Creating optimal conditions for reproducible data analysis in R with `fertile`

Audrey Bertin *
Program in Statistical and Data Sciences, Smith College
and
Benjamin S. Baumer
Program in Statistical and Data Sciences, Smith College

March 16, 2020

Abstract

The advancement of scientific knowledge increasingly depends on ensuring that data-driven research is reproducible: that two people with the same data obtain the same results. However, while the necessity of reproducibility is clear, there are significant behavioral and technical challenges that impede its widespread implementation, and no clear consensus on standards of what constitutes reproducibility in published research. We focus on a series of common mistakes programmers make while conducting data science projects in R, primarily through the RStudio integrated development environment. `fertile` is an R package that operates in two modes: proactively (to prevent reproducibility mistakes from happening in the first place), and retroactively (analyzing code that is already written for potential problems). Furthermore, `fertile` is designed to educate the user about why the mistakes are problematic, and how to fix them. We discuss experimental results from testing `fertile` in an introductory data science course.

Keywords: reproducibility, statistical software, workflow, collaboration

*The authors gratefully acknowledge contributions from Hadley Wickham, Jenny Bryan, Greg Wilson, Edgar Ruiz, and other members of the `tidyverse` team.

1 Introduction

As research is becoming increasingly data-driven, and because knowledge can be shared worldwide so rapidly, reproducibility is critical to the advancement of scientific knowledge.

Data-based research cannot be fully *reproducible* unless the requisite code and data files produce identical results when run by another analyst. When researchers provide the code and data used for their work in a well-organized and reproducible format, readers are more easily able to determine the veracity of any findings by following the steps from raw data to conclusions.

The creators of reproducible research can more easily receive more specific feedback (including bug fixes) on their work. Moreover, others interested in the research topic can use the code to apply the methods and ideas used in one project to their own work with minimal effort.

However, while the necessity of reproducibility is clear, there are significant behavioral and technical challenges that impede its widespread implementation, and no clear consensus on standards of what constitutes reproducibility in published research (Peng (2009)). Not only are the *components* of reproducible research up for discussion (e.g., need the software be open source?), but the corresponding *recommendations* for ensuring reproducibility also vary (e.g., should raw and processed data files be in separate directories?).

Much of the discussion around reproducibility is also generalized—it is written to be applicable to users working with a variety of statistical software programs. Since all statistical software programs operate differently, generalized recommendations on reproducibility are often shallow and unspecific. While they provide useful guidelines, they can often be difficult to implement, particularly to new analysts who are unsure how to apply such recommendations within the software programs they are using. Thus, reproducibility recommendations tailored to specific software programs are more likely to be adopted.

In this paper, we focus on reproducibility in the R programming language with a concentration on projects that use the RStudio integrated development environment. R is an ideal candidate for reproducibility recommendations due to the language's popularity for statistical analyses and the ease with which analysts can download and begin using the software.

A small body of papers and R packages focusing on reproducibility in the R community have been published.

Much of this work is narrowly tailored, with each package effectively addressising a small component of reproducibility—file structure, modularization of code, version control, etc. Many existing reproducibility packages, due to their focused nature, succeed at their area of focus, but at a cost. They are often difficult to learn and operate, providing a barrier to entry for less-experienced data analysts.

There appear to be no existing tools that cater to analysts looking for an easy-to-learn, easy-to-implement, fast, way to obtain a broad overview of their projects' reproducibility. To address this, we present an R package called **fertile** [1], a low barrier-to-entry package which focuses on a series of common mistakes programmers make while conducting data science research in R.

2 Literature Review

Reproducibility is critically important to the advancement of knowledge in all fields of scientific research. Researchers are acknowledging this; publications and discussions focusing on reproducibility seem to have increased in frequency over the last several years (Fidler & Wilcox 2018, Wallach et al. (2018), Gosselin (2020), Eisner (2018), McArthur (2019)).

Much of the available literature is focused on the methods for achieving reproducibility in specific disciplines, though much of this information is generalizable to all areas of scientific research. There is also work focusing specifically on reproducibility within the R community. Some of this work is in the form of academic papers, while other literature comes from blog posts, websites, and R packages, some of which are compiled by notable R-focused developers and researchers. In this section, we will discuss the reproducibility recommendations from both the formal and informal publications, which together influence the development of **fertile**.

[1]The authors gratefully acknowledge contributions from Hadley Wickham, Jenny Bryan, Greg Wilson, Edgar Ruiz, and other members of the **tidyverse** team.

2.1 Previous Work

Goodman et al. (2016) argues that the language and conceptual framework of research reproducibility varies across the sciences. There are no clear standards agreed upon across fields.

Kitzes et al. (2017) presents a collection of case studies on reproducibility practices from across the data-intensive sciences, illustrating a variety of recommendations and techniques for achieving reproducibility. Although the book does not come to a consensus on the exact standards of reproducibility that should be followed, several common trends and principles emerge from the case studies:

1. Use clear separation, labeling, and documentation.
2. Automate processes when possible.
3. Design the data analysis workflow as a sequence of small steps glued together, with outputs from one step serving as inputs into the next. This is a common suggestion within the computing community, originating as part of the Unix philosophy (Gancarz (2002)).

Cooper et al. (2017) focuses on data analysis in R and identifies similar important reproducibility components, while reiterating the need for clearly labeled, well-documented, and well-separated files. In addition, they recommend publishing a list of dependencies and using version control.

Broman reiterates the need for clear naming and file separation while sharing several additional suggestions: keep the project contained in one directory, use relative paths, and include a README (Broman (n.d.)).

The reproducibility recommendations from R OpenSci, a non-profit initiative founded in 2011 to make scientific data retrieval reproducible, share similar principles to those discussed previously. They focus on a need for a well-developed file system, with no extraneous files and clear labeling. They also reiterate the need to note dependencies and use automation when possible, while making clear a suggestion not present in the previously-discussed literature: the need to use seeds, which allow for the saving and restoring of the random number generator state, when running code involving randomness *Reproducibility in Science* (n.d.).

When considered in combination, these sources provide a well-rounded picture of the components important to research reproducibility. Using this literature as a guideline, we identify several key features of reproducible work to focus on. These recommendations are opinionated—due to the lack of agreement on which components of reproducibility are most important, we select those that are mentioned most often, as well as some that are mentioned less but that we view as important.

1. A well-designed file structure:

- Separate folders for different file types.
- No extraneous files.
- Minimal clutter.

2. Good documentation:

- Files are clearly named, preferably in a way where the order in which they should be run is clear.
- A README is present.
- Dependencies are noted.

3. Reproducible file paths:

- No absolute paths, or paths leading to locations outside of a project's directory, are used in code. Only portable (relative) paths.

4. Randomness is accounted for:

- If randomness is used in code, a seed must also be set.

5. Code conforms to tidyverse style:

- Although not discussed in any of the literature mentioned previously, we also believe **tidyverse** integration to be very important. Code conformation to **tidyverse** style helps ensure that analyses meeting **fertile**'s recommendations will be able to integrate seamlessly with packages from the **tidyverse**, an ever-expanding collection of packages designed to operate together with one another.

Much of the available literature focuses on file structure, organization, and naming, and **fertile**'s features are consistent with this. The *ideal* file structure is not agreed upon by academics, though there are some recommendations focused on R that are available. Marwick et al. (2018) provides the framework for file structure that **fertile** is based on: a structure similar to that of an R package (Wickham (2015), R-Core-Team (2020)), with an `R` folder, as well as `data`, `data-raw`, `inst` and `vignettes`.

2.2 R Packages

A small selection of R packages work to address the issue of research reproducibility, although not all of them are available on `CRAN`.

rrtools (Marwick 2019) addresses some of the issues discussed in Marwick et al. (2018) by creating a basic R package structure for a data analysis project and implementing a basic **testthat**::`check` functionality. The **orderly** (FitzJohn et al. 2020) package also focuses on file structure, requiring the user to declare a desired project structure (typically a step-by-step structure, where outputs from one step are inputs into the next) at the beginning and then creating the files necessary to achieve that structure. **workflowr**'s (Blischak et al. 2019) functionality is based around version control and making code easily available online. It works to generate a website containing time-stamped, versioned, and documented results. **checkers** (Ross et al. 2018) allows you to create custom checks that examine different aspects of reproducibility. **packrat** (Ushey et al. 2018) is focused on dependencies, creating a packaged folder containing a project as well as all of its dependencies, so that projects dependent on lesser-used packages can be easily shared across computers. **drake** (OpenSci 2020) works to analyze workflows, skip steps where results are up to date, and provide evidence that results match the underlying code and data. Lastly, the **reproducible** (McIntire & Chubaty 2020) package focuses on the concept of caching–saving information so that projects can be run faster each time they are re-completed from the start.

Many of these packages focus on one specific area of reproducibility, and while they each succeed well at their intended goal, they are not necessarily the most practical option for users trying to address a wide variety of factors influencing reproducibility at once.

Additionally, with the focused nature of these packages comes added complexity. The

functions in these packages are often quite complex to use, and many steps must be completed to achieve the required reproducibility goal. This cumbersome nature means that most reproducibility packages currently available are not easily accessible to users near the beginning of their R journey, nor particularly useful to those looking for quick and easy reproducibility checks.

3 Methods

fertile addresses these gaps by providing a simple, easy-to-learn reproducibility package that, rather than focusing intensely on a specific area, provides some information about a wide variety of aspects influencing reproducibility.

The package also provides flexibility–offering benefits to users at any stage in the data analysis workflow, unlike some other available packages which can only be used before the creation of a new project or after it is finished.

fertile is designed to be used on data analyses organized as R Projects (i.e. directories containing an `.Rproj` file). Once an R Project is created, **fertile** provides benefits throughout the data analysis process, both during development as well as after the fact.

fertile achieves this by operating in two modes: proactively (to prevent reproducibility mistakes from happening in the first place), and retroactively (analyzing code that has already been written for potential problems).

3.1 Proactive Use

Proactively, the package identifies potential mistakes as they are made by the user and outputs an informative message as well as a recommended solution. For example, **fertile** catches when a user passes a potentially problematic file path—such as an absolute path, or a path that points to a location outside of the project directory—to a variety of common input/output functions operating on many different file types.

```
library(fertile)
file.exists("project_miceps/mice.csv")
```

```
## [1] TRUE
abs_path <- fs::path_abs("project_miceps/mice.csv")
read_csv(abs_path)
## Error: Detected absolute paths
```

fertile is even more aggressive with functions (like `setwd()`) that are almost certain to break reproducibility, causing them to throw errors that prevent their execution and providing recommendations for better alternatives.

```
setwd("~/Desktop")
## Error: setwd() is likely to break reproducibility. Use here::here() instead.
```

These proactive warning features are activated immediately after attaching the **fertile** package and require no additional effort by the user.

3.2 Retroactive Use

Retroactively, **fertile** analyzes potential obstacles to reproducibility in an RStudio Project (i.e., a directory that contains an `.Rproj` file). The package considers several different aspects of the project which may influence reproducibility, including the directory structure, file paths, and whether randomness is used thoughtfully.

The end products of these analyses are reproducibility reports summarizing a project's adherence to reproducibility standards and recommending remedies for where the project falls short. For example, **fertile** might identify the use of randomness in code and recommend setting a seed if one is not present.

Users can access the majority of **fertile**'s retroactive features through two primary functions, `proj_check()` and `proj_analyze()`.

The `proj_check()` function runs fifteen different reproducibility tests, noting which ones passed, which ones failed, the reason for failure, a recommended solution, and a guide to where to look for help. These tests include: looking for a clear build chain, checking to make sure the root level of the project is clear of clutter, confirming that there are no files present that are not being directly used by or created by the code, and looking for uses of randomness that to not have a call to `set.seed()` present. A full list is provided below:

```
list_checks()
## -- The available checks in ‘fertile‘ are as follows: ----------
##  [1] "has_tidy_media"          "has_tidy_images"
##  [3] "has_tidy_code"           "has_tidy_raw_data"
##  [5] "has_tidy_data"           "has_tidy_scripts"
##  [7] "has_readme"              "has_no_lint"
##  [9] "has_proj_root"           "has_no_nested_proj_root"
## [11] "has_only_used_files"     "has_clear_build_chain"
## [13] "has_no_absolute_paths"   "has_only_portable_paths"
## [15] "has_no_randomness"
```

Subsets of the fifteen tests can be invoked using the **tidyselect** helper functions in combination with the more limited `proj_check_some()` function.

```
library(tidyselect)
proj_check_some("project_miceps", contains("paths"))
##  Checking for no absolute paths
##  Checking for only portable paths
##
##
##  Reproducibility checks passed: 2
```

The `proj_analyze()` function creates a report documenting the structure of a data analysis project. This report contains information about all packages referenced in code, the files present in the directory and their types, suggestions for moving files to create a more organized structure, and a list of reproducibility-breaking file paths used in code.

```
proj_analyze("project_miceps")
## # A tibble: 1 x 3
##   package      N used_in
##   <chr>    <int> <chr>
## 1 rmarkdown     1 project_miceps/analysis.Rmd
## # A tibble: 9 x 4
##   file              ext       size mime
##   <fs::path>        <chr> <fs::byt> <chr>
## 1 Estrogen_Receptor~ docx     10.97K application/vnd.openxmlformats-officedocum~
## 2 citrate_v_time.png png     188.45K image/png
```

```
## 3 proteins_v_time.p~ png     378.91K image/png
## 4 Blot_data_updated~ csv      14.43K text/csv
## 5 CS_data_redone.csv csv       7.39K text/csv
## 6 mice.csv           csv      14.33K text/csv
## 7 README.md          md           39 text/markdown
## 8 miceps.Rproj       Rproj       204 text/rstudio
## 9 analysis.Rmd       Rmd        4.94K text/x-markdown
## # A tibble: 7 x 3
##   path_rel           dir_rel    cmd
##   <fs::path>         <fs::path> <chr>
## 1 Blot_data_updated~ data-raw   file_move('project_miceps/Blot_data_updated.csv~
## 2 CS_data_redone.csv data-raw   file_move('project_miceps/CS_data_redone.csv', ~
## 3 Estrogen_Receptor~ inst/other file_move('project_miceps/Estrogen_Receptors.do~
## 4 analysis.Rmd       vignettes  file_move('project_miceps/analysis.Rmd', fs::di~
## 5 citrate_v_time.png inst/image file_move('project_miceps/citrate_v_time.png', ~
## 6 mice.csv           data-raw   file_move('project_miceps/mice.csv', fs::dir_cr~
## 7 proteins_v_time.p~ inst/image file_move('project_miceps/proteins_v_time.png',~
## NULL
```

fertile also contains logging functionality, which records commands run in the console that have the potential to affect reproducibility, enabling users to look at their past history at any time. The package focuses mostly on package loading and file opening, noting which function was used, the path or package it referenced, and the timestamp at which that event happened.

Users can access the log recording their commands at any time via the `log_report()` function:

```
library(purrr)
library(forcats)
library(fertile)
read.csv("project_miceps/mice.csv")
```

```
log_report()
## # A tibble: 4 x 4
##   path          path_abs                          func      timestamp
##   <chr>         <chr>                             <chr>     <dttm>
## 1 package:purrr <NA>                              base::li~ 2020-03-16 18:44:07
## 2 package:forc~ <NA>                              base::li~ 2020-03-16 18:44:07
## 3 package:fert~ <NA>                              base::li~ 2020-03-16 18:44:07
## 4 project_mice~ /Users/audreybertin/Documents/fer~ utils::r~ 2020-03-16 18:44:07
```

The log, if not managed, can grow very long over time. For users who do not desire such functionality, the `log_clear()` provides a way to erase the log and start over.

```
log_clear()
log_report()
## # A tibble: 0 x 0
```

In addition to the interactive log, users can access a render log, which is created by running all of the `.R` and `.Rmd` files within a project in order to collect information on packages, file paths, and file structure used to create the output for `proj_analyze()`. This log can be accessed with the function `render_log_report()`.

3.3 Utility Functions

fertile also provides several useful utility functions that may assist with the process of data analysis. Among other things, these include functions to check the type of a file and a way to create a copy of a project in a temporary directory.

```
is_path_portable("~")
## [1] FALSE


is_data_file("project_miceps/mice.csv")
## [1] TRUE
is_image_file("project_miceps/mice.csv")
## [1] FALSE
is_image_file("project_miceps/proteins_v_time.png")
## [1] TRUE
```

```
dir <- getwd()
new_dir <- sandbox(dir)
```

```
dir
## [1] "/Users/audreybertin/Documents/fertile-paper"
fs::dir_ls(dir) %>% head()
## /Users/audreybertin/Documents/fertile-paper/README.md
## /Users/audreybertin/Documents/fertile-paper/agsm.bst
## /Users/audreybertin/Documents/fertile-paper/bibliography.bib
## /Users/audreybertin/Documents/fertile-paper/fertile poster.pptx
## /Users/audreybertin/Documents/fertile-paper/fertile-paper.Rproj
## /Users/audreybertin/Documents/fertile-paper/fertile.Rmd
```

```
new_dir
## /var/folders/v6/f62qz88s0sd5n3yqw9d8sb300000gn/T/RtmpNmHP0Q/fertile-paper
fs::dir_ls(new_dir) %>% head()
## /var/folders/v6/f62qz88s0sd5n3yqw9d8sb300000gn/T/RtmpNmHP0Q/fertile-paper/README.md
## /var/folders/v6/f62qz88s0sd5n3yqw9d8sb300000gn/T/RtmpNmHP0Q/fertile-paper/agsm.bst
## /var/folders/v6/f62qz88s0sd5n3yqw9d8sb300000gn/T/RtmpNmHP0Q/fertile-paper/bibliography.bib
## /var/folders/v6/f62qz88s0sd5n3yqw9d8sb300000gn/T/RtmpNmHP0Q/fertile-paper/fertile poster.pptx
## /var/folders/v6/f62qz88s0sd5n3yqw9d8sb300000gn/T/RtmpNmHP0Q/fertile-paper/fertile-paper.Rproj
## /var/folders/v6/f62qz88s0sd5n3yqw9d8sb300000gn/T/RtmpNmHP0Q/fertile-paper/fertile.Rmd
```

3.4 Sample Use Cases

fertile is designed in such a way as to be useful to users of all backgrounds and experiences. In this section, we provide two sample use cases demonstrating situations where users could benefit from using the package.

3.4.1 Introductory Data Science Student

Susan is taking an introductory data science course. This is her first time learning how to code, and she has not yet been exposed to ideas of research reproducibility. Her professor has assigned a data analysis project which must be completed in R Markdown. The project requires her to read in a data file located on her computer and use it to produce a graph.

She reads in the data, makes the graph, and knits her `.Rmd` file. It compiles successfully, so she submits the assignment. The next day, she receives an email from her professor saying that her assignment failed to compile and that she needs to make changes and try again.

Susan doesn't understand why it didn't work on the professor's computer when it did on her own. The professor recommends that she install **fertile** and run `proj_check()` on her assignment. She does this and gets a message informing her she used an absolute path to open her dataset, and she should use a relative path. She looks up what this means and then uses the new information to update her assignment. She resubmits and her second attempt is successful.

On future projects, she always loads and runs fertile to make sure her work is okay before submitting.

3.4.2 Experienced R User

Emma is a post-doc, with several years of R experience. She is familiar with some basic rules of reproducibility—file paths should always be relative and randomness should always be associated with a seed–but has never needed to pass any sort of reproducibility check before because her professors never emphasized that.

She has just finished a research project and is looking to submit her work to a journal. When researching the journal to which she is interested in submitting, she discovers that it has high standards for research reproducibility and a dedicated editor focusing on that aspect of submission. She goes online and finds the journal's guidelines for reproducibility. They are more complete than any guidelines to which she has previously been required to conform. In addition to notes about file paths and randomness, the journal requires a clean, well-organized folder structure, broken down by file category and stripped of files that do not serve a purpose. In order to be approved, submissions must also have a clear build chain and an informative README file.

Unsure of the best way to achieve this structure, Emma goes online to find help. In her search, she comes across **fertile**. She downloads the package, and in only a handful of commands, she identifies and removes excess files in her directory and automatically organizes her files into a structure reminiscent of an R package. She now meets the guidelines for the journal and can submit her research.

4 Results

In an effort to understand the package's effectiveness, we also share preliminary results from a randomized, controlled experiment conducted on an undergraduate introductory data science course [2]. The purpose of the study is to determine whether **fertile** helps students produce data science research that is more likely to be reproducible.

[2]This study is approved by Smith College IRB, Protocol #19-032.

5 Results

6 Conclusion

fertile is an R package that lowers barriers to reproducible data analysis projects in R, providing a wide array of checks and suggestions addressing many different aspects of project reproducibility.

fertile is meant to be educational, providing informative error messages that show how the ways in which the mistakes the user is making are problematic, as well as sharing recommendations on how to fix them. The package is designed in this way so as to promote a greater understanding of reproducibility concepts in its users, with the goal of increasing the overall awareness and understanding of reproducibility in the R community.

The package has very low barriers to entry, making it accessible to users with various levels of background knowledge. Unlike many other R packages focused on reproducibility that are currently available, the features of **fertile** can be accessed almost effortlessly. Many of the retroactive features can be accessed in only two lines of code requiring minimal arguments, and some of the proactive features can be accessed with no additional effort beyond loading the package. This, in combination with the fact that **fertile** does not focus on one specific area of reproducibility, instead covering (albeit in less detail) a wide variety of topics, means that **fertile** makes it easy for data analysts of all skill levels to quickly gain a better understanding of the reproducibility of the work.

In the moment, it often feels easiest to take a shortcut—to use an absolute path or change a working directory. However, when considering the long term path of a project, spending the extra time to improve reproducibility is worthwhile. **fertile**'s user-friendly features can help data analysts avoid these harmful shortcuts with minimal effort.

References

Blischak, J., Carbonetto, P. & Stephens, M. (2019), *workflowr: A Framework for Reproducible and Collaborative Data Science*. R package version 1.6.0.

URL: *https://CRAN.R-project.org/package=workflowr*

Broman, K. (n.d.), 'initial steps toward reproducible research: organize your data and code'.
URL: *https://kbroman.org/steps2rr/pages/organize.html*

Cooper, N., Hsing, P.-Y., Croucher, M., Graham, L., James, T., Krystalli, A. & Michonneau, F. (2017), 'A guide to reproducible code in ecology and evolution'.
URL: *https://www.britishecologicalsociety.org/wp-content/uploads/2017/12/guide-to-reproducible-code.pdf*

Eisner, D. (2018), 'Reproducibility of science: Fraud, impact factors and carelessness', *Journal of Molecular and Cellular Cardiology* **114**, 364 – 368.
URL: *http://www.sciencedirect.com/science/article/pii/S0022282817303334*

Fidler, F. & Wilcox, J. (2018), Reproducibility of scientific results, *in* E. N. Zalta, ed., 'The Stanford Encyclopedia of Philosophy', winter 2018 edn, Metaphysics Research Lab, Stanford University.

FitzJohn, R., Ashton, R., Hill, A., Eden, M., Hinsley, W., Russell, E. & Thompson, J. (2020), *orderly: Lightweight Reproducible Reporting*. R package version 1.0.4.
URL: *https://CRAN.R-project.org/package=orderly*

Gancarz, M. (2002), *Linux and the Unix Philosophy*, Digital Press, USA.

Goodman, S. N., Fanelli, D. & Ioannidis, J. P. A. (2016), 'What does research reproducibility mean?', *Science Translational Medicine* **8**(341), 341ps12.
URL: *https://stm.sciencemag.org/content/8/341/341ps12*

Gosselin, R.-D. (2020), 'Statistical analysis must improve to address the reproducibility crisis: The access to transparent statistics (acts) call to action', *BioEssays* **42**(1), 1900189.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1002/bies.201900189*

Kitzes, J., Turek, D. & Deniz, F. (2017), *The Practice of Reproducible Research: Case Studies and Lessons from the Data-Intensive Sciences*, University of California Press, Berkeley, CA.
URL: *https://www.practicereproducibleresearch.org*

Marwick, B. (2019), *rrtools: Creates a Reproducible Research Compendium*. R package version 0.1.0.
URL: *https://github.com/benmarwick/rrtools*

Marwick, B., Boettiger, C. & Mullen, L. (2018), 'Packaging data analytical work reproducibly using R (and friends)', *The American Statistician* **72**(1), 80–88.
URL: *https://peerj.com/preprints/3192.pdf*

McArthur, S. L. (2019), 'Repeatability, reproducibility, and replicability: Tackling the 3r challenge in biointerface science and engineering', *Biointerphases* **14**(2), 020201.
URL: *https://doi.org/10.1116/1.5093621*

McIntire, E. J. B. & Chubaty, A. M. (2020), *reproducible: A Set of Tools that Enhance Reproducibility Beyond Package Management*. R package version 1.0.0.
URL: *https://CRAN.R-project.org/package=reproducible*

OpenSci, R. (2020), *drake: A Pipeline Toolkit for Reproducible Computation at Scale*. R package version 7.11.0.
URL: *https://cran.r-project.org/package=drake*

Peng, R. D. (2009), 'Reproducible research and Biostatistics', *Biostatistics* **10**(3), 405–408.
URL: *https://doi.org/10.1093/biostatistics/kxp014*

R-Core-Team (2020), 'Writing r extensions', *R Foundation for Statistical Computing* .
URL: *http://cran.stat.unipd.it/doc/manuals/r-release/R-exts.pdf*

Reproducibility in Science (n.d.).
URL: *http://ropensci.github.io/reproducibility-guide/*

Ross, N., DeCicco, L. & Randhawa, N. (2018), *checkers: Automated checking of best practices for research compendia*. R package version 0.1.0.
URL: *https://github.com/ropenscilabs/checkers/blob/master/DESCRIPTIONr*

Ushey, K., McPherson, J., Cheng, J., Atkins, A. & Allaire, J. (2018), *packrat: A Dependency Management System for Projects and their R Package Dependencies*. R package

version 0.5.0.

URL: *https://CRAN.R-project.org/package=packrat*

Wallach, J. D., Boyack, K. W. & Ioannidis, J. P. A. (2018), 'Reproducible research practices, transparency, and open access data in the biomedical literature, 20152017', *PLOS Biology* **16**(11), 1–20.

URL: *https://doi.org/10.1371/journal.pbio.2006930*

Wickham, H. (2015), *R Packages*, 1st edn, OReilly Media, Inc.