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

Audrey M. Bertin
Statistical and Data Sciences
Smith College
abertin@smith.edu

Benjamin S. Baumer
Statistical and Data Sciences
Smith College
bbaumer@smith.edu

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

1 Introduction

Data-based research cannot be fully *reproducible* unless the requisite code and data files produce identical results when run by another analyst.

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.

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 (**?**). 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. Several researchers and R users have recognized this and worked to publish a small body of papers and R packages focusing on increasing reproducibility in the R community.

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 well, but at a cost. They are often semi-complex 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

2.1 Research Articles

2.2 Other Articles

2.3 R Packages

3 Methods

fertile operates 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). fertile is available for download at https://github.com/baumer-lab/fertile.

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.

shadecolorrgb0.969, 0.969, 0.969fgcolor library(fertile) file.exists(” /Desktop/my$_{d}ata.csv$”)

```
## [1] TRUE
```

read.csv(” /Desktop/my$_{d}ata.csv$”)

```
errorcolor## Error: Detected absolute
paths
```

fertile is even more aggressive with functions (like setwd()) that are almost certain to break reproducibility—it causes them to throw errors.

shadecolorrgb0.969, 0.969, 0.969fgcolor setwd(” /Desktop”)

```
errorcolor## Error: setwd() is
likely to break reproducibility. Use
here::here() instead.
```

The proactive features are activated immediately after loading the fertile package and require no additional effort by the user.

In addition to the interactive warning system, fertile provides several useful utility functions. 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.

shadecolorrgb0.969, 0.969, 0.969fgcolor is$_{p}ath_{p}ortable$(” ”)

```
## [1] FALSE
```

is$_{d}ata_{f}ile$(” $/Desktop/my_{d}ata.csv$”)

```
## [1] TRUE
```

3.2 Retroactive use

Retroactively, fertile analyzes potential obstacles to reproducibility in an RStudio Project (i.e., a directory that contains an .Rproj file), including the directory structure, the analyst's use of file paths, randomness, etc. fertile creates reproducibility reports that identify potential mistakes and provide recommendations for remedies. For example, fertile might identify the use of randomness in code and recommend setting a seed.

Users can access the majority of fertile's retroactive features through two primary functions.

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. Subsets of the fifteen tests can be invoked using the select helper functions from dplyr and the proj_check_some() function.

shadecolorrgb0.969, 0.969, 0.969fgcolor library(tidyverse) proj$_{c}heck_{s}ome$(”.”,*contains*(”*paths*”))

```
warningcolor## Warning in
grep("^(LaTeX|Package [[:alnum:]]+)
Warning:", x): input string 461
## is invalid in this locale
warningcolor## Warning in
grep("^(LaTeX|Package [[:alnum:]]+)
Warning:", x): input string 462
## is invalid in this locale
warningcolor## Warning in
grep("^(LaTeX|Package [[:alnum:]]+)
Warning:", x): input string 463
## is invalid in this locale
warningcolor## Warning in
grep("^(LaTeX|Package [[:alnum:]]+)
Warning:", x): input string 464
## is invalid in this locale
warningcolor## Warning in
grep("^(LaTeX|Package [[:alnum:]]+)
Warning:", x): input string 465
## is invalid in this locale
warningcolor## Warning in grep("^\\s*$",
x): input string 461 is invalid in this
locale
warningcolor## Warning in grep("^\\s*$",
x): input string 462 is invalid in this
locale
warningcolor## Warning in grep("^\\s*$",
x): input string 463 is invalid in this
locale
warningcolor## Warning in grep("^\\s*$",
x): input string 464 is invalid in this
locale
warningcolor## Warning in grep("^\\s*$",
```

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

```
x):  input string 465 is invalid in this
locale
warningcolor## Warning in
grep("^(LaTeX|Package [[:alnum:]]+)
Warning:", x):  input string 422
## is invalid in this locale
warningcolor## Warning in
grep("^(LaTeX|Package [[:alnum:]]+)
Warning:", x):  input string 423
## is invalid in this locale
warningcolor## Warning in
grep("^(LaTeX|Package [[:alnum:]]+)
Warning:", x):  input string 424
## is invalid in this locale
warningcolor## Warning in
grep("^(LaTeX|Package [[:alnum:]]+)
Warning:", x):  input string 425
## is invalid in this locale
warningcolor## Warning in
grep("^(LaTeX|Package [[:alnum:]]+)
Warning:", x):  input string 426
## is invalid in this locale
```

```
##  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 informations 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.

3.3 Sample Use Cases

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.

5 Conclusion

fertile is an R package that lowers barriers to reproducible data analysis projects in R. The features of fertile can be accessed almost effortlessly, making it easy for data analysts of all skill levels and backgrounds to gain a better understanding of how to make their work reproducible. fertile is designed to educate the user about why the mistakes are problematic and how to fix them, promoting a greater understanding of reproducibility concepts in its users. It is written for a wide audience, simple enough to be used by students in an introductory data science course, but still helpful to experienced analysts. fertile also addresses a human challenge of reproducibility. 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

[Peng2009] Roger D. Peng. 2009. Reproducible research and Biostatistics. *Biostatistics*, 10(3):405–408, 07.

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