Addressing The Scientific Reproducibility Crisis Through Educational Software Integration

Audrey M. Bertin

Submitted to the Department of Statistical and Data Sciences
of Smith College
in partial fulfillment
of the requirements for the degree of
Bachelor of Arts

Benjamin S. Baumer, Honors Project Advisor

May 2021

Acknowledgements

I want to thank a few people.

Preface

This is an example of a thesis setup to use the reed thesis document class (for LaTeX) and the R bookdown package, in general.

Table of Contents

Chapter 1: This will automatically install the {remotes} package and {thesisdown} **1**

Chapter 2: An Introduction to Reproducibility **3**
2.1 What Is Reproducibility? 4
2.2 The Reproducibility Crisis 4
2.3 The Components of Reproducible Research 4
2.4 Current Attempts to Address Reproducibility in Scientific Publishing 4
2.4.1 Case Studies Across The Sciences 4
2.4.2 Case Studies In The Statistical And Data Sciences 4
2.4.3 The Bigger Picture 4
2.4.4 Assessing the Success of Academic Reproducibility Policies . . 4
2.5 Limitations on Achieving Reproducibility in Scientific Publishing . . . 4
2.5.1 Challenges for Authors 4
2.5.2 Challenges for Journals 4
2.6 Attempts to Address These Limitations 4
2.6.1 Through Education 4
2.6.2 Through Software 4

Chapter 3: `fertile`: My Contribution To Addressing Reproducibility **5**
3.1 Understanding The Gaps In Existing Reproducibility Solutions 5
3.1.1 In Education 5
3.1.2 In Software 5
3.2 `fertile`, An R Package Creating Optimal Conditions For Reproducibility 6
3.2.1 Package Overview 6
3.2.2 Proactive Use 6
3.2.3 Retroactive Use 7
3.2.4 Logging 10
3.2.5 Utility Functions 10
3.2.6 File Path Management 10
3.2.7 File Types 11
3.2.8 Temporary Directories 11
3.2.9 Managing Project Dependencies 12
3.3 How `fertile` Works 12

3.4 `fertile` in Practice: Experimental Results From Smith College Student Use . . . 13

Chapter 4: Incorporating Reproducibility Tools Into The Greater Data Science Community . . . 17
4.1 Potential Applications of `fertile` . . . 17
4.1.1 In Journal Review . . . 17
4.1.2 By Beginning Data Scientists . . . 17
4.1.3 By Advanced Data Scientists . . . 17
4.1.4 For Teaching Reproducibility . . . 17
4.2 Integration Of `fertile` And Other Reproducibility Tools in Data Science Education . . . 17

Conclusion . . . 19

Appendix A: The First Appendix . . . 21

Appendix B: The Second Appendix, for Fun . . . 23

References . . . 25

Abstract

The preface pretty much says it all.

Second paragraph of abstract starts here.

Dedication

You can have a dedication here if you wish.

Chapter 1

This will automatically install the {remotes} package and {thesisdown}

Placeholder

Chapter 2

An Introduction to Reproducibility

Placeholder

2.1 What Is Reproducibility?

2.2 The Reproducibility Crisis

2.3 The Components of Reproducible Research

2.4 Current Attempts to Address Reproducibility in Scientific Publishing

2.4.1 Case Studies Across The Sciences

2.4.2 Case Studies In The Statistical And Data Sciences

2.4.3 The Bigger Picture

2.4.4 Assessing the Success of Academic Reproducibility Policies

2.5 Limitations on Achieving Reproducibility in Scientific Publishing

2.5.1 Challenges for Authors

2.5.2 Challenges for Journals

2.6 Attempts to Address These Limitations

2.6.1 Through Education

2.6.2 Through Software

Chapter 3

`fertile`: My Contribution To Addressing Reproducibility

3.1 Understanding The Gaps In Existing Reproducibility Solutions

Although the current state of reproducibility in academia is quite poor, it is not an impossible challenge to overcome. The relative simplicity of addressing reproducibility, particularly when compared with replicability, makes it an ideal candidate for solution-building. Although significant progress on addressing reproducibility on a widespread scale is a long-term challenge, impactful forward progress–if on a smaller scale–can be achieved in the short-term.

As we have seen, software developers, data scientists, and educators around the world have realized this potential, taking steps to help address the current crisis of reproducibility. Journals have put in place guidelines for authors, statisticians have developed `R` packages that help structure projects in a reproducible format, and educators have begun integrate reproducibility exercises into their courses.

However, many of these attempts to address reproducibility have significant drawbacks associated with them. We have already explored the issues with journal policies, both for authors and reviewers, in-depth. In this section, we will consider the education and software solutions and their associated challenges.

3.1.1 In Education

3.1.2 In Software

Many of these packages are narrow, with each effectively addressing a small component of reproducibility: file structure, modularization of code, version control, etc. These packages often succeed in their area of focus, but at the cost of accessibility to a wider audience. Their functions 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.

A more effective way of realizing widespread reproducibility is to make the process for doing so simple enough that it takes little to no conscious effort to implement. You want users to "fall into a hole" (we paraphrase Hadley Wickham) of good practice.

However, while these tools can be useful, they are generalized so as to be useful to the widest audience. As a result, their checks are not designed to be R-specific, which makes them sub-optimal for users looking to address reproducibility issues involving features specific to the R programming language, such as package installation and seed setting.

3.2 fertile, An R Package Creating Optimal Conditions For Reproducibility

3.2.1 Package Overview

fertile attempts to address these gaps in existing software 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. fertile is flexible, offering benefits to users at any stage in the data analysis workflow, and provides R-specific features, which address certain aspects of reproducibility that can be missed by external project development software.

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).

Much of the available literature focuses on file structure, organization, and naming, and fertile's features are consistent with this. Marwick, Boettiger, & Mullen (2018) provide the framework for file structure that fertile is based on: a structure similar to that of an R package (R-Core-Team (2020), Wickham (2015)), with an R folder, as well as data, data-raw, inst, and vignettes.

3.2.2 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("~/Desktop/my_data.csv")
```

```
[1] TRUE
```

```
read.csv("~/Desktop/my_data.csv")
```

```
Error: Detected absolute paths
```

```
read.csv("../../../Desktop/my_data.csv")
```

```
Error: Detected paths that lead outside the project directory
```

`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.

3.2.3 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 do not have a call to `set.seed()` present. A full list is provided below:

```
list_checks()
```

```
-- The available checks in ‘fertile‘ are as follows: ------------ fertile 0.0.0.9027 --

 [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 (Henry & Wickham (2020)) in combination with the more limited `proj_check_some()` function.

```
proj_dir <- "project_miceps"
```

```
proj_check_some(proj_dir, contains("paths"))
```

```
-- Compiling... ------------------------------------------------ fertile 0.0.0.9027 --

-- Rendering R scripts... -------------------------------------- fertile 0.0.0.9027 --

-- Running reproducibility checks ------------------------------ fertile 0.0.0.9027 --

v Checking for no absolute paths

v Checking for only portable paths

-- Summary of fertile checks ----------------------------------- fertile 0.0.0.9027 --

v Reproducibility checks passed: 2
```

Each test can also be run individually by calling the function matching its check name.

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(proj_dir)
```

```
-- Analysis of reproducibility for project_miceps ---------------- fertile 0.0.0.902
--   Packages referenced in source code -------------------------- fertile 0.0.0.902
# A tibble: 9 x 3
  package       N used_in
  <chr>     <int> <chr>
1 broom         1 project_miceps/analysis.Rmd
2 dplyr         1 project_miceps/analysis.Rmd
3 ggplot2       1 project_miceps/analysis.Rmd
4 purrr         1 project_miceps/analysis.Rmd
5 readr         1 project_miceps/analysis.Rmd
6 rmarkdown     1 project_miceps/analysis.Rmd
7 skimr         1 project_miceps/analysis.Rmd
8 stargazer     1 project_miceps/analysis.Rmd
9 tidyr         1 project_miceps/analysis.Rmd
--   Files present in directory ---------------------------------- fertile 0.0.0.902
# 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     187.46K image/png
3 proteins_v_time.p~ png     378.17K 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
--   Suggestions for moving files ------------------------------- fertile 0.0.0.902
# 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',~
--   Problematic paths logged ------------------------------------ fertile 0.0.0.902
NULL
```

3.2.4 Logging

`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:

```
log_report()
```

```
# A tibble: 6 x 4
  path          path_abs                                func      timestamp
  <chr>         <chr>                                   <chr>     <dttm>
1 package:remo~ <NA>                                    base::re~ 2020-09-14 15:26:52
2 package:thes~ <NA>                                    base::re~ 2020-09-14 15:26:52
3 package:thes~ <NA>                                    base::li~ 2020-09-14 15:26:52
4 package:purrr <NA>                                    base::li~ 2020-09-14 15:27:13
5 package:forc~ <NA>                                    base::li~ 2020-09-14 15:27:13
6 project_mice~ /Users/audreybertin/Documents/the~ readr::r~ 2020-09-14 15:27:13
```

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

3.2.5 Utility Functions

`fertile` also provides several useful utility functions that may assist with the process of data analysis.

3.2.6 File Path Management

The `check_path()` function analyzes a vector of paths (or a single path) to determine whether there are any absolute paths or paths that lead outside the project directory.

```
# Path inside the directory
check_path("project_miceps")
```

```
# A tibble: 0 x 3
# ... with 3 variables: path <chr>, problem <chr>, solution <chr>
```

```
# Absolute path (current working directory)
check_path(getwd())
```

```
Error: Detected absolute paths
```

```
# Path outside the directory
check_path("../fertile.Rmd")
```

```
Error: Detected paths that lead outside the project directory
```

3.2.7 File Types

There are several functions that can be used to check the type of a file:

```
is_data_file(fs::path(proj_dir, "mice.csv"))
```

```
[1] TRUE
```

```
is_image_file(fs::path(proj_dir, "proteins_v_time.png"))
```

```
[1] TRUE
```

```
is_text_file(fs::path(proj_dir, "README.md"))
```

```
[1] TRUE
```

```
is_r_file(fs::path(proj_dir, "analysis.Rmd"))
```

```
[1] TRUE
```

3.2.8 Temporary Directories

The `sandbox()` function allows the user to make a copy of their project in a temporary directory. This can be useful for ensuring that projects run properly when access to the local file system is removed.

```
proj_dir
```

```
[1] "project_miceps"
```

```
fs::dir_ls(proj_dir) %>% head(3)
```

```
project_miceps/Blot_data_updated.csv   project_miceps/CS_data_redone.csv
project_miceps/Estrogen_Receptors.docx
```

```
temp_dir <- sandbox(proj_dir)
temp_dir
```

```
/var/folders/v6/f62qz88s0sd5n3yqw9d8sb300000gn/T/RtmpEMcSA2/project_miceps
```

```
fs::dir_ls(temp_dir) %>% head(3)
```

```
/var/folders/v6/f62qz88s0sd5n3yqw9d8sb300000gn/T/RtmpEMcSA2/project_miceps/Blot_data_updat
/var/folders/v6/f62qz88s0sd5n3yqw9d8sb300000gn/T/RtmpEMcSA2/project_miceps/CS_data_redone.
/var/folders/v6/f62qz88s0sd5n3yqw9d8sb300000gn/T/RtmpEMcSA2/project_miceps/Estrogen_Recept
```

3.2.9 Managing Project Dependencies

One of the challenges with ensuring that work is reproducible is the issue of dependencies. Many data analysis projects reference a variety of `R` packages in their code. When such projects are shared with other users who may not have the required packages downloaded, it can cause errors that prevent the project from running properly.

The `proj_pkg_script()}` function assists with this issue by making it simple and fast to download dependencies. When run on an `R` project directory, the function creates a `.R` script file that contains the code needed to install all of the packages referenced in the project, differentiating between packages located on CRAN and those located on GitHub.

```
install_script <- proj_pkg_script(proj_dir)
cat(readChar(install_script, 1e5))
```

```
# Run this script to install the required packages for this
R project.
# Packages hosted on CRAN...
install.packages(c( 'broom', 'dplyr', 'ggplot2', 'purrr',
'readr', 'rmarkdown', 'skimr', 'stargazer', 'tidyr' ))
# Packages hosted on GitHub...
```

3.3 How `fertile` Works

Much of the functionality in `fertile` is achieved by writing `shims` **link to wikipedia page here**. `fertile`'s shimmed functions intercept the user's commands and perform various logging and checking tasks before executing the desired function. Our process is:

1. Identify an `R` function that is likely to be involved in operations that may break reproducibility. Popular functions associated with only one package (e.g., `read_csv()` from `readr`) are ideal candidates.

2. Create a function in `fertile` with the same name that takes the same arguments (and always the dots `...`).

3. Write this new function so that it:

a) captures any arguments,
b) logs the name of the function called,
c) performs any checks on these arguments, and
d) calls the original function with the original arguments. Except where warranted, the execution looks the same to the user as if they were calling the original function.

Most shims are quite simple and look something like what is shown below for `read_csv()`.

```
fertile::read_csv
```

```
function(file, ...) {
  if (interactive_log_on()) {
    log_push(file, "readr::read_csv")
    check_path_safe(file)
    readr::read_csv(file, ...)
  }
}
<bytecode: 0x7f9e23fafa70>
<environment: namespace:fertile>
```

`fertile` shims many common functions, including those that read in a variety of data types, write data, and load packages. This works both proactively and retroactively, as the shimmed functions written in `fertile` are activated both when the user is coding interactively and when a file containing code is rendered.

In order to ensure that the `fertile` versions of functions ("shims") always supersede ("mask") their original namesakes when called, `fertile` uses its own shims of the `library` and `require` functions to manipulate the `R search` path so that it is always located in the first position. In the `fertile` version of `library()`, we detach `fertile` from the search path, load the requested package, and then re-attach `fertile`. This ensures that when a user executes a command, `R` will check `fertile` for a matching function before considering other packages. While it is possible that this shifty behavior could lead to unintended consequences, our goal is to catch a good deal of problems before they become problematic. Users can easily disable `fertile` by detaching it, or not loading it in the first place.

3.4 `fertile` in Practice: Experimental Results From Smith College Student Use

`fertile` is designed to: 1) be simple enough that users with minimal `R` experience can use the package without issue, 2) increase the reproducibility of work produced

by its users, and 3) educate its users on why their work is or is not reproducible and provide guidance on how to address any problems.

To test `fertile`'s effectiveness, we began an initial randomized control trial of the package on an introductory undergraduate data science course at Smith College in Spring 2020 **ADD FOOTNOTE** (This study was approved by Smith College IRB, Protocol #19-032).

The experiment was structured as follows:

1.Students are given a form at the start of the semester asking whether they consent to participate in a study on data science education. In order to successfully consent, they must provide their system username, collected through the command `Sys.getenv("LOGNAME")`. To maintain privacy the results are then transformed into a hexadecimal string via the `md5()` hashing function.

2. These hexadecimal strings are then randomly assigned into equally sized groups, one experimental group that receives the features of `fertile` and one group that receives a control.

3. The students are then asked to download a package called `sds192` (the course number and prefix), which was created for the purpose of this trial. It leverages an `.onAttach()` function to scan the `R` environment and collect the username of the user who is loading the package and run it through the same hashing algorithm as used previously. It then identifies whether that user belongs to the experimental or the control group. Depending on the group they are in, they receive a different version of the package.

4. The experimental group receives the basic `sds192` package, which consists of some data sets and `R` Markdown templates necessary for completing homework assignments and projects in the class, but also has `fertile` installed and loaded silently in the background. The package's proactive features are enabled, and therefore users will receive warning messages when they use absolute or non-portable paths or attempt to change their working directory. The control group receives only the basic `sds192` package, including its data sets and `R` Markdown templates. All students from both groups then use their version of the package throughout the semester on a variety of projects.

5. Both groups are given a short quiz on different components of reproducibility that are intended to be taught by `fertile` at both the beginning and end of the semester. Their scores are then compared to see whether one group learned more than the other group or whether their scores were essentially equivalent. Additionally, for every homework assignment submitted, the professor takes note of whether or not the project compiles successfully.

Based on the results, we hope to determine whether `fertile` was successful at achieving its intended goals. A lack of notable difference between the *experimental* and *control* groups in terms of the number of code-related questions asked throughout the semester would indicate that `fertile` achieved its goal of simplicity. A higher average for the *experimental* group in terms of the number of homework assignments

that compiled successfully would indicate that `fertile` was successful in increasing reproducibility. A greater increase over the semester in the reproducibility quiz scores for students in the *experimental* group compared with the *control* group would indicate that `fertile` achieved its goal of educating users on reproducibility. Success according to these metrics would provide evidence showing `fertile`'s benefit as tool to help educators introduce reproducibility concepts in the classroom.

Chapter 4

Incorporating Reproducibility Tools Into The Greater Data Science Community

4.1 Potential Applications of `fertile`

4.1.1 In Journal Review

4.1.2 By Beginning Data Scientists

4.1.3 By Advanced Data Scientists

4.1.4 For Teaching Reproducibility

Nicole Janz – Brining the Gold Standard into the Classroom: Replication in University Teaching

4.2 Integration Of `fertile` And Other Reproducibility Tools in Data Science Education

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, including file organization, file path usage, documentation, and dependencies. `fertile` is meant to be educational, providing informative error messages that indicate why users' mistakes are problematic and 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.

Appendix A

The First Appendix

This first appendix includes all of the R chunks of code that were hidden throughout the document (using the `include = FALSE` chunk tag) to help with readibility and/or setup.

In the main Rmd file

Appendix B

The Second Appendix, for Fun

References

Placeholder

Henry, L., & Wickham, H. (2020). Tidyselect: Select from a set of strings. Retrieved from https://CRAN.R-project.org/package=tidyselect

Marwick, B., Boettiger, C., & Mullen, L. (2018). Packaging data analytical work reproducibly using R (and friends). *The American Statistician*, *72*(1), 80–88. http://doi.org/doi.org/10.1080/00031305.2017.1375986

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

Wickham, H. (2015). *R packages* (1st ed.). Sebastopol, CA: O'Reilly Media, Inc.