Replication materials are great but software versioning still poses a problem for open science

Taylor J. Wright[*] Bruno Rodrigues [†]

August 11, 2023

[*]Brock University, twright3@brocku.ca
[†]Ministry of Higher Education and Research, Luxembourg

1 Introduction

Beginning in the early 2010s with psychology and quickly spilling over into political science and economics on the backs of high profile cases of fraud (Broockman, Kalla, and Aronow (2015)) and errors (Herndon, Ash, and Pollin (2014)), the "replication crisis" in the social sciences has become mainstream, even finding its way into the popular press (Aschwanden (2015), Gelman (2018)). Concerns over how well published results hold up to replication in other settings has lead to large scale initiatives to document how trustworthy the existing stock of evidence is (e.g. Camerer et al. (2016), Chang and Li (2022), and Collaboration (2015)). While the *replicability* of studies has recieved much attention, there has been a tandem movement discussing the importance of and need for research *reproducibility*. While the exact scope and definition is subject to ongoing debate, replication is, broadly speaking, in some sense re-testing a hypothesis while changing an element of previous research (e.g. the sample or estimating equation) whereas reproducibility (sometimes referred to as verification or computational reproducibility) is following a study's protocol or methods exactly and obtaining the results presented in the study.[1] In our view, reproducibility is an insufficient but necessary condition for replication—that is, it does not make sense to spend resources on replication if research is not reproducible.

These concerns about the reproducibility (and replicability) of social science research have prompted a push, often referred to as Data Access and Research Transparency (DA-RT) in political science following Lupia and Elman (2014), for journals to require the publication of research materials that accompany academic research.[2] Specifically, the provision of underlying data and code used for data preparation and analysis. This push has seen some success, with a growing number journals now requiring the provision of replication materials as a condition of publication. Some journals, such as the American Econonmic Review (AER) or American Journal of Political Science (AJPS), even have verification policies that require authors upload their replication materials and have their results verified by another team (either the journal's replication team (AER) or a third party, (AJPS)) prior to publication. However, even if these materials are provided, and even when in place these policies do not have perfect compliance (Philip (2010), Stockemer, Koehler, and Lentz (2018)), regular software updates and new version releases can result in the replication materials failing to faithfully reproduce the authors' results

[1] For further discussion on definitions, scopes, and types of replication and reproducibility see, for example, Clemens (2017), Derksen and Morawski (2022), Nosek and Errington (2020).

[2] See, for example, the 2014 Joint Editors Transparency Statement which was signed by editors of 27 leading political science journals: https://www.dartstatement.org/2014-journal-editors-statement-jets

or even run at all. [3]

In this paper we present a case study of an article published in Journal of Politics in January 2022 titled, "Multiracial Identity and Political Preferences", Davenport, Franco, and Iyengar (2022), that details that replication challenges arising from changes in the statistical software R. We were unable to reproduce the authors' results using either the current version of R, or the version that the authors indicate they used. The lack of reproducibility arose due to a change in the defaults used by base R when generating random numbers starting in version 3.6.0.

We contribute to the existing literature… Strand 1: Discussion of importance of availablility of replication/reproducibility materials Contribution 1: A necessary but insufficient condition Strand 2: Discussion of problems raised by software and package versioning for reproducibility Contribution 2: A walk through of the problem with a concrete example Strand 3: Discussion of tools and best practices for ensuring reproducibility Contribution 3: Step-by-step guide to using Docker, {renv}, and {targets} to ensure reproducibility

The rest of the article proceeds as follows: Section 2 walks through the reproducibility issues in Davenport, Franco, and Iyengar (2022); Section 3 discusses currently available tools and best practices (e.g. Docker and R packages such as `renv`, `groundhog`) for ensuring that replication materials continue to faithfully reproduce research results, despite post-publication changes in the tools used; and Section 4 concludes.

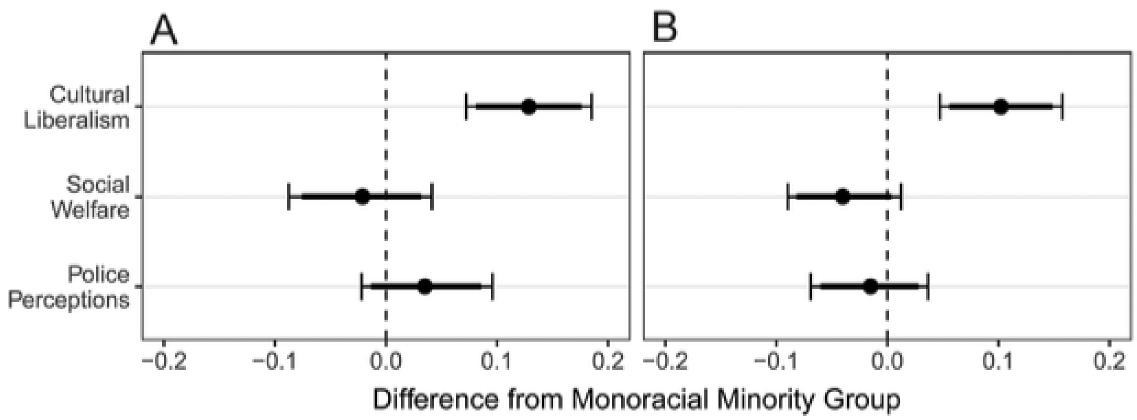
[3] Simonsohn (2021) presents several examples of R changes that could break scripts.

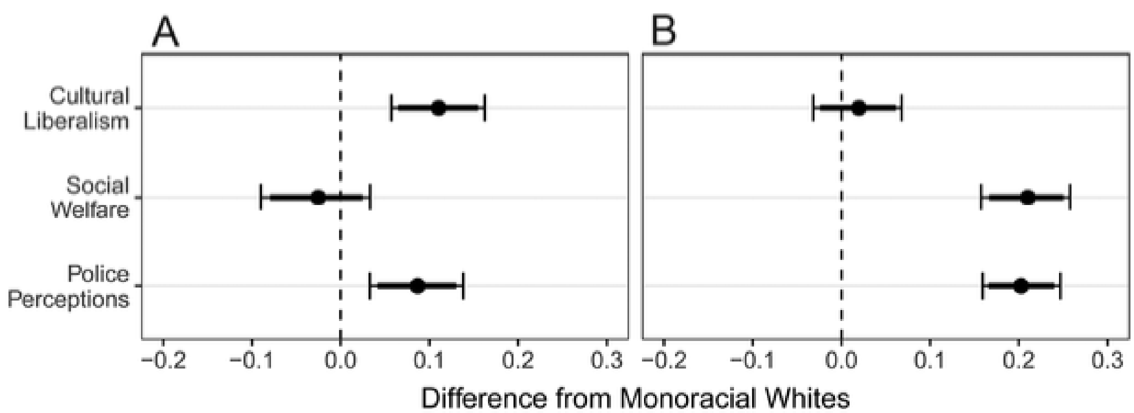
2 Reproduction

2.1 Illustrating the issue with software versioning

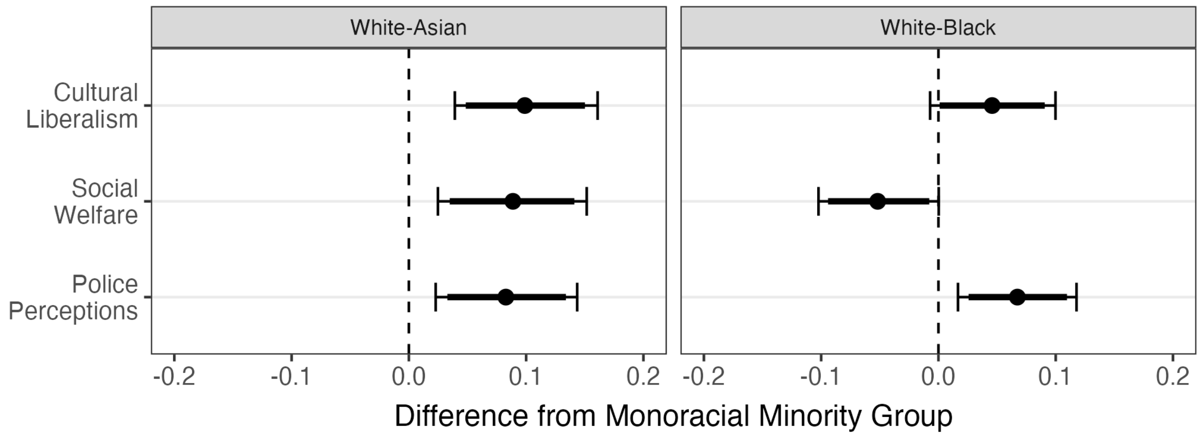
A key thing here is that I don't think we want to be too hostile sounding towards these authors, they had readme files and reproducibility materials available. It's just that manual entry human-error hobgoblins got them with the R versioning.

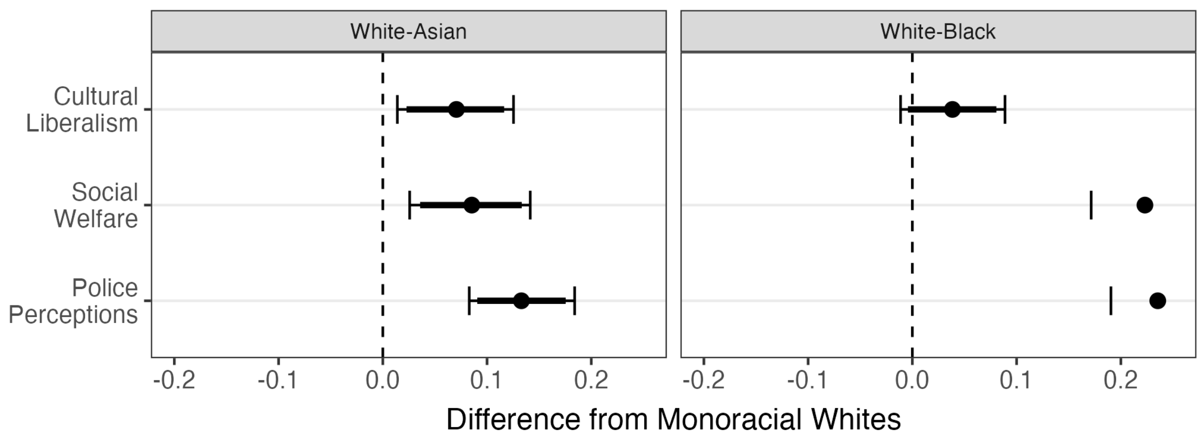
- Brief discussion of authors' paper and context
- The results of their code using stated software version in documentation (just screenshot right now)



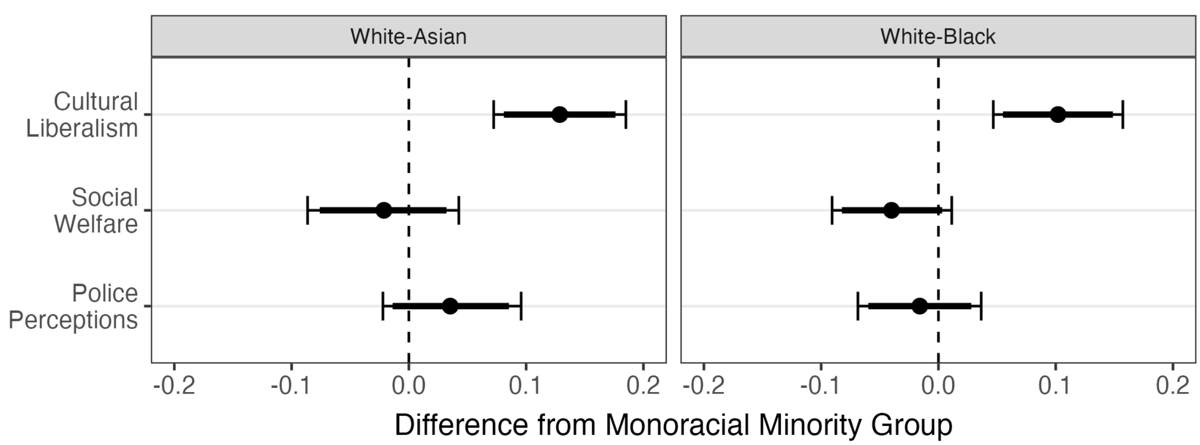


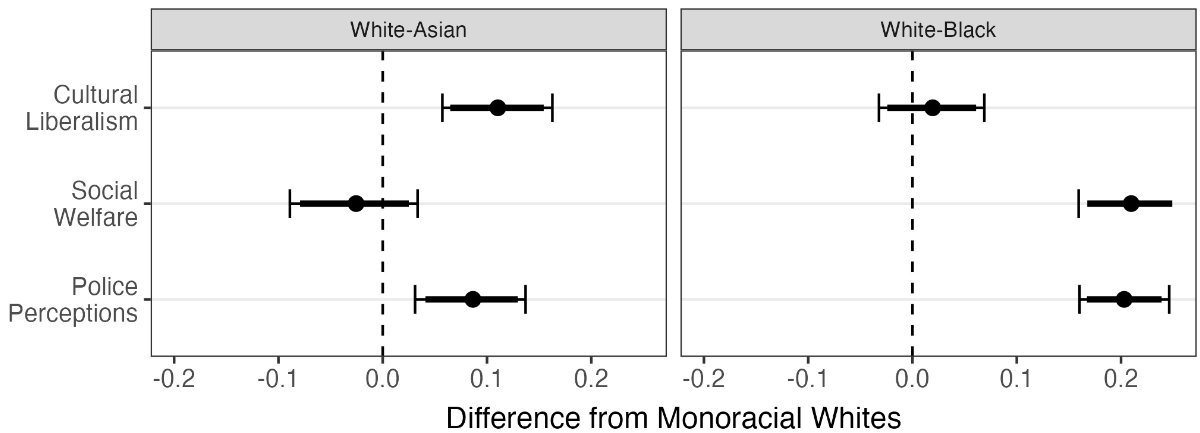
- The results of their code using later software version (post 3.6.0)





- Results of their code using earlier software version (pre 3.6.0)





The authors note that they use R version 3.6.2. Issue seems to be the weights the authors use are non-integer and Zelig uses sample() in that case which following changes to base R yields different results in 3.6.x and 3.5.x (see http://docs.zeligproject.org/articles/weights.html). Prior to R 3.6.x RNGkind(sample.kind = "Rounding") was the default behaviour but after 3.6.0 the sample function's new default behaviour is RNGkind(sample.kind = "Rejection") (see https://blog.revolutionanalytics.com/2019/05/whats-new-in-r-360.html).

Soemthing about how there are ways to ensure that the estimates are consistent (changing the weights to integers or setting the RNGkind to be backwards compatible) but documenting the correct version of R used in the analysis is probably the easiest way. Segue into…

3 Discussion

3.1 The problem is the seed?

The table below shows the quantiles of the means obtained from 100 runs with 100 different random seeds on R version 3.5 for the `m2m` models. We should check where the coefficients from the original paper fall in that distribution, and see if coefficients should vary so much simply from changing the seed?

race	model	q_05_mean	q_20_mean	q_40_mean	q_50_mean	q_60_mean	q_80_mean	q_95_mean
White-Asian	Police Perceptions	-0.02	0.00	0.02	0.03	0.03	0.04	0.07
White-Asian	Social Welfare	-0.02	0.01	0.02	0.03	0.04	0.06	0.08
White-Asian	Cultural Liberalism	0.08	0.10	0.11	0.12	0.13	0.15	0.17
White-Black	Police Perceptions	-0.02	0.00	0.02	0.02	0.03	0.04	0.07
White-Black	Social Welfare	-0.08	-0.06	-0.04	-0.04	-0.04	-0.02	0.00
White-Black	Cultural Liberalism	0.04	0.06	0.07	0.08	0.08	0.10	0.12

The table below shows the quantiles of the means obtained from 100 runs with 100 different random seeds on R version 3.5 for the `m2w` models:

race	model	q_05_mean	q_20_mean	q_40_mean	q_50_mean	q_60_mean	q_80_mean	q_95_mean
White-Asian	Police Perceptions	0.05	0.07	0.08	0.09	0.10	0.12	0.14
White-Asian	Social Welfare	0.00	0.02	0.03	0.05	0.06	0.07	0.10
White-Asian	Cultural Liberalism	0.05	0.06	0.09	0.09	0.10	0.11	0.13
White-Black	Police Perceptions	0.17	0.19	0.20	0.21	0.22	0.23	0.27
White-Black	Social Welfare	0.15	0.18	0.19	0.20	0.20	0.22	0.25
White-Black	Cultural Liberalism	-0.01	0.01	0.02	0.03	0.03	0.04	0.06

3.2 Rebuilding the original development environment using Docker

Replicating results from past studies is quite challenging, for many reasons. This article focuses on one of these reasons: results cannot be reproduced because of changes introduced in more recent versions of the software used for analysis, despite the availability of both data and replication scripts.

To replicate the results from the original study and to pinpoint the impact of the change introduced in R 3.6.0, we chose to use Docker. Docker is a containerization tool which enables one to build so-called *images*. These images contain a software product alongside its dependencies and even pieces of a Linux operating system. To use the software product, customers in turn only need to be able to run Docker *containers* instantiated from the image definition. Containerization tools such as Docker solved the "works on my machine" problem: this problem arises when software that works well on the development machine of the developer fails to run successfully on a customer's machine. This usually happens because the customer's computer does not have the necessary dependencies to run the software (which are traditionally not shipped alongside the software product) or because of version mismatch of the operating system.

A research project can also be seen as a *software product*, and suffers thus from the same "works on my machine" problem as any other type of software. Containerization tools offer a great opportunity for reproducibility in research: instead of just sharing a replication script, authors can now easily share the right version of the software used to produce these scripts, as well as the right version of the used libraries by building and sharing a Docker image (or at least provide the necessary blueprint to enable others to do so, as we will discuss below). Future researchers looking to replicate the results can now simple run a container from the provided image (or build an image themselves if the original authors provided the required blueprints).

Concretely, to build a Docker image, a researcher writes a so-called *Dockerfile*. Here is an example of a very simple Dockerfile:

```
FROM rocker/r-ver:4.3.0

CMD ["R"]
```

This Dockerfile contains two lines: the first line states which Docker image we are going to use as a base. Our image will be based on the `rocker/r-ver:4.3.0` image. The Rocker project is a repository containing many images that ship different versions of R and packages pre-installed: so the image called `r-ver:4.3.0` is an image that ships R version 4.3.0. The last line states which command should run when the user runs a container defined from our image, so in this case, simply the R interactive prompt. Below is an example of a Dockerfile that runs an analysis script:

```
FROM rocker/r-ver:4.3.0

RUN R -e "install.packages('dplyr')"

RUN mkdir /home/research_project
```

```
RUN mkdir /home/research_project/project_output

RUN mkdir /home/research_project/shared_folder

COPY analyse_data.R /home/research_project/analyse_data.R

RUN cd /home/research_project && R -e "source('analyse_data.R')"

CMD mv /home/research_project/project_output/* /home/research_project/shared_folder/
```

This Dockerfile starts off from the same base image, an image that ships R version 4.3.0, then it installs the `{dplyr}` package, a popular R package for data manipulation and it creates three directories:

- `/home/research_project`
- `/home/research_project/project_output`
- `/home/research_project/shared_folder`.

Then, it copies the `analyse_data.R` script, which contains the actual analysis made for the purposes of the research project, into the Docker image. The second-to-last line runs the script, and the last line moves the outputs generated from running the `analyse_data.R` script to a folder called `shared_folder`. It is important to say that `RUN` statements will be executed as the image gets built, and `CMD` statements will be executed as a container runs. Using this Dockerfile, an image called `research_project` can be built with the following command:

```
docker build -t research_project .
```

This image can then be archived and shared for replication purposes. Future researchers can then run a container from that image using a command such as:

```
docker run -d -it --rm --name research_project_container \
  -v /host/machine/shared_folder:/home/research_project/shared_folder:rw \
  research_project
```

The container, called `research_project_container` will execute the `CMD` statement from the Dockerfile, in other words, move the outputs to the `shared_folder`. This folder is like a tunnel between the machine that runs the container and the container itself: by doing this, the outputs generated within the container can now be accessed from the host's computer.

The image built by the process above is immutable: so as long as users can run it, the outputs produced will be exactly the same as when the original author ran the original analysis. However, if the image gets lost, and needs to be rebuilt, the above Dockerfile will not generate the same image. This is because the Dockerfile, as it is written above, will download the version of `{dplyr}` that is current at the time it gets built. So if a user instead builds the image in 5 years, the version of `{dplyr}` that will get downloaded will not be the same as the one that was actually used for the original analysis. The version of R, however, will forever remain at version 4.3.0.

So to ensure that future researchers will download the right versions of packages that were originally used for the project, the original researcher also needs to provide a list of packages that were used as well as the packages' versions. This can be quite tedious if done by hand, but thankfully, there are ways to generate such lists very easily. The `{renv}` package for the R programming language provides such a function. Once the project is done, one simply needs to call:

```r
renv::init()
renv::hydrate()
renv::snapshot()
```

to generate a file called `renv.lock`. This file contains the R version that was used to generate it, the list of packages that were used for the project, as well as their versions and links to download them. This file can be used to easily install all the required packages in the future by simply running:

```
renv::restore()
```

A researcher can thus add the following steps in the Dockerfile to download the right packages when building the image:

```
COPY renv.lock /home/research_project/renv.lock

RUN R -e "setwd('/home/research_project);renv::init();renv::restore()"
```

However, what should researchers that want to replicate a past study do if the original researcher did not provide a Dockerfile nor an `renv.lock` file? This is exactly the challenge that we were facing when trying to replicate the results of Davenport, Franco, and Iyengar (2022). We needed to find a way to first, install the right version of R, then the right version of the packages that they used, run their original script, and then repeat this procedure but this time on a recent version of R.

In order to achieve this, we used a Docker image provided by the R Installation Manager[4] project. This Docker image includes a tool, called `rig`, that makes it easy to switch R versions, so we used it to first define an image that would use R version 3.5.0 by default as a base. Here are the three commands from the Dockerfile to achieve this:

```
FROM rhub/rig:latest

RUN rig install 3.5.0

RUN rig default 3.5.0
```

Then, we had to install the packages that the original authors used to perform their analysis. We had to make some assumptions: since we only had the list of used packages, but not their exact versions, we assumed that the required packages were installed one year before the paper was published, so sometime in May 2019. With

[4]https://github.com/r-lib/rig

this assumption, we then used the Posit Package Manager[5], which provides snapshots of CRAN that can be used to install R packages as they were on a given date. We thus configured R to download packages from the snapshot taken on May 16th, 2019. Then, the original replication script gets executed at image build time and we can obtain the outputs from running a container from this image definition. With this setup, it was very simple to only switch R versions and re-executed everything. We simply had to switch the commands from the Dockerfile:

```
FROM rhub/rig:latest

RUN rig install 4.2.0

RUN rig default 4.2.0
```

Everything else: package versions and operating system that the replication script runs on, stayed the same.

With this setup, we were thus able to run the original analysis on an environment that was as close as possible to the original environment used by Davenport, Franco, and Iyengar (2022). However, it is impossible to regenerate the exact environment now. As stated, we had to make an assumption on the date the packages were downloaded, but the packages use by the original authors might have been much older. Another likely difference is that the operating system used inside Docker is the Ubuntu Linux distribution. While Ubuntu is a popular Linux distribution, it is much more likely that the original authors used either Windows or macOS to perform their analysis. In most cases, the operating system does not have an impact on results, but there have been replication studies that failed because of this, as in Bhandari Neupane et al. (2019). Thankfully, mismatch of the operating system does not seem to be an issue here.

3.3 Reproducibility isn't just version and package management — using {targets} to improve readibility and reproducibility

Reproducibility should go beyond providing the right versions of the analysis software used. Another import aspect is *readability*, which is difficult to quantify. There are however some approaches that we suggest researchers

[5]https://packagemanager.posit.co/client/#/repos/2/overview

could employ to improve the readability of their code and thus increase the probability of a successful replication.

Most research papers and studies' computer code is written as one, or several scripts that must be run in a certain order. These script-based workflows usually grow very large and become chaotic. Documentation must be written alongside the scripts to explain how they work and in which order they're supposed to be executed, and this documentation then has to be maintained and updated as the scripts evolve. When some parts of the code gets changed in one script, the researcher has to remember which parts of the other scripts get impacted and either only run the relevant, now outdated, parts, or re-run the whole project which can be quite time-consuming. Here again, it helps to view a researcher paper (and in particular its computer code) as a piece of software. Software engineers are faced with the exact same problem but solved it many decades ago using so-called build automation tools. These build automation tools allow one to describe how a particular piece of software should get build. Other software engineers that which to build that piece of software thus only need to execute the build automation tool which will take care of executing the right steps in the right order.

The solution we propose is for researchers to use build automation tools. For the R programming language, a very popular build automation tool is provided through the `{targets}` package. Using `{targets}` has many benefits:

- `{targets}` keeps track of all the inter-dependencies between the different pieces of the entire codebase. If some part gets changed, `{targets}` *knows* which other parts are affected and only re-executes these;
- another consequence of this is that `{targets}` also knows which parts of the codebase are independent from each other and can thus be safely executed in parallel. This can lead to tremendous execution speed gains;
- `{targets}` works by having the user define the computations as a pipeline. This pipeline is in essence the composition of many pure functions, which increases the readibility of the project.

As an illustration of this, we rewrote the original study as a `{targets}` pipeline.

4 Conclusion

References

Aschwanden, Christie. 2015. "Psychology Is Starting To Deal With Its Replication Problem." *FiveThirtyEight*.

Bhandari Neupane, Jayanti, Ram P. Neupane, Yuheng Luo, Wesley Y. Yoshida, Rui Sun, and Philip G. Williams. 2019. "Characterization of Leptazolines a–d, Polar Oxazolines from the Cyanobacterium Leptolyngbya Sp., Reveals a Glitch with the 'Willoughby–Hoye' Scripts for Calculating NMR Chemical Shifts." *Organic Letters* 21 (20): 8449–53.

Broockman, David, Joshua Kalla, and Peter Aronow. 2015. "Irregularities in LaCour (2014)." *UC Berkeley. Http://Stanford. Edu/~ Dbroock/Broockman_kalla_aronow_lg_irregularities. Pdf*.

Camerer, Colin F, Anna Dreber, Eskil Forsell, Teck-Hua Ho, Jürgen Huber, Magnus Johannesson, Michael Kirchler, et al. 2016. "Evaluating Replicability of Laboratory Experiments in Economics." *Science* 351 (6280): 1433–36.

Chang, Andrew C., and Phillip Li. 2022. "Is Economics Research Replicable? Sixty Published Papers from Thirteen Journals Say 'Often Not'." *Critical Finance Review* 11 (1): 185–206. https://doi.org/10.1561/104.00000053.

Clemens, Michael A. 2017. "The Meaning of Failed Replications: A Review and Proposal." *Journal of Economic Surveys* 31 (1): 326–42.

Collaboration, Open Science. 2015. "Estimating the Reproducibility of Psychological Science." *Science* 349 (6251): aac4716.

Davenport, Lauren, Annie Franco, and Shanto Iyengar. 2022. "Multiracial identity and political preferences." *The Journal of Politics* 84 (1): 620–24.

Derksen, Maarten, and Jill Morawski. 2022. "Kinds of Replication: Examining the Meanings of 'Conceptual Replication' and 'Direct Replication'." *Perspectives on Psychological Science* 17 (5): 1490–1505.

Gelman, Andrew. 2018. "Essay: The Experiments Are Fascinating. But Nobody Can Repeat Them." *The New York Times*, November.

Herndon, Thomas, Michael Ash, and Robert Pollin. 2014. "Does High Public Debt Consistently Stifle Economic Growth? A Critique of Reinhart and Rogoff." *Cambridge Journal of Economics* 38 (2): 257–79.

Lupia, Arthur, and Colin Elman. 2014. "Openness in political science: Data access and research transparency: Introduction." *PS: Political Science & Politics* 47 (1): 19–42.

Nosek, Brian A, and Timothy M Errington. 2020. "What Is Replication?" *PLoS Biology* 18 (3): e3000691.

Philip, Glandon. 2010. "Report on the American Economic Review Data Availability Compliance Project." *Unpublished Manuscript*.

Simonsohn, Uri. 2021. "[95] Groundhog: Addressing The Threat That R Poses To Reproducible Research." *Data Colada*. http://datacolada.org/95.

Stockemer, Daniel, Sebastian Koehler, and Tobias Lentz. 2018. "Data access, transparency, and replication: New insights from the political behavior literature." *PS: Political Science & Politics* 51 (4): 799–803.