The Journal of Open Source Software

Rclean: A Tool for Writing Cleaner, More Transparent Code

Matthew K. Lau[1], **Thomas F. J.-M. Pasquier**[2, 3], **and Margo Seltzer**[4]

1 Harvard Forest, Harvard University **2** Department of Computer Science, University of Bristol **3** School of Engineering and Applied Science, Harvard University **4** Department of Computer Science, University of British Columbia

DOI: 10.21105/joss.01312

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @danielskatz

Submitted: 07 February 2019
Published: 16 February 2020

Introduction

The growth of programming in the sciences has been explosive in the last decade. This has facilitated the rapid advancement of science through the agile development of computational tools. However, concerns have begun to surface about the reproducibility of scientific research in general (Baker, 2016) and the potential issues stemming from issues with analytical software (Stodden, Seiler, & Ma, 2018). Specifically, there is a growing recognition across disciplines that simply making data and software "available" is not enough and that there is a need to improve the transparency and stability of scientific software (Pasquier et al., 2018).

At the core of the growth of scientific computation, the `R` statistical programming language has grown exponentially to become one of the top ten programming languages in use today. At its root R is a *statistical* programming language. That is, it was designed for use in analytical workflows, and the majority of the R community is focused on producing code for idiosyncratic projects that are *results* oriented. Also, R's design is intentionally at a level that abstracts many aspects of programming that would otherwise act as a barrier to entry for many users. This is good in that there are many people who use R with little to no formal training in computer science or software engineering, but these same users can also be frequently frustrated by code that is fragile, buggy, and complicated enough to quickly become obtuse even to the authors. The stability, reproducibility, and re-use of scientific analyses in R would be improved by refactoring, which is a common practice in software engineering (Martin, 2009). From this perspective, tools that can lower the time and energy required to refactor analytical scripts and otherwise help to "clean" code, but abstracted enough to be easily accessible, could have a significant impact on scientific reproducibility across all disciplines (Visser et al., 2015).

To provide support for easier refactoring in R, we have created `Rclean`. The `Rclean` package provides tools to automatically reduce a script to the parts that are specifically relevant to a research product (e.g., a scientific report, academic talk, research article, etc.) Although potentially useful to all R coders, it was designed to ease refactoring for scientists who use R but do not have formal training in software engineering.

Methods

The goal of `Rclean` is to provide a set of tools that help someone reduce and organize code based on results. More often then not, when someone is writing an R script, the intent is to produce a set of results, such as a statistical analysis, figure, table, etc. This set of results is always a subset of a much larger set of possible ways to explore a dataset, as there are many statistical approaches and tests, let alone ways to create visualizations and other representations of patterns in data. This commonly leads to lengthy, complicated scripts

from which researchers manually subset results, but likely never to be refactored because of the difficulty in disentangling the code needed to produce some results and not others. The 'Rclean' package uses an automated technique based on data provenance to analyze existing scripts and provide ways to identify and extract code to produce a desired output.

Data Provenance

All of these processes rely on the generation of data provenance. The term provenance means information about the origins of some object. Data provenance is a formal representation of the execution of a computational process (https://www.w3.org/TR/prov-dm/), to rigorously determine the the unique computational pathway from inputs to results (Carata et al., 2014). To avoid confusion, note that "data" in this context is used in a broad sense to include all of the information generated during computation, not just the data that are collected in a research project that are used as input to an analysis. Having the formalized, mathematically rigorous representation that data provenance provides guarantees that the analyses that `Rclean` conducts are theoretically sound. Most importantly, because the relationships defined by the provenance can be represented as a graph, it is possible to apply network search algorithms to determine the minimum and sufficient code needed to generate the chosen result in the `clean` function.

There are multiple approaches to collecting data provenance, but `Rclean` uses "prospective" provenance, which analyzes code and uses language specific information to predict the relationship among processes and data objects. `Rclean` relies on a library called `CodeDepends` to gather the prospective provenance for each script. For more information on the mechanics of the `CodeDepends` package, see (Temple Lang, Peng, Nolan, & Becker, 2020). To get an idea of what data provenance is, take a look at the `code_graph` function. The plot that it generates is a graphical representation of the prospective provenance generated for `Rclean` .

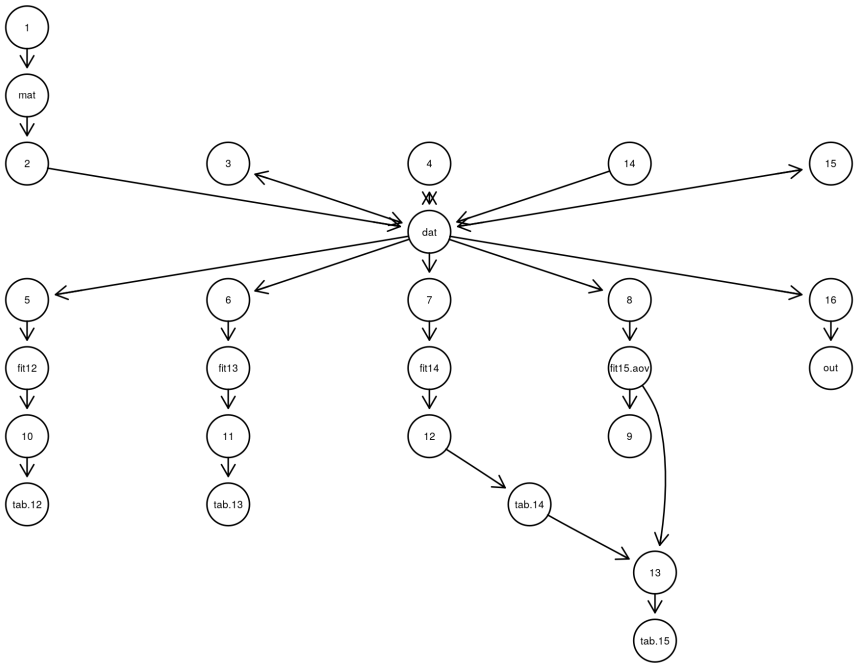

Figure 1: Network diagram of the prospective data provenance generated for an example script. Arrows indicate which lines of code (numbered) produced which objects (named).

In the future, it would also be useful to extend the existing framework to support other provenance methods. One such possibility is *retrospective provenance*, which tracks a computational process as it is executing. Through this active, concurrent monitoring, retrospective provenance can gather information that static prospective provenance can't. Greater details of the computational process would enable other features that could address some challenges, such as processing information from comments, parsing control statements, and replicating random processes. However, using retrospective provenance comes at a cost. In order to gather it, the script needs to be executed. When scripts are computationally intensive or contain bugs that stop execution, then retrospective provenance can not be obtained for part or all of the code. Some work has already been done in the direction of implementing retrospective provenance for code cleaning in R (see http://end-to-end-provenance.github.io).

Software Availability

The software is currently hosted on GitHub, and we recommend using the `devtools` library (Wickham, Hester, & Chang, 2019) to install directly from the repository (https://github.com/ROpenSci/Rclean). The package is open-source and welcomes contributions. Please visit the repository page to report issues, request features or provide other feedback.

Discussion

We see promise in connecting `Rclean` with other clean code and reproducibility tools. One example is the `reprex` package, which provides a simple API for sharing reproducible examples (Bryan, Hester, Robinson, & Wickham, 2019). Another possibility is to help transition scripts to function, package and workflow creation and refactoring via toolboxes like `drake` (Landau, 2020). `Rclean` could provide a reliable way to extract parts of a larger script that would be piped to a simplified reproducible example, in the case of `reprex`, or, since it can isolate the code from inputs to one or more outputs, be used to extract all of the components needed to write one or more functions that would be a part of a package or workflow, as is the goal of `drake`. To conclude, we hope that `Rclean` makes writing scientific software easier for the R community. We look forward to feedback and help with extending its application, particularly in the area of reproducibility. To get involved, report bugs, suggest features, please visit the project page.

Acknowledgments

This work was improved by discussions with ecologists at Harvard Forest and through the helpful review provided by the ROpenSci community, particularly Anna Krystalli, Will Landau, and Clemens Schmid. Much of the work was funded by US National Science Foundation grant SSI-1450277 for applications of End-to-End Data Provenance.

References

Baker, M. (2016). 1,500 scientists lift the lid on reproducibility. *Nature*, *533*, 452–454. doi:10.1038/533452a

Bryan, J., Hester, J., Robinson, D., & Wickham, H. (2019). *reprex: Prepare Reproducible Example Code via the Clipboard*. Retrieved from https://CRAN.R-project.org/package=reprex

Carata, L., Akoush, S., Balakrishnan, N., Bytheway, T., Sohan, R., Seltzer, M., & Hopper, A. (2014). A Primer on Provenance. *Queue*, *12*(3), 10–23. doi:10.1145/2602649.2602651

Landau, W. M. (2020). *drake: A Pipeline Toolkit for Reproducible Computation at Scale*. Retrieved from https://CRAN.R-project.org/package=drake

Martin, R. (2009). *Clean code: a handbook of agile software craftsmanship* (p. 431).

Pasquier, T., Lau, M. K., Han, X., Fong, E., Lerner, B. S., Boose, E. R., Crosas, M., et al. (2018). Sharing and Preserving Computational Analyses for Posterity with encapsulator. *Computing in Science & Engineering*, *20*(4), 111–124. doi:10.1109/MCSE.2018.042781334

Stodden, V., Seiler, J., & Ma, Z. (2018). An empirical analysis of journal policy effectiveness for computational reproducibility. *Proceedings of the National Academy of Sciences of the United States of America*, *115*(11), 2584–2589. doi:10.1073/pnas.1708290115

Temple Lang, D., Peng, R., Nolan, D., & Becker, G. (2020). *CodeDepends: Analysis of R Code for Reproducible Research and Code Comprehension*. Retrieved from https://github.com/duncantl/CodeDepends

Visser, M. D., McMahon, S. M., Merow, C., Dixon, P. M., Record, S., & Jongejans, E. (2015). Speeding Up Ecological and Evolutionary Computations in R; Essentials of High Performance Computing for Biologists. (F. Ouellette, Ed.)*PLOS Computational Biology*, *11*(3), e1004140. doi:10.1371/journal.pcbi.1004140

Wickham, H., Hester, J., & Chang, W. (2019). *devtools: Tools to Make Developing R Packages Easier*. Retrieved from https://CRAN.R-project.org/package=devtools