages/policies.html#Submission

12. 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.

13. Package submission [Internet]. Bioconductor; 2020. Available: https://www.bioconductor.org/developers/package-submission/

14. McElreath R. Statistical rethinking: A bayesian course with examples in r and stan. CRC press; 2020.

15. Decan A, Mens T, Claes M, Grosjean P. When github meets cran: An analysis of inter-repository package dependency problems. 2016 ieee 23rd international conference on software analysis, evolution, and reengineering (saner). IEEE; 2016. pp. 493–504.

16. Decan A, Mens T, Claes M, Grosjean P. On the development and distribution of r packages: An empirical analysis of the r ecosystem. Proceedings of the 2015 european conference on software architecture workshops. 2015. pp. 1–6.

17. Wickham H, Hester J, Chang W. Devtools: Tools to make developing r packages easier [Internet]. 2020. Available: https://CRAN.R-project.org/package=devtools

18. Carl Boettiger DE with contributions by, Fultz N, Gibb S, Gillespie C, Górecki J, Jones M, et al. Drat: 'Drat' r archive template [Internet]. 2020. Available: https://CRAN.R-project.org/package=drat

19. Transforming science through open data and software [Internet]. rOpenSci; 2020. Available: https://ropensci.org/

20. Theußl S, Zeileis A. Collaborative software development using r-forge. Special invited paper on" the future of r". The R Journal. The R Foundation for Statistical Computing; 2009;1: 9–14.

21. Xie Y. Bookdown: Authoring books and technical documents with r markdown [Internet]. 2020. Available: https://CRAN.R-project.org/package=bookdown

22. Xie Y. Bookdown: Authoring books and technical documents with r markdown. Chapman; Hall/CRC; 2016.

23. Chase W. Dataviz and the 20th anniversary of r, an interview with hadley wickham [Internet]. Medium; 2020. Available: https://medium.com/nightingale/dataviz-and-the-20th-anniversary-of-r-an-interview-with-hadley-wickham-ea24507

24. Team RC. Mailing lists [Internet]. The R Foundation; 2020. Available: https://www.r-project.org/mail.html

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

26. Muenchen RA. The popularity of data analysis software. URL http://r4statscom/popularity. 2012;

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

28. Leek J. How i decide when to trust an r package [Internet]. 2015. Available: https://simplystatistics.org/2015/11/06/how-i-decide-when-to-trust-an-r-package/

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

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

31. Hester J. How does covr work anyway? [Internet]. The R Foundation; 2020. Available: https://cran.r-project.org/web/packages/covr/vignettes/how_it_works.html

32. Wickham H. Testthat: Get started with testing. The R Journal. 2011;3: 5–10. Available: https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf

33. Hester J. Covr: Test coverage for packages [Internet]. 2020. Available: https://CRAN.R-project.org/package=covr

34. Barts C. How to use github badges to stop feeling like a noob [Internet]. freeCodeCamp; 2018. Available: https://www.freecodecamp.org/news/how-to-use-badges-to-stop-feeling-like-a-noob-d4e6600d37d2/

35. Wilson GV. Where's the real bottleneck in scientific computing? American Scientist. 2006;94: 5.

36. Bryan J. Excuse me, do you have a moment to talk about version control? The American Statistician. Taylor & Francis; 2018;72: 20–27.

37. Perkel J. Democratic databases: Science on github. Nature News. 2016;538: 127.

38. Smith A. Improving github for science [Internet]. GitHub, Inc. 2014. Available: https://github.blog/2014-05-14-improving-github-for-science/

39. Vannoorenberghe L. RDocumentation: Scoring and ranking [Internet]. DataCamp; 2017. Available: https://www.datacamp.com/community/blog/rdocumentation-ranking-scoring

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

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

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

43. Wickham H, Bryan J. Usethis: Automate package and project setup [Internet]. 2019. Available: https://CRAN.R-project.org/package=usethis

44. Wickham H. Testthat: Get started with testing. The R Journal. 2011;3: 5–10. Available: https:

//journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf

45. Wickham H, Danenberg P, Csárdi G, Eugster M. Roxygen2: In-line documentation for r [Internet]. 2020. Available: https://CRAN.R-project.org/package=roxygen2

46. Henry L, Wickham H. Rlang: Functions for base types and core r and 'tidyverse' features [Internet]. 2020. Available: https://CRAN.R-project.org/package=rlang

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

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

49. 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.

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

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