Reproducing "A large-scale study on research code quality and execution"

Samuel Grayson Darko Marinov Reed Milewicz
Daniel S. Katz

2023-05-03

Replicating "A large-scale study on research code quality and execution"

Abstract

We replicate dataset from "A large-scale study on research code quality and execution" by Trisovic et al. using our own experimental infrastructure. Our results agree in part, disagree in part, and extend the original results. We designed our experimental infrastructure to be extensible for other kinds of replication studies, where one wants to run a set of codes with a certain interpreter and examine the results. WIth insight from the original work and its replication, we attempted to make our infrastructure open-source and reproducible on a large variety of paltforms.

Introduction

Note that this manuscript reads properly on printed media, but the embedded hyperlinks provide references to internet resources for an interested reader.

Reproducibility is essential for scientists to scrutinize each other's results and to build form prior work. With the advent of computational experiments in science, it is natural to ask what fraction of those computational experiments are reproducible.

Authors set out to answer this question for specific communities in the past [4]. Of these, Collberg et al. [1], Stodden et al. [2], Zhao et al. [3], and Trisovic et al. [4] are especially interesting because they study codes used in basic research. Of these, Zhao et al. [3] and Trisovic et al. [4] are automatic re-execution studies of codes found in repositories that have the explicit purpose of encouraging reproducibility for scientific codes. Zhao et al. studied the defunct myExperiment repository while Trisovic et al. studied Harvard Dataverse. As such, we sought to reproduce Trisovic et al. Not only will reproduction validate the data, but also future research on automatic code cleaning of R scripts could begin by modifying Trisovic et al.'s original experiments.

According to the latest ACM Badging Terms [5], "reproduction" is defined as a different team using the same experimental setup, while "replication" is a different team using a different experimental setup. This work constitutes a reproduction; we will develop our own experimental infrastructure to ask and answer the same questions.

In Section 2, we describe Dataverse and the original work. In Section 3, we explain our methodology, especially the ways it differs from the prior work. In Section 4, we present our results, using the original work's statistical procedure and novel statistical procedures. In Section 5, we discuss the significance of the results.

Original work

In this section, we describe Dataverse, the methodology of the original work, and the results of it.

Dataverse is a "software platform for sharing, finding, citing, and preserving research data" [6], and *Harvard Dataverse* is a large instance of Dataverse. Scientists can upload the code or data used in a publication, where this bundle is called a *dataset* in Dataverse. Datasets can be modified, where each modification gets a new two-number version code (e.g., “1.0”). Each dataset gets a DOI, which points to the latest version of that dataset.

In “A large-scale study on research code quality and execution” [4] (henceforth, “the original work”), Trisovic et al. studied codes in Harvard Dataverse according to the following pseudocode:

```
1. Identify datasets uploaded to Harvard Dataverse that contain R scripts.
2. For each R version:
    1. Once with code-cleaning and once without:
        1. For each Dataset identified in step 1.:
            1. Run a container based on that version and code-cleaning
               with that Dataset ID
```

The container, given a Dataset ID, invokes a runner script which does the following:

```
1. Download the information for the Dataset.
2. For each file in listed in the Dataset information:
    1. Download the file
    2. Check that the hash of the downloaded file is equal to the expected
       hash in the Dataset information.
    3. If the hash matches, the attempt was successful, otherwise try an-
       other attempt.
3. If code-cleaning was enabled at container-build time:
    1. For each R file we downloaded:
        1. Run the code-cleaning operation.
4. Install packages in R needed for the analysis.
5. For each R file we downloaded:
    1. Try to run the R file.
    2. If the R script fails, classify its error.
```

    3. If the error was one of two specific kinds:
        1. Rerun the R file a different way.
    4. Write results to a log file
6. Upload log files to a database.

Here we examine the last five (of eight) research questions from the original work. We prioritized these research questions as they relate directly to reproducibility rates.

- **RQ4**: What is the code re-execution rate?
    - The original work gives the following table:

Table 1: Unlabeled table from the original work, page 7 of the PDF version.

	Without code cleaning	With code cleaning	Best of both
Success rate	24%	40%	56%
Successes	952	1472	1581
Errors	2878	2223	1238
Timed outs	3829	3719	5790
Total files	7659	7414	8609
Total datasets	2071	2085	2109

- **RQ5**: Can automatic code cleaning with small changes to the code aid in its re-execution?
    - The table tbl. 1 shows that "we see that code cleaning can successfully address some errors" and "there were no cases of code cleaning"breaking" the previously successful code."
- **RQ6**: Are code files designed to be independent of each other or part of a workflow?

  - “If there are one or more files that successfully re-executed in a dataset, we mark that dataset as’success’. A dataset that only contains errors is marked as ‘error’, and datasets with TLE values are removed. In these aggregated results (dataset-level), 45% of the datasets (648 out of 1447) have at least one automatically re-executable R file. There is no drastic difference between the file-level success rate (40%) and the dataset-level success rate (45%), suggesting that the majority of files in a dataset are meant to run independently. However, the success rate would likely be better had we known the execution order.”

- **RQ7**: What is the success rate in the datasets belonging to journal Dataverse collections?

  - The original work “find[s] a slightly higher than average re-execution rate (42% and aggregated 47% instead of 40% and 45%)... Our results suggest that the strictness of the data sharing policy is positively correlated to the re-execution rate of code files”

- How do release dates of R and datasets impact the re-execution rate?

  - “We observe that R 3.6 has the highest success rate per year.”
  - “We are unable to draw significant conclusions on whether the old code had more or fewer errors compared to the recent code (especially considering the sample size per year).”

- **RQ8**: What is the success rate per research field?

  - “The highest re-execution rates were observed for the health and life sciences.”

Methodology

In this section, we describe the problems we have observed with the prior work’s methodology, and present our own methodology.

First, we attempted a replication of the original work. However, we found that the original work had some internal discrepancies and some non-reproducible aspects.

Internal discrepancies (ID) include:

- **ID1:** The raw data from the original work contains a different set of datasets (DOIs) for each experimental condition (see Appendix IIB for proof). {#tbl:original} also shows this problem, because there are a different number of "Total datasets" for different verisons of R. The choice of datasets-to-test is a controlled variable and should held constant when changing experimental conditions, such as versions of R. One possible reason for the change is that the runner script can silently crash, and if it does, it will not output results for files after that point (see Appendix IA for more details). We could not find a way to tell if the runner script crashed from the released data. Changes in the script between versions (see **ID3**) cause these omissions to be inconsistent between versions.

- **ID2:** The runner has two ways of running an R file, 5.1 and 5.3.1 in the above pseudocode. The images uploaded to Dockerhub change the version of R for step 5.1 but not 5.3.1 (see Appendix IIC for proof). The scripts that fail 5.1 will be rerun in 5.3.1 with the wrong version of R.

- **ID3:** One Docker image must be prepared for each experimental condition, because the image encodes the experimental condition. All things not related to that experimental condition should remain unchanged in the images. The images have different versions of the runner script and other important files (see Appendix IIA and IID for proof).

- **ID4:** The R 3.2 and 3.6 environments request `r` while the R 4.0 environment requests `r-base` (see the original work's Dockerfile). `r` depends on `r-base` and `r-recommended`. Therefore, the 3.2 and 3.6 environment will have packages that the 4.0 environment does not have. Some scripts may succeed in R 3.6 and fail in R 4.0, not because of the R version, but because the difference in installed packages.

- **ID5:** There is a bug in the data-processing scripts that cause the number of scripts with “time limit exceeded” to be over-reported. See Appendix IIF for details.

Non-replicable (NR) aspects include:

- **NR1:** Datasets in Dataverse can upload a new version, which will get the same DOI (e.g., doi:10.7910/DVN/U3QJQZ); The code uses the latest version of the dataset in Dataverse (see the original work’s download_dataset.py). Re-executing the code will be analyzing different experiments. There is no record of which version the original work used, but one could filter for versions released prior to the approximate date of the experiment.

- **NR2:** The main script, which is run after the Docker image is built, calls `install.packages(...)`, which installs the latest version of a package at the time of its execution. This results in a different software environment in the replication than that in the original work. In particular, one line of the runner script, `install.packages(reticulate)`, fails when we attempted it, although it presumably worked in the original work’s execution in 2020. This is package is installed during the execution of the main script *after* the image is built, so using the images from Dockerhub will not help this.

- **NR3:** Conda may have hosted R 3.2.1 at some point, but it currently does not (see the Conda R repository and Conda Forge repository). Therefore, the R environment used by the original work cannot be built.

To try to avoid these issues, we decided to do a reproduction instead of a replication.

- We run all of the DOIs under every version of R, avoiding **ID1**. We run all scripts in the dataset even if one fails.

- The original work used the R command, `source(r_file)`, and fell back on the shell command `Rscript r_file` when that failed. TODO: why? We only use `Rscript r_file`. Furthermore, the container for R 4.0 does not contain R 3.6 at all, so there is no chance of confusing versions of R in that container, avoiding **ID2**.
- We re-executed the code without changing it for all experimental conditions avoiding **ID3** and **ID4**.
- Our project logs the version of the code that it uses, avoiding **NR1**.
- Our containers are built using the Nix package manager, which pins the exact version of every dependency used, avoiding **NR2**.
- Nix can archive all of the sources it pulls from into a replication package, which avoids **NR3**.

The original work uses AWS Batch and AWS DynamoDB, which makes replication more difficult on other platforms. For this work, we used Dask [7] to interface to our Azure VMs, and filesystem_spec [8] to interface with Azure storage buckets. As long as a user can set up a Dask cluster, they can easily import and execute our code. Meanwhile, filesystem_spec abstracts the local file system, SSH FS, AWS S3, Azure Storage, and several others.

Our Docker container contains the specified R version, the recommended R packages (Ubuntu and Anaconda both agree on this set), Busybox utilities, GNU Make, and GNU Time.

While we wanted to use the exact versions of R used by the original work, Nix does not have 4.0.1, so we used 4.0.2; there is a bug with Nix's version of R MASS in 3.2.1, so we used 3.2.3.

Results

We began with the 2170 datasets listed in the original work's master list, get-dois/dataset_dois.txt. Note that the results only contain a subset of these (see **ID1**).

There is a confirmed issue with Harvard Dataverse where the MD5 hash it displays does not match the file that is available for downloading [1] [2]. This issue affects 18/2170 ~ 1.5% of the datasets, which we removed from consideration. The original work also excluded these.

We rephrased the unlabeled table on Page 4[3] to use proportions relative to the total, and we augmented it with our results:

The biggest difference is that there are many fewer timed-out scripts. Both this work and the original work applies a five-hour time-limit to the collection of scripts as a whole and a one-hour time-limit to each individual script (the original work only applies the per-script limit in step 5.1, not in 5.3.1 of the pseudocode; see **ID2** for details). It would be surprising if half of the scripts in Harvard Dataverse ran for longer than hour, or five hours combined. We have fewer timeouts because the original work was affected by **ID5**, which overreported the number of timeouts. Another reason is that the original work does two calls to `install.packages`, which involve compiling C packages for R, quite an expensive operation. Our runner does not need this `install.packages` call at all, so it does less work against the time-limit.

Many scripts will call `install.packages`, which will install the latest version of a package. However, the latest version is not necessarily compatible with the running version of R. `install.packages` does not attempt to solve dependency conflicts. For example, the latest version, 3.4.2, of popular plotting package ggplot2 requires R 3.3 or newer. When Trisovic et al. ran their experiments in 2020, this may not have been the case.

We also exclude datasets selected by the original work that do not contain any R scripts. They may have contained R scripts at the time that the experiments in the original work were executed.

Data analysis differences:

[1] Issue #9501 on dataverse GitHub

[2] Issue #37 on dataverse.harvard.edu GitHub

[3] The percentage in this row is not analogous to the "success rate" in the original work. The original work defines "success rate" in their table as the number of successful scripts divided by the number of successes plus failures. The percentage here is the number of successful scripts divided by the total. Their definition ignores timed out scripts; if we adopt their definition, then our numbers do not change appreciably (we few timeouts), but their success rate is 24%.

- Apply statistical tests applied to assess significance.
- Note that Trisovic et al. defines success rate as the ratio of success to success plus errors (i.e., excluding timed out codes).

Discussion

Can we reproduce the main claims of the paper? * How close can we come? * Do the differences matter?

What was easy

What was difficult

Communication with the original authors

What would we do differently if we were redoing our reproduction from scratch?

(or maybe this is part of the first two previous subsections?)

Conclusion

Trisovic et al. establish an important and seminal dataset for large-scale reproducibility studies of modern scientific codes. We seek to reproduce their work in a way that avoids some internal discrepancies and non-replicability problems. Our results partially confirm and partially revise the original work. We hope that this dataset and platform for assessing reproducibility serves future work in reproducibility and automatic bug fixing.

Lessons learned

How to make more things reproducible? * What would we have liked the original authors to have done differently then? * What would the original authors likely do differently now (based on newer understanding/practices/tools/etc)?

Appendix I: Instructions for reproducing

Source code archive

Appendix II: Bugs in the Trisovic runner script

A: Bug with `source(...)` causes runner to crash

Many Dataverse R scripts contain `rm(list=ls())`. This is commonly recommended to clean the R environment [9]. Unfortunately, this gets `source(..)`ed by `exec_r_files.R`. Note that `rm(...)` and `ls(...)` remove or list variables in the R's global namespace. Then `exec_r_files.R` will fail because it cannot find its own variables.

For example, `doi:10.7910/DVN/XY2TUK` contains two R scripts, `BasSchub_ISQ_Shocks_Figures.R` which contains `rm(list=ls())` and `BasSchub_ISQ_Shocks_ClusterRobust.R` which does not. The scripts execute in alphabetical order, so `BasSchub_ISQ_Shocks_ClusterRobust.R` gets executed and put in the results. However, the `BasSchub_ISQ_Shocks_Figures.R` causes a crash and will not appear in the results, for the experiments which have this bug.

3.6-no-cleaning has only the first, so it has this bug (see `data/run_log_r32_no_env.csv` line 3161). On the other hand, 4.0-no-cleaning does not have this bug, so it will have results both scripts (see `data/run_log_r40_no_env.csv line` 5259). This explains some of the discrepancies in **ID1** and is further evidence for **ID3**.

B: Different DOIs and files

The following Python session, which you can run for yourself, shows that different DOIs are reported between experimental conditions, which supports **ID1**

```
>>> import requests, csv
>>> url_prefix = "https://raw.githubusercontent.com/atrisovic/dataverse-r-study/master"
>>> all_dois = requests.get(f"{url_prefix}/get-dois/dataset_dois.txt").text.strip().split("\n")
>>> def get_dois(data_file):
```

```
...     request = requests.get(f"{url_prefix}/analysis/data/{data_file}")
...     reader = csv.reader(request.text.split("\n"), delimiter="\t")
...     return set(row[0] for row in reader if len(row) > 1)
>>> dois32 = get_dois("run_log_r32_env_download.csv")
>>> dois36 = get_dois("run_log_r36_env_download.csv")
>>> len(dois32 - dois36) # dois in r32 but not in r36
2
>>> len(dois36 - dois32) # dois in r36 but not in r32
282
```

This may be caused by: * transient download errors for one R version but not another; * cases where `./download_datasets.py` executes partially, but is interrupted by a timeout from line 17 of `run_analysis`. Note that while other parts of the script ignore some kinds of download errors, they will not ignore this kind of download error for the reasons explained in the last bullet of Appendix IIF. * cases where the runner crashes in one R version but not another (one such case is described in Appendix IIA). The return code of the runner is not reported in the original work's dataset, so there is no way to tell on how many cases the runner crashed.

These differences between R versions are exacerbated by different versions of the script, and they make it difficult to compare results across R versions.

C: Mismatched versions of R

`Dockerfile` line 24 executes `conda activate ...` to set the R version for `exec_r_files.R` line 30, which `source(...)`es the R script under test.

If that fails, `exec_r_files.R` line 72 calls execute_files.py line 5, which invokes `Rscript` from a hard-coded path. This hard-coded path needs to be the proper R version as the former for this method to yield consistent results, but in the published containers, it was not. This Bash session, which you can run for yourself, shows that the image called `aws-image-r32` used 3.2 for the former, but 4.0 for the latter method, confirming **ID2**.

```
$ docker run --rm -it --entrypoint bash atrisovic/aws-image-r32 -c 'cat ~/.bashrc' | grep 'conda activate r_'
conda activate r_3.2.1
$ docker run --rm -it --entrypoint bash atrisovic/aws-image-r32 -c 'cat execute_files.py' | grep 'Rscript'
    p3 = Popen(['/opt/conda/envs/r_4.0.1/bin/Rscript', f], \
```

D: Different versions of the script

We can use Diffoscope [10] to examine the differences between containers. This Bash session, which you can run for yourself, shows that the images `r32` and `r40` have significant differences in the runner script which contributes to **ID1** and **ID3**.

```
$ docker pull atrisovic/aws-image-r32
$ docker save --output aws-image-r32.tar atrisovic/aws-image-r32
$ docker pull atrisovic/aws-image-r40
$ docker save --output aws-image-r40.tar atrisovic/aws-image-r40
$ docker run --rm -t -w $PWD -v $PWD:$PWD:ro registry.salsa.debian.org/reproducible-builds/diffoscope *.tar
shows different versions of usr/workdir/exec_r_files.R and other important files
```

E: Original image does not build due to out-of-date Debian version

`apt-get update` in the original work's `Dockerfile` line 3 fails. There has been a new major release of Debian since the time of experimentation. When there is a new release, the source repository changes its information. `apt-get` refuses to update when the information is changed from the last time it was run. Changing the line to `apt-get update --allow-releaseinfo-change` allows us to bypass this check. Presumably the original work intended to use `continuumio/miniconda3` as a base image, but they use `continuum/miniconda` instead, which has not been updated in three years, since before the last Debian major release.

F: Time limit exceeded is overcounted

The Jupyter Notebook `02-get-combined-success-rates.ipynb` reads tables where each row is a script, and each column has the result of running that file in a specific R version. The notebook aggergates the results together, according to rules such as: - If a script succeeds in any R version, it is considered to succeed. - If a script times out in any R version and none succeed, it is considered to time out. - Otherwise, the script is considered to error out.

However there is a bug in that notebook:

1. The notebook `02-get-combined-success-rates.ipynb`, uses an "outer join". Therefore, if a script is absent from one of the R-versions, it will receive NaN in that cell. This occurs in the cells labeled "In[7]" and "In[9]" for the case with code-cleaning, and is duplicated to "In[30]" and "In[32]" for the case without code-cleaning.

For the sake of example, suppose we have the following results (on different scripts).

R 3.2 results:

File	Result
script0.R	error

R 3.6 results:

File	Result
script1.R	error

Then the outer merge of R 3.2 and 3.6 would be:

File	R 3.2	R 3.6
script0.R	NaN	error
script1.R	error	NaN

2. The cell labeled "In[11]" and "In[34]" assigns the aggregated-result NaN if none of the individual-results are NaN or time-limit-exceeded.
3. The cells labeled "In[17]" and "In[38]" remove datasets that fail to download, but there are other reasons we will discuss below that can cause a script result to be missing.
4. The cell labeled "In[23]" and "In[43]" save the result of the preceding steps as `data/aggregate_results_env.csv`.
5. The cell labeled "In[2]" and "In[5]" of 03-success-dataset-list.ipynb loads this file, `data/aggregate_results_env.csv`.

6. The cell labeled "In[4]" and "In[7]" of 03-success-dataset-list.ipynb counts the number of NaNs in the result column. This number matches exactly what is put in the unlabeled table on page 7 of the prior work's PDF, so we assume this is their provenance.

If the only way that data can be missing is for a time out, then this approach would be valid. However, there are other ways that the data may be missing:

- If the runner crashes, it will not write results for some of the scripts. We found one condition which causes the runner to crash in Appendix IIA.
- When the time limit is exceeded, `run_analysis.sh` line 36 writes a record saying saying that a file named "unknown" timed out. When the the script exits with an error, `exec_r_files.R` line 88 will write t he actual name of the script and the error. Therefore, the outer-join will have two rows indicating a timed out result where it should only have one:

File	R 3.2	R 3.6	Result
unknown	NaN	timed out	NaN
script.R	error	NaN	NaN

- line 17 of `run_analysis.sh` downloads the dataset from Dataverse with a timeout. When this timeout is hit, line 20 writes a record into `run_log.csv` saying that a file named "unknown" experienced an error called "download error". Like in the previous case, "unknown" will not exist if another R version succeeds in downloading that dataset, which happens quite often in the dataset. While `02-get-combined-success-rates.ipynb` *does* exclude download errors found in `run_log_ds.csv`, it does not check for download errors in `run_log.csv`, so it will not exclude this DOI.

[1] C. Collberg and T. A. Proebsting, “Repeatability in computer systems research,” *Commun. ACM*, vol. 59, no. 3, pp. 62–69, Feb. 2016, doi: 10.1145/2812803. [Online]. Available: https://dl.acm.org/doi/10.1145/2812803. [Accessed: May 27, 2022]

[2] V. Stodden, J. Seiler, and Z. Ma, “An empirical analysis of journal policy effectiveness for computational reproducibility,” *Proceedings of the National Academy of Sciences*, vol. 115, no. 11, pp. 2584–2589, Mar. 2018, doi: 10.1073/pnas.1708290115. [Online]. Available: https://www.pnas.org/doi/full/10.1073/pnas.1708290115. [Accessed: Jan. 19, 2023]

[3] J. Zhao *et al.*, “Why workflows break — understanding and combating decay in Taverna workflows,” in *2012 IEEE 8th International Conference on E-Science (e-Science)*, Oct. 2012, p. 9, doi: 10.1109/eScience.2012.6404482 [Online]. Available: https://www.research.manchester.ac.uk/portal/en/publications/why-workflows-break--understanding-and-combating-decay-in-taverna-workflows(cba81ca4-e92c-408e-8442-383d1f15fcdf)/export.html

[4] A. Trisovic, M. K. Lau, T. Pasquier, and M. Crosas, “A large-scale study on research code quality and execution,” *Sci Data*, vol. 9, no. 1, p. 60, Feb. 2022, doi: 10.1038/s41597-022-01143-6. [Online]. Available: https://www.nature.com/articles/s41597-022-01143-6. [Accessed: Dec. 13, 2022]

[5] A. Inc. staff, “Artifact Review and Badging.” Aug. 2020 [Online]. Available: https://www.acm.org/publications/policies/artifact-review-and-badging-current. [Accessed: Jan. 19, 2023]

[6] D. community developers and D. S. and P. team at the Institute for Quantitative Social Science, “Dataverse®.” Institute for Quantitative Social Science, Apr. 2023 [Online]. Available: https://github.com/IQSS/dataverse. [Accessed: Apr. 21, 2023]

[7] M. Rocklin, “Dask: Parallel Computation with Blocked algorithms and Task Scheduling,” 2015, pp. 126–132, doi: 10.25080/Majora-7b98e3ed-013 [Online]. Available: https://conference.scipy.org/proceedings/scipy2015/matthew_rocklin.html. [Accessed: May 03, 2023]

[8] Fs. authors, “Filesystem_spec.” May 2023 [Online]. Available: https://github.com/fsspec/filesystem_spec. [Accessed: May 03, 2023]

[9] G. Vanpelt, “How to Clear the Environment in R with the rm(list=ls()) Command,” *Lxadm.com*. Jan. 2023 [Online]. Available: https://lxadm.com/r-rm-list-ls/. [Accessed: May 03, 2023]

[10] M. Glukhova, “Tools for Ensuring Reproducible Builds for Open-Source Software,” Master’s thesis, Lappeenranta University of Technology, 2017 [Online]. Available: https://lutpub.lut.fi/bitstream/handle/10024/135304/MariaGlukhova_ToolsForEnsuringReproducibleBuildsForOpenSourceSoftware.pdf?sequence=2