My Final College Paper

A Thesis
Presented to
The Division of Statistical and Data Sciences
Smith College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts

Audrey M. Bertin

May 2021

Approved for the Division
(Statistical and Data Sciences)

_____

Benjamin S. Baumer

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

Introduction **1**

Chapter 1: An Introduction to Reproducibility **3**
1.1 What is reproducibility? 3
1.2 The Reproducibility Crisis 4
1.3 A Brief History of Reproducibility 4
1.4 The Components of Reproducible Research 5
1.4.1 Literature 5
1.5 Current Focus on Reproducibility in Academia 6

Chapter 2: Addressing the Challenges of Reproducibility **7**
2.1 Review Of Previous Work 7
2.1.1 R Packages and Other Software 7
2.2 Identifying Gaps In Existing Solutions 9
2.3 My Contribution: `fertile`, An R Package Creating Optimal Conditions For Reproducibility 9
2.3.1 Package Overview 9
2.3.2 Proactive Use 9
2.3.3 Retroactive Use 10
2.3.4 Logging 13
2.3.5 Utility Functions 13
2.3.6 File Path Management 13
2.3.7 File Types 14
2.3.8 Temporary Directories 14
2.3.9 Managing Project Dependencies 15
2.4 How `fertile` Works 15
2.5 `fertile` in Practice: Experimental Results From Smith College Student Use 16

Chapter 3: Incorporating Reproducibility Tools Into The Greater Data Science Community **19**
3.1 Potential Applications of `fertile` 19
3.1.1 In Journal Review 19
3.1.2 By Beginning Data Scientists 19
3.1.3 By Advanced Data Scientists 19

3.1.4 For Teaching Reproducibily 19
3.2 Integration Of `fertile` And Other Reproducibility Tools in Data Science Education 19

Conclusion **21**

Appendix A: The First Appendix **23**

Appendix B: The Second Appendix, for Fun **25**

References **27**

Abstract

The preface pretty much says it all.

Second paragraph of abstract starts here.

Dedication

You can have a dedication here if you wish.

Introduction

Potential sources:

https://arxiv.org/abs/1401.3269

https://academic.oup.com/isp/article-abstract/17/4/392/2528285

https://dl.acm.org/doi/abs/10.1145/3186266?casa_token=3yv8mooiZXYAAAAA:AUWshgsmm-4ulr7qYK2vm3a6EdJneFLgn3nxplGeaZpT7hgFcIRkRA7edGHdjg_pPvs5p-GoHHFo

https://dl.acm.org/doi/abs/10.1145/3027385.3029445?casa_token=rgNbcFWIA1AAAAAA:cZRPAy1KYewKfheFe74GBDwKzF9Q8X0xMau0AtBVkYSTYrd7apJEnwDY_T8oqh1cZQED1gHjjtq4

https://berkeleysciencereview.com/2014/06/reproducible-collaborative-data-science/

https://guides.lib.uw.edu/research/reproducibility/teaching

https://escholarship.org/uc/item/90b2f5xh

https://www.mitpressjournals.org/doi/full/10.1162/dint_a_00053

https://www.pnas.org/content/115/11/2584

https://www.pnas.org/content/115/11/2561

Chapter 1

An Introduction to Reproducibility

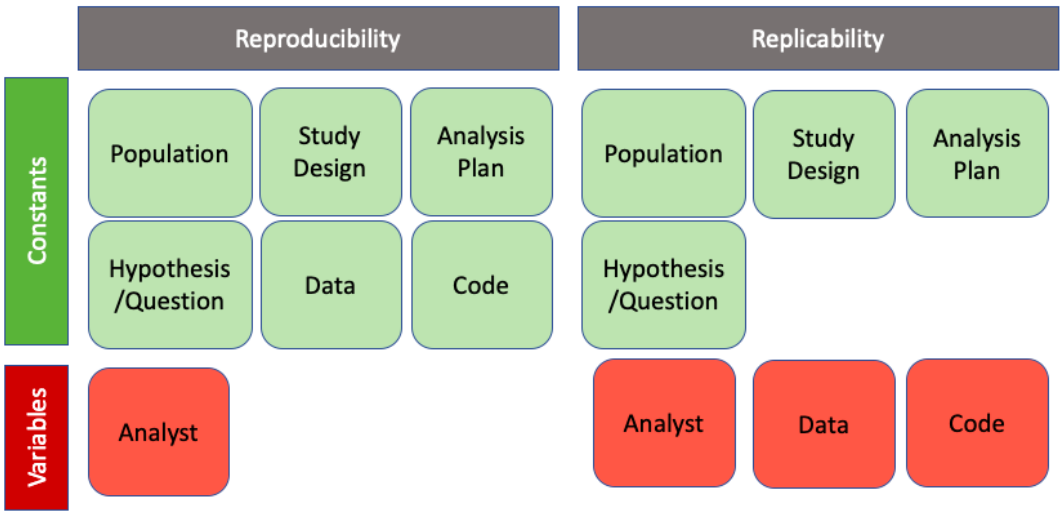
1.1 What is reproducibility?

In the field of data science, research is considered fully *reproducible* when the requisite code and data files produce identical results when run by another analyst.

Reproducible research has a wide variety of benefits in the scientific community. 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 also 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.

Although often confused, the concept of *reproducibility* is distinct from the similar idea of *replicability*: the ability of a researcher to duplicate the results of a study when following the original procedure but collecting new data. Replicability has larger-scale implications than reproducibilty; the findings of research studies can not be accepted unless a variety of other researchers come to the same conclusions through independent work.

```
knitr::include_graphics("figure/versus.png")
```



Reproducibility and replicability are both necessary to the advancement of scientific research, but they vary significantly in terms of their difficulty to achieve. Reproducibility, in theory, is somewhat simple to attain in data analyses–because code is inherently non-random (excepting applications involving random number generation) and data remain consistent, variability is highly restricted. The achievement of replicability, on the other hand, is a much more complex challenge, involving significantly more variablility and requiring high quality data, effective study design, and incredibly robust hypotheses.

1.2 The Reproducibility Crisis

Despite the relative simplicity of the reproducibility problem, the scientific community has been unable to solve it. 52% of respondents in a 2016 Nature survey believed that science was going through a "crisis" of reproducibility. `https://www.nature.com/news/1-500-scientists-lift-the-lid-on-reproducibility-1.19970` Other studies paint an even bleaker picture: a 2015 study found that over 50% of studies psychology failed reproducibility tests and research from 2012 found that figure closer to 90% in the field of cancer biology. (`https://www.nature.com/news/over-half-of-psychology-studies-fail-reproducibility-test-1.18248`, `https://www.nature.com/articles/483531a`)

Without reproducibility, the scientific community cannot properly verify study results. This makes it difficult to identify which information should be believed and which should not and increases the likelihood that studies sharing misleading information will be dispersed.

1.3 A Brief History of Reproducibility

In recent years, reproducibility has moved to the forefront of scientific discussion. The rise of data-driven technologies, alongside our newly founded ability to instantly share

knowledge worldwide, has made reproducibility increasingly critical to the advancement of scientific understanding. Academics have recognized this, and publications on the topic appear to have increased siginficantly in the last several years (Eisner (2018); Fidler & Wilcox (2018); Gosselin (2020); McArthur (2019); Wallach, Boyack, & Ioannidis (2018)).

However, peproducibility has not always been at the forefront of scientific discussion.

- Brief history of reproducibility • Reproducibility vs replicability

1.4 The Components of Reproducible Research

1.4.1 Literature

Publications on reproducibility can be found in all areas of scientific research. However, as Goodman, Fanelli, & Ioannidis (2016) argue, the language and conceptual framework of research reproducibility varies significantly across the sciences, and there are no clear standards on reproducibility agreed upon by the scientific community as a whole. We consider recommendations from a variety of fields and determine the key aspects of reproducibility faced by scientists in different disciplines.

Kitzes, Turek, & Deniz (2017) present 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 their work does not come to a consensus on the exact standards of reproducibility that should be followed, several common trends and principles emerge from their case studies: 1) use clear separation, labeling, and documentation, 2) automate processes when possible, and 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 (2003)).

Cooper et al. (2017) focus on data analysis in `R` and identify a similar list of important reproducibility components, reinforcing 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 (2019) 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`.

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 (Martinez et al. (2018)).

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. These recommendations are a matter of opinion—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. Readable, styled code:

- Code should be written in a coherent style. Code that conforms to a style guide or is written in a consistent dialect is easier to read (Hermans & Aldewereld (2017)). We believe that the `tidyverse` provides the most accessible dialect of `R`.

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`.

1.5 Current Focus on Reproducibility in Academia

Chapter 2

Addressing the Challenges of Reproducibility

[

The relative simplicity of the reproducibility challenge 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.

My research has focused on doing just this, providing tools to assist with achieving reproducibility in one segment of the scientific community: RStudio users.

] — DO WE NEED THIS?

2.1 Review Of Previous Work

2.1.1 R Packages and Other Software

Much of this work is highly generalized, written to be applicable to users working with a variety of statistical software programs. Because all statistical software programs operate differently, these recommendations are inherently vague and difficult to implement, particularly to new analysts who are relatively unfamiliar with their software. Focused attempts to address reproducibility in specific certain software programs are more likely to be successful. We focus on `R`, due to its open-source nature, accessibility, and popularity as a tool for statistical analysis.

A small body of `R` packages focuses on research reproducibility. `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, Carbonetto, & Stephens (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, DeCicco, & Randhawa (2018)) allows you to create custom checks that examine different aspects of reproducibility. `packrat` (Ushey, McPherson, Cheng, Atkins, & Allaire (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)) analyzes workflows, skips steps where results are up to date, and provides 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 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.

`Continuous integration` tools provide more general approaches to automated checking, which can enhance reproducibility with minimal code. For example, `wercker`—a command line tool that leverages Docker—enables users to test whether their projects will successfully compile when run on a variety of operating systems without access to the user's local hard drive (Oracle Corporation (2019)). `GitHub Actions` is integrated into GitHub and can be configured to do similar checks on projects hosted in repositories. `Travis CI` and `Circle CI` are popular continuous integration tools that can also be used to check `R` code.

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.

2.2 Identifying Gaps In Existing Solutions

2.3 My Contribution: `fertile`, An R Package Creating Optimal Conditions For Reproducibility

2.3.1 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).

2.3.2 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.

2.3.3 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 f

 [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... -------------------

-- Running reproducibility checks -----------

v Checking for no absolute paths

v Checking for only portable paths

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

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_mi

--   Packages referenced in source code -----

# 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 -------------

                     file   ext    size
1 Estrogen_Receptors.docx  docx  10.97K
2      citrate_v_time.png   png 184.07K
3     proteins_v_time.png   png 377.88K
4   Blot_data_updated.csv   csv  14.43K
5      CS_data_redone.csv   csv   7.39K
6                mice.csv   csv  14.33K
7               README.md    md      39
8            miceps.Rproj Rproj     204
9            analysis.Rmd   Rmd   4.94K
                                                                          mime
1 application/vnd.openxmlformats-officedocument.wordprocessingml.document
2                                                                    image/png
3                                                                    image/png
4                                                                     text/csv
5                                                                     text/csv
6                                                                     text/csv
7                                                                text/markdown
8                                                                 text/rstudio
9                                                               text/x-markdown

--   Suggestions for moving files -----------

                 path_rel    dir_rel
1   Blot_data_updated.csv   data-raw
2      CS_data_redone.csv   data-raw
3 Estrogen_Receptors.docx inst/other
4            analysis.Rmd  vignettes
5      citrate_v_time.png inst/image
6                mice.csv   data-raw
7     proteins_v_time.png inst/image

1     file_move('project_miceps/Blot_data_updated.csv', fs::dir_create('project_miceps/dat
2        file_move('project_miceps/CS_data_redone.csv', fs::dir_create('project_miceps/dat
3 file_move('project_miceps/Estrogen_Receptors.docx', fs::dir_create('project_miceps/inst/
4             file_move('project_miceps/analysis.Rmd', fs::dir_create('project_miceps/vign
5      file_move('project_miceps/citrate_v_time.png', fs::dir_create('project_miceps/inst/
6                  file_move('project_miceps/mice.csv', fs::dir_create('project_miceps/dat
7     file_move('project_miceps/proteins_v_time.png', fs::dir_create('project_miceps/inst/

--   Problematic paths logged ---------------

NULL
```

2.3.4 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: 3 x 4
  path        path_abs                       func     timestamp
  <chr>       <chr>                          <chr>    <dttm>
1 package:pu~ <NA>                           base::l~ 2020-09-04 18:04:05
2 package:fo~ <NA>                           base::l~ 2020-09-04 18:04:05
3 project_mi~ /Users/audreybertin/Docume~ readr::~ 2020-09-04 18:04:05
```

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.

2.3.5 Utility Functions

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

2.3.6 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
```

2.3.7 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
```

2.3.8 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/RtmpycQcgK/project_miceps
```

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

```
/var/folders/v6/f62qz88s0sd5n3yqw9d8sb300000gn/T/RtmpycQcgK/project_miceps/Blot_data
/var/folders/v6/f62qz88s0sd5n3yqw9d8sb300000gn/T/RtmpycQcgK/project_miceps/CS_data_r
/var/folders/v6/f62qz88s0sd5n3yqw9d8sb300000gn/T/RtmpycQcgK/project_miceps/Estrogen_
```

2.3.9 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', 'ski
# Packages hosted on GitHub...
```

2.4 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: 0x7ff482ebd638>
<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.

2.5 `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 3

Incorporating Reproducibility Tools Into The Greater Data Science Community

3.1 Potential Applications of `fertile`

3.1.1 In Journal Review

3.1.2 By Beginning Data Scientists

3.1.3 By Advanced Data Scientists

3.1.4 For Teaching Reproducibily

3.2 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

```
# This chunk ensures that the thesisdown package is
# installed and loaded. This thesisdown package includes
# the template files for the thesis.
if (!require(remotes)) {
  if (params$`Install needed packages for {thesisdown}`) {
    install.packages("remotes", repos = "https://cran.rstudio.com")
  } else {
    stop(
      paste('You need to run install.packages("remotes")",
            "first in the Console.')
    )
  }
}
if (!require(thesisdown)) {
  if (params$`Install needed packages for {thesisdown}`) {
    remotes::install_github("ismayc/thesisdown")
  } else {
    stop(
      paste(
        "You need to run",
        'remotes::install_github("ismayc/thesisdown")',
        "first in the Console."
      )
    )
  }
}
library(thesisdown)
```

```
# Set how wide the R output will go
options(width = 70)
```

In Chapter ??:

Appendix B

The Second Appendix, for Fun

References

Blischak, J., Carbonetto, P., & Stephens, M. (2019). Workflowr: A framework for reproducible and collaborative data science. Retrieved from `https://CRAN.R-project.org/package=workflowr`

Broman, K. (2019). Initial steps toward reproducible research: Organize your data and code. *Sitewide ATOM*. Retrieved from `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. *British Ecological Society*. Retrieved from `https://www.britishecologicalsociety.org/wp-content/uploads/2017/12/guide-to-reproducible-code.pdf`

Eisner, D. A. (2018). Reproducibility of science: Fraud, impact factors and carelessness. *Journal of Molecular and Cellular Cardiology*, *114*, 364–368. `http://doi.org/https://doi.org/10.1016/j.yjmcc.2017.10.009`

Fidler, F., & Wilcox, J. (2018). Reproducibility of scientific results. In E. N. Zalta (Ed.), *The stanford encyclopedia of philosophy* (Winter 2018). `https://plato.stanford.edu/archives/win2018/entries/scientific-reproducibility/`; Metaphysics Research Lab, Stanford University.

FitzJohn, R., Ashton, R., Hill, A., Eden, M., Hinsley, W., Russell, E., & Thompson, J. (2020). Orderly: Lightweight reproducible reporting. Retrieved from `https://CRAN.R-project.org/package=orderly`

Gancarz, M. (2003). *Linux and the unix philosophy* (2nd ed.). Woburn, MA: Digital Press.

Goodman, S. N., Fanelli, D., & Ioannidis, J. P. A. (2016). What does research reproducibility mean? *Science Translational Medicine*, *8*(341), 1–6. `http://doi.org/10.1126/scitranslmed.aaf5027`

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. `http://doi.org/10.1002/bies.201900189`

Henry, L., & Wickham, H. (2020). Tidyselect: Select from a set of strings. Retrieved

from https://CRAN.R-project.org/package=tidyselect

Hermans, F., & Aldewereld, M. (2017). Programming is writing is programming. In *Companion to the first international conference on the art, science and engineering of programming* (pp. 1–8).

Kitzes, J., Turek, D., & Deniz, F. (2017). *The practice of reproducible research: Case studies and lessons from the data-intensive sciences*. Berkeley, CA: University of California Press. Retrieved from https://www.practicereproducibleresearch.org

Martinez, C., Hollister, J., Marwick, B., Szöcs, E., Zeitlin, S., Kinoshita, B. P., … Meinke, B. (2018). Reproducibility in Science: A Guide to enhancing reproducibility in scientific results and writing. Retrieved from http://ropensci.github.io/reproducibility-guide/

Marwick, B. (2019). Rrtools: Creates a reproducible research compendium. Retrieved from 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. http://doi.org/doi.org/10.1080/00031305.2017.1375986

McArthur, S. L. (2019). Repeatability, reproducibility, and replicability: Tackling the 3R challenge in biointerface science and engineering. *Biointerphases*, *14*(2), 1–2. http://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. Retrieved from https://CRAN.R-project.org/package=reproducible

OpenSci, R. (2020). Drake: A pipeline toolkit for reproducible computation at scale. Retrieved from https://cran.r-project.org/package=drake

Oracle Corporation. (2019). Wercker. Retrieved from https://github.com/wercker/wercker

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

Ross, N., DeCicco, L., & Randhawa, N. (2018). Checkers: Automated checking of best practices for research compendia. Retrieved from 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. Retrieved from 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, 2015-2017. *PLOS Biology*, *16*(11), 1–20. http://doi.org/10.1371/journal.pbio.2006930

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