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

Taylor J. Wright* Bruno Rodrigues †

February 4, 2025

The growing concern regarding reproducibility and replicability of social science results has powered the adoption of open data and code requirements at journals and norms among researchers. However, even when these norms and requirements are followed, changes to the software used in data cleaning and analysis can render papers non-reproducible. This paper details the challenges of reproducibility in the face of software updates. We present a case study of a published article whose results are no longer reproducible due to changes in the software used. We then discuss the tools and techniques researchers can use to ensure that their research remains reproducible despite changes in the software used.

*Brock University, twright3@brocku.ca
†Ministry of Research and Higher education, 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 led 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 Open Science Collaboration 2015). While the *replicability* of studies has received 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, 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, using the original data, obtaining the results presented in the study.[1]

In our view, reproducibility is an insufficient but necessary condition for replication—that is, work cannot be replicable if it is not reproducible and it likely does not make sense to spend resources on replication if research is not in the first place 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 and publication 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. Rainey and Roe (2024) collect data availability policies between January and April of 2023 for "Political Science" and "International Relations" journals in Social Science Citation Index and

[1] For further discussion on definitions, scopes, and types of replication and reproducibility see, for example, Clemens (2017), Derksen and Morawski (2022), Hamermesh (2007), and 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

APSA journals, finding that 20% require data to be shared and 65% encourage (but do not require) it. Some journals, such as the American Economic Review(AER) or American Journal of Political Science (AJPS), even have verification policies that require authors to upload their replication materials and have their results verified by another team (either the journal's replication team (AER) or a third party (AJPS)) before 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 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 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 numbers starting in version 3.6.0.

We contribute to the existing literature in three ways. First, we add to the discussion about the importance of the availability of replication materials. Existing research documents that the availability of replication materials is associated with a higher citation count (Piwowar, Day, and Fridsma 2007; Piwowar and Vision 2013) and with a higher likelihood of replication (Philip 2010). Much of this research focuses on the provision of data and code as a necessary condition for facilitating replication and rebuilding trust in science (Dafoe 2014; Lupia and Alter 2014). We contribute to this literature by emphasizing that, while we agree that data and code availability are necessary, their existence cannot guarantee reproducibility. This brings us to our second contribution: we expand on the discussion of problems raised by software and package versioning for reproducibility by providing a walkthrough of the problem using a concrete example. Lastly, we further the discussion of tools and best practices for ensuring reproducibility. Specifically, we provide a step-by-step guide to using Docker and the `renv` `R` package to ensure the reproducibility of research.

The rest of the article proceeds as follows: Section 2 walks through the reproducibility issues

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

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.

# 2 Reproduction

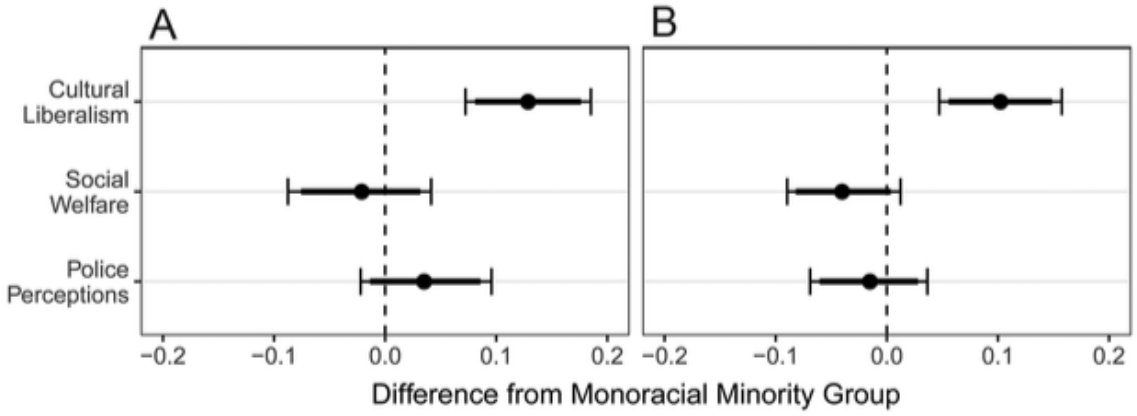
## 2.1 Illustrating the issue with software versioning

Davenport, Franco, and Iyengar (2022) examine the political ideology and policy preferences of multiracial Americans (those who identify with more than one racial group). This section of Americans is growing rapidly, but little is known about their politics. The authors survey White-Asians and White-Blacks from YouGov's Internet panel and compare them to similarly surveyed White, Black, and Asian monoracial respondents, looking for differences in political ideology, party affiliation, positions about cultural issues, social welfare, and police treatment across racial groups.

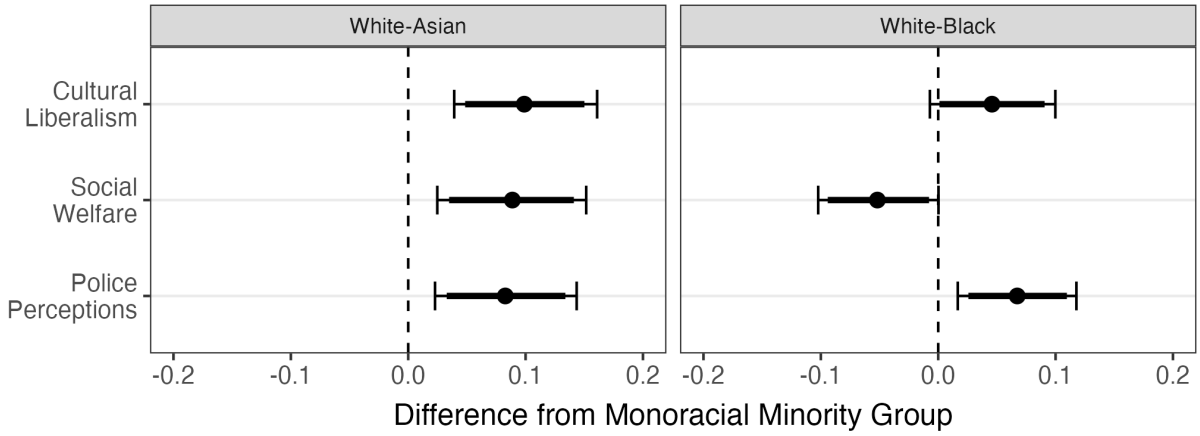
Figure 1a and presents Figure 1 from Davenport, Franco, and Iyengar (2022) which contains the main results. Panel A corresponds to results for White-Asian respondents while panel B corresponds to White-Black respondents. The x-axis reflects the differences in a given multiracial group's attitudes relative to the corresponding monoracial group (e.g. White-Asian relative to Asians) with higher values meaning more liberal attitudes. This figure indicates that White-Asians have more liberal attitudes on cultural issues than Asians while White-Blacks have more liberal attitudes on cultural issues than Blacks but views regarding social welfare and police perception appear to be comparable between White-Asians and Asians and between White-Blacks and Blacks.

Importantly, the authors provide replication materials at the Journal of Politics Dataverse, including the code used to conduct their analysis as well as the data itself. The code files contain comments noting that the analysis "Requires `R` version 3.6.2 (2019-12-12)". This allows us to recreate the figures.

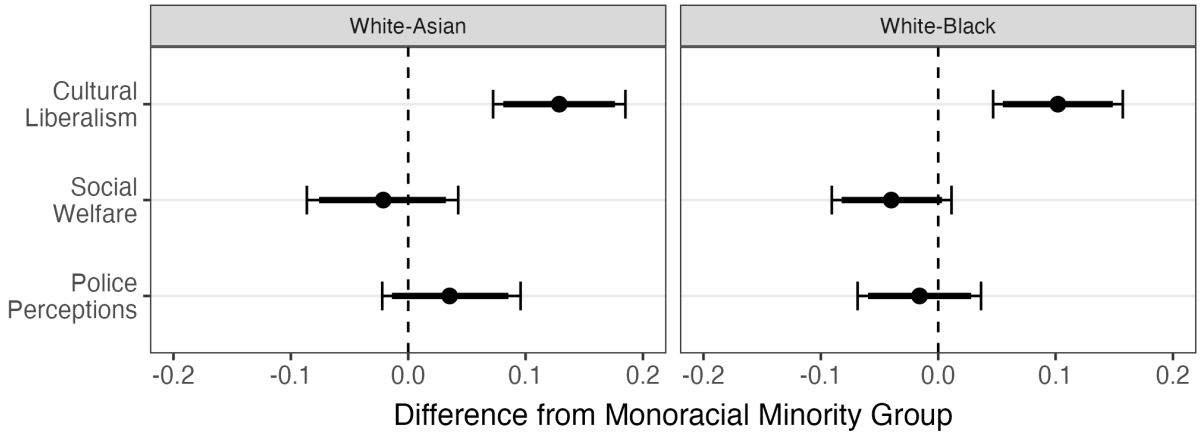
Figure 2a presents Figure 1 from Davenport, Franco, and Iyengar (2022) when using a later software version (specifically a version of `R` post 3.6.0 such as the version 3.6.2 that the authors note in their code files). In Panel A, White-Asians are now more liberal than Asians and Whites on all issues. In Panel B White-Blacks have more liberal views on cultural issues than Blacks (in terms of point estimates though the confidence interval now contains 0) and are now more liberal regarding police perceptions.




(a) Figure 1 from Davenport, Franco, and Iyengar (2022). Multiracial attitudes relative to monoracial minorities. A, White-Asian; B, White-Black. Higher values are more liberal attitudes.




(a) Figure 1 from Davenport, Franco, and Iyengar (2022) when using a later software version (R post 3.6.0). Multiracial attitudes relative to monoracial minorities. A, White-Asian; B, White-Black. Higher values are more liberal attitudes.




(a) Figure 1 from Davenport, Franco, and Iyengar (2022) when using an earlier software version (R pre 3.6.0). Multiracial attitudes relative to monoracial minorities. A, White-Asian; B, White-Black. Higher values are more liberal attitudes.

Figure 3a provides Figure 1 Davenport, Franco, and Iyengar (2022) when using an earlier software version (specifically a version of `R` pre 3.6.0). These are otherwise the same as those presented in the published article.

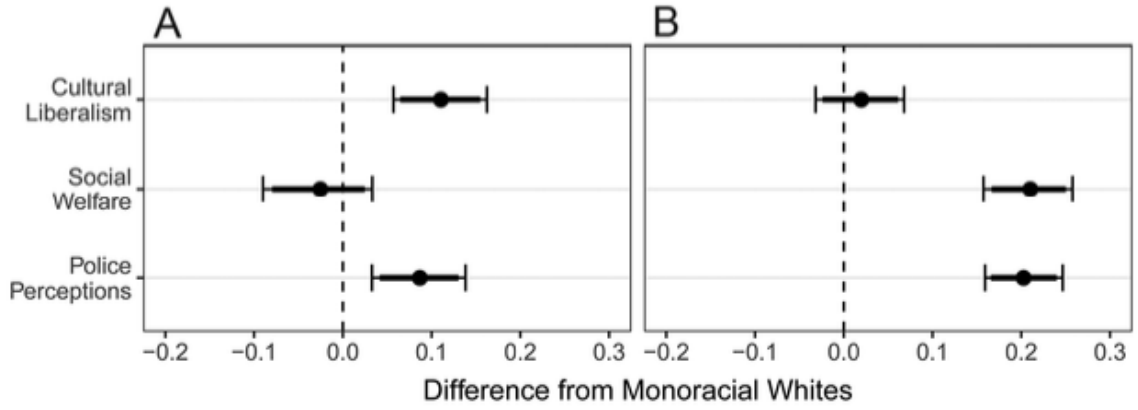
Figure 4a presents Figure 2 from Davenport, Franco, and Iyengar (2022). Again, Panel A corresponds to results for White-Asian respondents while panel B corresponds to White-Black respondents. The x-axis reflects a similar difference but relative to monoracial Whites (e.g. White-Asian relative to Whites). Figure 2 Panel A suggests that White-Asians are more liberal than Whites when it comes to cultural issues and police perceptions but similarly liberal with views on social welfare. Figure 2 Panel B indicates that White-Blacks are similarly liberal to Whites on cultural issues but are more liberal on issues of social welfare and police perceptions.

Figure 5a presents Figure 2 from Davenport, Franco, and Iyengar (2022) when using a later software version (specifically a version of `R` post 3.6.0 such as the version 3.6.2 that the authors note in their code files). In Panel A, White-Asians are now more liberal than Asians and Whites on all issues. In Panel B of Figure 5a the point estimates appear very similar though there is an issue computing confidence intervals.

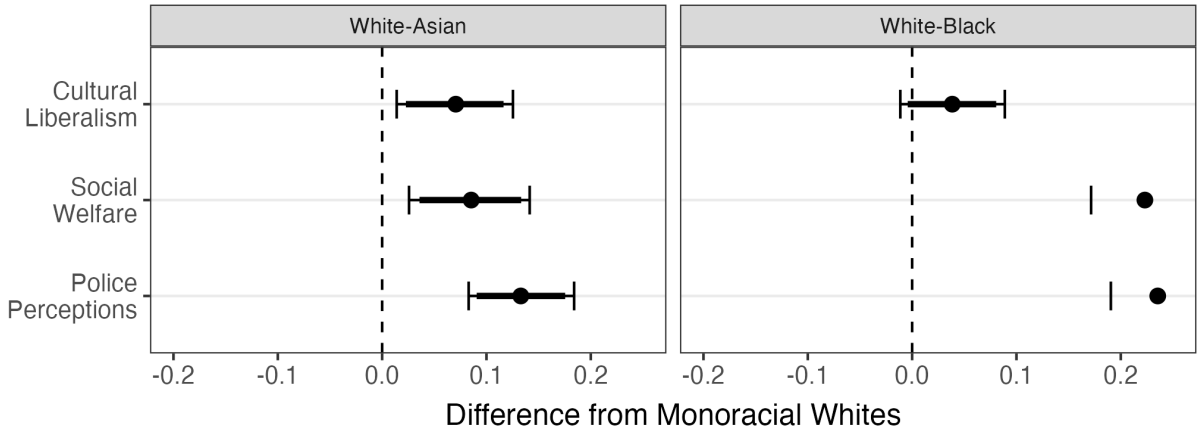
Figure 6a provides Figure 2 from Davenport, Franco, and Iyengar (2022) when using an earlier software version (specifically a version of `R` pre 3.6.0). Minor issue with the confidence interval in Panel B aside, these are also otherwise the same as those presented in the published article.

The issue seems to be that the weights the authors use are non-integer and the main function used in their analysis, `Zelig()`, uses `sample()` in cases with non-integer values (see http://docs.zeligproject.org/articles/weights.html). Following changes to base `R` in 3.6.x the sample() function, and thus now `Zelig()`, yields different results in `R` versions later than 3.5.x. Before `R` 3.6.0 `RNGkind(sample.kind = "Rounding")` was the default behavior but after 3.6.0 the sample function's new default behavior is `RNGkind(sample.kind = "Rejection")`.[4] Why was this change made? According to the `RNGkind` documentation, "sample.kind" can be "Rounding" or "Rejection", or partial matches to these. The former was the default in versions prior to 3.6.0: it made

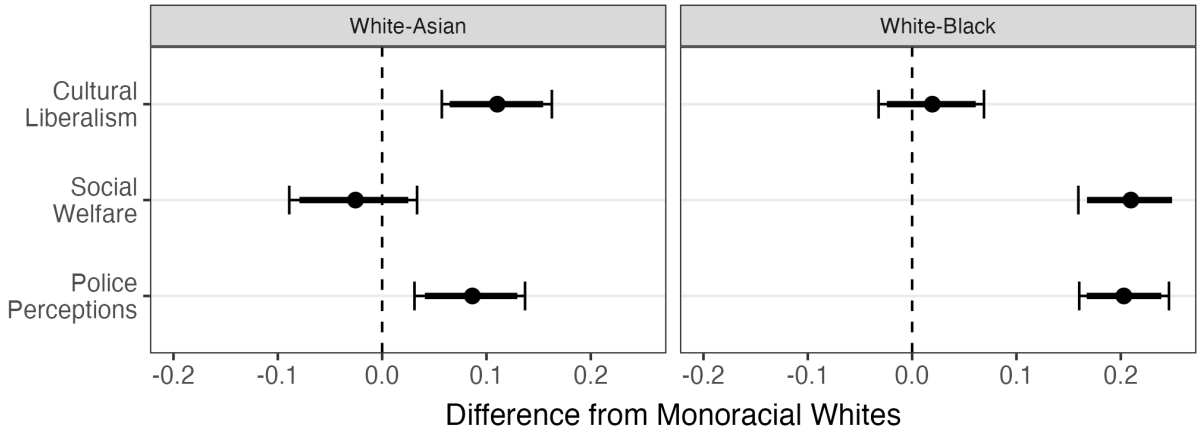
[4]See this announcement about `R` 3.6.0 for more details: https://blog.revolutionanalytics.com/2019/05/whats-new-in-r-360.html




(a) Figure 2 from Davenport, Franco, and Iyengar (2022). Multiracial attitudes relative to monoracial Whites. A, White-Asian; B, White-Black. Higher values are more liberal attitudes.




(a) Figure 2 from Davenport, Franco, and Iyengar (2022) when using a later software version (R post 3.6.0). Multiracial attitudes relative to monoracial Whites. A, White-Asian; B, White-Black. Higher values are more liberal attitudes.




(a) Figure 2 from Davenport, Franco, and Iyengar (2022) when using an earlier software version (R pre 3.6.0). Multiracial attitudes relative to monoracial Whites. A, White-Asian; B, White-Black. Higher values are more liberal attitudes.

sample noticeably non-uniform on large populations, and should only be used for reproduction of old results" (R Core Team 2024). In essence, the algorithm for number generation contained a bug that led to bias in the probabilities of numbers being chosen.[5]. As such, the correct version is the one created using R after version 3.6.0.

A key takeaway, and focus of this paper, is that despite the authors providing replication materials and indicating which version of `R` they believed they used, the results cannot be reproduced. This aspect is not a criticism of the authors, who were unknowingly using software that contained an error, but rather a demonstration of the challenges of reproducibility in the face of software updates.[6] There are ways to ensure that the estimates are consistent (such as ensuring to always explicitly articulate options in functions so that changes to defaults cannot break existing code. However, reproducibility is still threatened by software updates that result in larger changes (such as what inputs are required). Manually documenting software versions (including packages or commands) in the code or README files is a good start, but as our example shows, it is not enough to ensure faithful reproduction. It should also be noted that the package that provides the `Zelig()` function, `Zelig` is not available anymore. It is still possible to install it from the CRAN archives, but this constitutes an additional risk that must be managed for reproducibility.

In general, risks for reproducibility include: package or software updates that render code backward incompatible (for example, an update that changes the order of arguments supplied to a function); the use of packages or software that are not or no longer are being actively developed; similarly, use of packages or software with only a single maintainer (especially if the language used is dated or obscure); and the use of packages or software with many dependencies, as only one of these dependencies needs to fail in order to render code non-functioning. Each of these represent possible failure points for code and we encourage researchers to consider these when deciding which software or packages to use in their projects.

[5]We refer interested readers to the thread reporting the bug and links contained within: https://bugs.r-project.org/show_bug.cgi?id=17494

[6]It is important to note that these kinds of breaking changes R are relatively infrequent and breaking changes are more likely to come from updated packages.

## 2.2 Result sensitivity to seed

Given that the results obtained by Davenport, Franco, and Iyengar (2022) changed when the way that numbers are generated in `R` changed in version 3.6.0, we sought to check whether the results are robust to different seeds or if this change was simply bad luck.

Figure 7 plots the distribution of the means obtained from running the code used to create Figure 1a 100 times with 100 different random seeds on R version 3.5. Each panel in this figure shows the results for the models comparing multiracial attitudes to those of monoracial minorities along a different measure of attitudes. The solid black line plots the distribution of point estimates, the solid red vertical lines are the point estimates from Figure 1a, while the dashed red vertical lines are the 95% confidence intervals. These results suggest the estimates obtained by Davenport, Franco, and Iyengar (2022) are very similar to those that would have been generated using any other seed. The estimate for the social welfare comparison between white-Asians to Asians is perhaps somewhat more negative (and thus less liberal) than was found using the original seed. This makes sense, given that our results in Figure 5a shift towards being more liberal for this comparison. To us, this suggests that the authors' original conclusions for these comparisons are reasonably not dramatically impacted by the specific seed used.

Figure 8 similarly plots the distribution of estimates obtained from 100 runs with 100 different random seeds on R version 3.5 for the models comparing multiracial attitudes to those of monoracial whites. As above, the solid black line plots the distribution of point estimates, the solid red vertical lines are the point estimates from Figure 1a while the dashed red vertical lines are the 95% confidence intervals. Each panel corresponds to a different comparison between racial groups and a different measure of attitudes. Similarly to Figure 7, the original estimates for White-Asian social welfare attitudes are less liberal than those suggested by other seeds, however the other estimates are in line with those from our many seeds exercise.

In summary, the results obtained by Davenport, Franco, and Iyengar (2022) are not able to be recreated despite providing code and a version number for `R`. This is due to a bug fix in `R` that

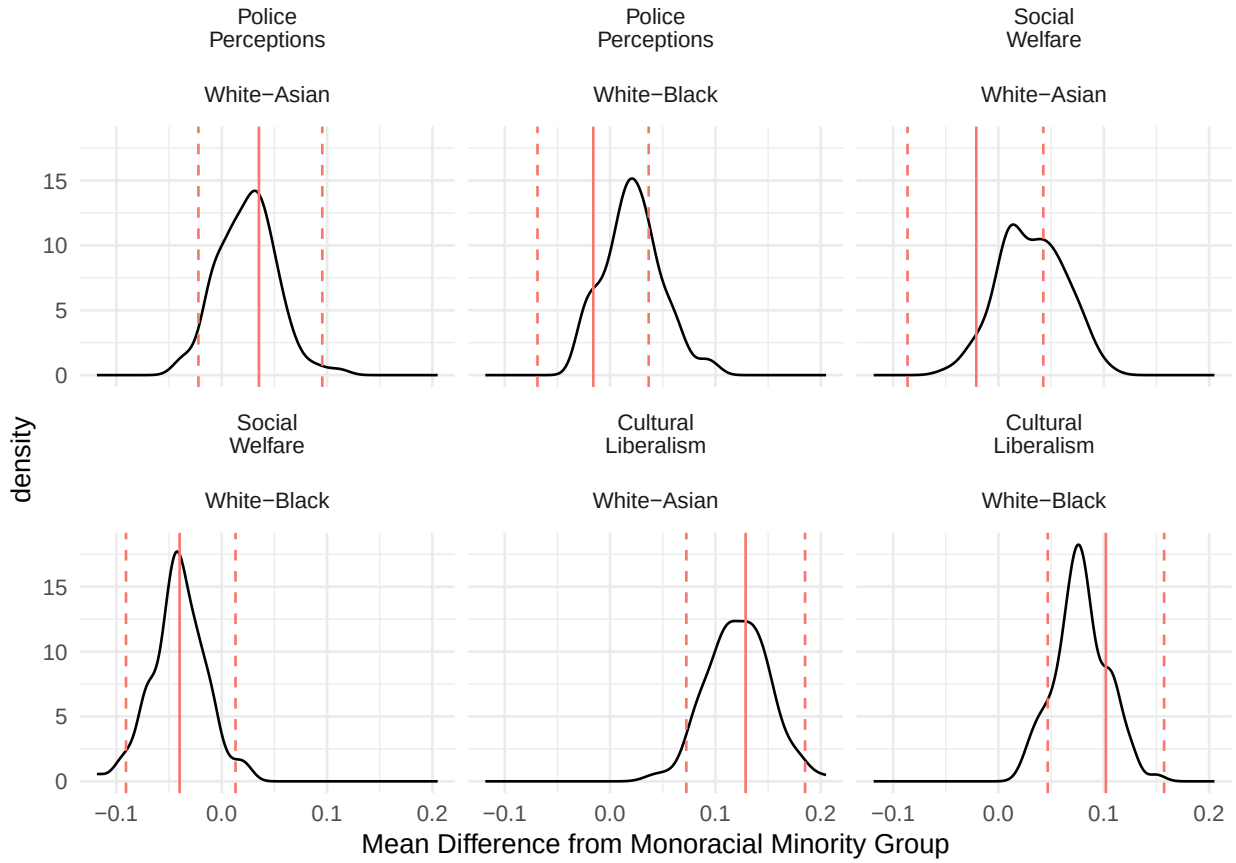



Figure 7: Solid black line splots distribution of estimates of multiracials' attitudes, relative to monoracial minority background. 100 repetitions with 100 random seeds. Solid red line represents points estimates and dashed red line reflects 95% confidence intervals from Davenport, Franco, and Iyengar (2022).

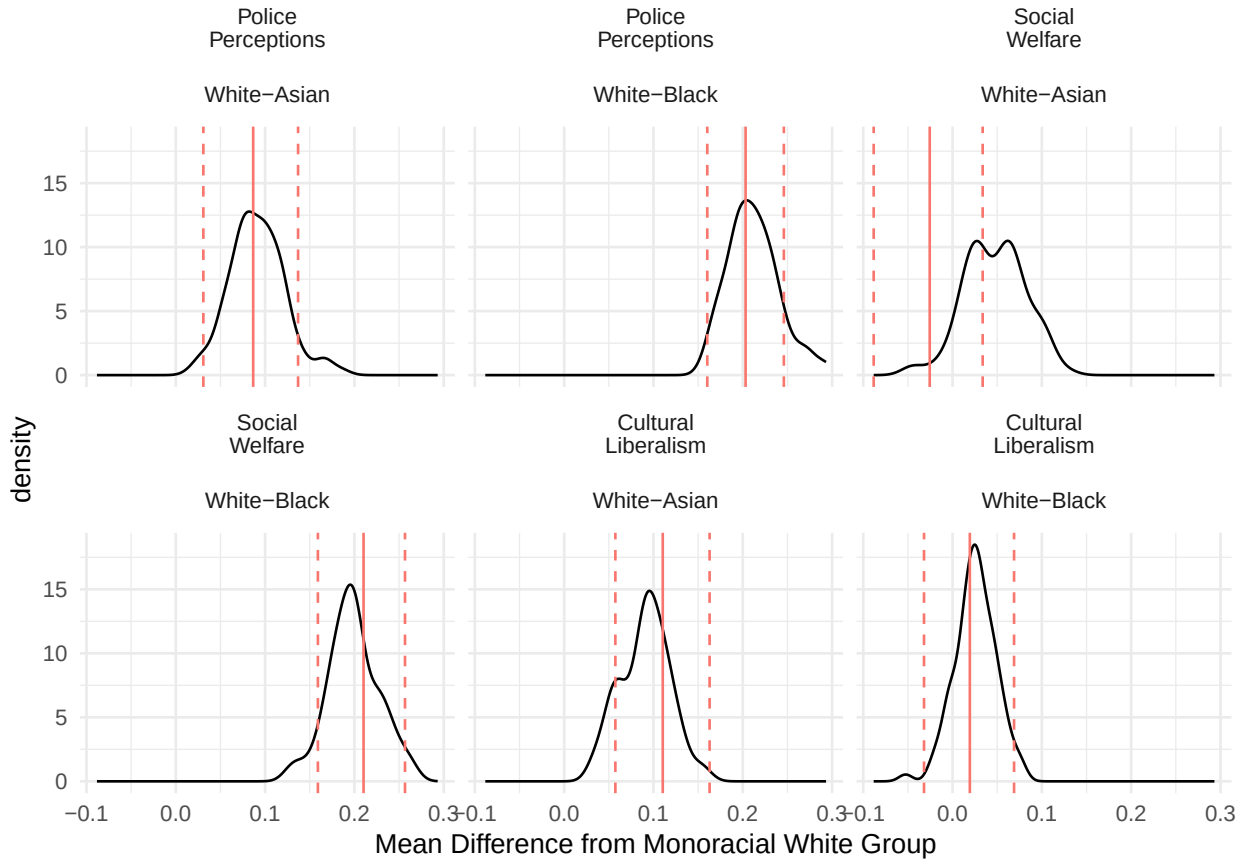



Figure 8: Solid black line splots distribution of estimates of multiracials' attitudes, relative to monoracial white background. 100 repetitions with 100 random seeds. Solid red line represents points estimates and dashed red line reflects 95% confidence intervals from Davenport, Franco, and Iyengar (2022).

changed default behavior when generating numbers. As the authors use code that generates numbers in this way, they could be affected by the possible bias arising from the bug which was present in the version of `R` they used. In order to examine this, we ran their code 100 times using 100 random seeds and plotted the results against the original estimates. We conclude that the original results appear reasonable and are not driven by the bug that was fixed in `R`.

# 3 Discussion

## 3.1 Building a 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 the author' original scripts.

To reproduce 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 that enables one to build so-called *images*. The images are usually built on top of a Linux operating system and contain all of the required software to build any type of output. The images are made up of the software, dependencies, code, etc., that will be used to generate the analysis. Think of the image like a big sandbox that gets frozen in time and doesn't change but has all of the shovels and buckets that we need ready to go. After the image containing the output has been built, users in turn only need to be able to run Docker *containers* instantiated from the image definition to build the output. Containers are basically one instance of an image, every time we step foot into that sandbox do create something, we start from the same position where it was frozen in time. If we want to change that start position, we need to rebuild the sandbox slightly differently. 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 a version mismatch of the operating system.

The same approach can be used to build research projects, which also suffer 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 reproduce the results can now simply 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*, which lists all the necessary steps to build an output. 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 (Boettiger and Eddelbuettel 2017) 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 be run when the user runs a container defined from our image, so in this case, simply the R interactive prompt. Concretely, the above image would allow someone to run R version 4.3.0 as a container, completely isolated from the rest of their computer. This allows users to run different versions of R on the same system. We can add more commands to run a complete research project, from start to finish. Below is an example of a Dockerfile that runs an analysis script, by first stating which base image is going to be used, and then adding packages, files and data to run the entire project:

```
FROM rocker/r-ver:4.3.0

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

RUN mkdir /home/research_project
```

```
RUN mkdir /home/research_project/data

RUN mkdir /home/research_project/project_output

RUN mkdir /home/research_project/shared_folder

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

COPY dataset.csv /home/research_project/dataset.csv

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 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 (`mkdir` meaning "make directory"):

- `/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 research project, and the `dataset.csv` file, 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 entered into a terminal/command line:

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

The "-t" is an option that tags this image with the name `research_project` so that we can refer to it later. This name could be whatever you would like. The "." means " this current directory/folder", so the command will build the image from the Dockerfile contained in the current folder. Again, during the building of an image, everything up to the `CMD` line in the Dockerfile will be run.

This image can then be archived and shared for replication purposes. Other images can be built on top of this one, and will thus contain R version 4.3.0, the `{dplyr}` package, the data and the project's folder structure. Future researchers can also run a container from that image using a terminal/command line 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
```

This is a long command and we have broken it into multiple lines using " " as a line break. The " docker run" is the command that starts the new container based on the image previously generated. We tell " docker run" which image we want to use by specifying the tag at the end, here "research_project", which is what we tagged the image as previously. Using the "-d" option starts the container running in the background and "–rm" will remove the container when it is finished running. The "-v" option lets us specify the path to the shared folder on our computer where we want to output to be placed.

The container, called `research_project_container`, using the "–name" option, will execute the `CMD` statement from the Dockerfile. In other words, it will 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 "/host/machine/shared_folder" is the path to the shared folder on the host's computer while "/home/research_project/shared_folder" is the path to shared folder inside of the Docker container. The ":rw" needs to be present to allow the Docker container to read and write

in the shared folder. After this command runs, the host should be able to navigate to the path to the shared folder and access all of the output produced.

The image built by the process above is immutable: so as long as users can run it, the outputs produced will be 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 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:

```
renv::init()
```

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()
```

This will download the right packages at image build time. It would also be possible to copy the images from the host computer into the image, however, this would turn the build process non-reproducible, as the image would only successfully build on that particular host computer.

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()"
```

If we have modified the Dockerfile we previously introduced to add these lines, a researcher can build the image again with the `docker build` command that we introduced above and then run the container with the `docker run` command previously specified. Every time we change the Dockerfile to add more commands before the `CMD` command used to run the docker container, we need to rebuild the docker image.

# 4 Conclusion

Researchers have to deal with the following aspects of their projects to ensure their long-term reproducibility:

- They need to state the correct versions of R (or any other language) used for the analysis;
- They need to state the correct versions of every other used package for the analysis;
- They need to make sure that their code is readable and easy to explore; and
- They need to provide instructions on how to run the whole project.

By using Docker, it is possible to provide the correct versions of `R` and packages (using `renv`, `groundhog` or a snapshot of the CRAN repository). Another important way to improve reproducibility is also to set up research projects as the output of pipelines instead of scripts, but this is outside the scope of this paper. But this is not only useful for making projects reproducible: by using these tools, researchers also avoid common pitfalls while working on their projects. Research projects can last for years and researchers typically work on several projects in parallel; by using the tools discussed in this article, researchers can thus have project-specific `R` and package versions, thus avoiding the common issue of updating their package library because they need the latest feature of a package for one project, but because packages got updated, another project now produces different results or simply does not run anymore.

However, we realize that the entry cost to a tool like Docker is not low: so at the very least, we advise researchers interested in reproducibility to update R and R packages at the start of a new project, and then to be very conservative with updating `R` itself while working on a project and to use either `renv` or `groundhog` to have project-specific libraries. This is because if researchers at least provide the R version used and an renv.lock file (if they used `renv`, or a date, if they used `groundhog`) it is always possible to re-generate an environment with the right version of `R` and packages as we did for this paper. This solution is not the gold standard of reproducibility though, but easy enough to be implemented at a low cost. This also implies that researchers need to share their code and all other necessary information on a platform such as Github, and it implies that

research has to be performed using free and open source tools as proprietary tools cannot easily be installed by other researchers wanting to reproduce a study.

Furthermore, one must be aware that while Docker offers many benefits, it is not a panacea. Users of Docker must pay attention to several issues. First of all, while the operating system one uses Docker on does not matter, neither for building images nor for running containers, the CPU architecture does matter. This means that an image built on (and thus compatible with) the 64-bit CPU architecture that is used in Intel and AMD processors (henceforth amd64) cannot be used on the arm64 architecture. Arm64 CPUs are the ones driving mobile phones and modern Apple computers (under the commercial name "Apple Silicon"), in other words, this means that if a researcher built an image that is not compatible with arm64 CPUs, reproducers on modern Apple computers might have issues running containers from this image. Arm64 is becoming also more popular for devices made by other manufacturers, and we believe that it will become more common in the near future. That being said, more and more base images that can be used to start off a researcher's own analysis both ship an amd64 and arm64 versions, which means that at the very least, reproducers will be able to rebuild the image. However, it is still not guaranteed that the image will successfully rebuild, as the software included must then also be compatible with arm64. Secondly, in the ideal scenario, images would be stored and made available to reproducers to avoid imposing on reproducers the build process. By using best practices images should be rebuildable, however, one must keep in mind that the base operating system that is used to start off an image for a project also has a finite lifespan. Most base images use the Ubuntu operating system in its *long term support* (LTS) variant. These LTS versions receive support for 5 years: this means that trying to rebuild an image after the support runs out will not be possible, and it will become mandatory to upgrade the base image and rerun the build process. This is actually a difficulty that we encountered when trying to reproduce the results of this study, which is why we had to use an image that made it easy to install old versions of R on new versions of operating systems (as described in the appendix). Thankfully, there are several ways to store images, either for free on a so-called image registry such as Docker Hub[7] or to GitHub Packages, a registry offered by GitHub. It is also possible to store the images on a standard FTP or web server on an institutional website and offer them as a download.

[7] https://hub.docker.com/

To conclude, over-reliance on Docker can be an issue. While we have several alternatives for handling packages, Docker was the only solution that we suggested to essentially "freeze" versions of R and other lower-level system dependencies, and users must be aware of its shortcomings. We want to finish this conclusion by stating that other tools achieve the same purpose, notably Nix (Dolstra 2006). Nix is not a containerization tool like Docker, but a package manager that allows users to install exact versions of software, including scientific software, reproducibly. It is possible to essentially use Nix to replace Docker and `renv` (or `groundhog`), but presenting this tool is outside the scope of this article.

# 5 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.

Boettiger, Carl, and Dirk Eddelbuettel. 2017. "An Introduction to Rocker: Docker Containers for r." *arXiv Preprint arXiv:1710.03675*.

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.

Dafoe, Allan. 2014. "Science Deserves Better: The Imperative to Share Complete Replication Files." *PS: Political Science & Politics* 47 (1): 60–66.

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.

Dolstra, Eelco. 2006. *The Purely Functional Software Deployment Model*. Utrecht University.

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

Hamermesh, Daniel S. 2007. "Replication in Economics." *Canadian Journal of Economics/Revue Canadienne d'économique* 40 (3): 715–33.

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 George Alter. 2014. "Data Access and Research Transparency in the Quantitative Tradition." *PS: Political Science & Politics* 47 (1): 54–59.

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*.

Piwowar, Heather A, Roger S Day, and Douglas B Fridsma. 2007. "Sharing Detailed Research Data Is Associated with Increased Citation Rate." *PloS One* 2 (3): e308.

Piwowar, Heather A, and Todd J Vision. 2013. "Data Reuse and the Open Data Citation Advantage." *PeerJ* 1: e175.

R Core Team. 2024. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Rainey, Carlisle, and Harley Roe. 2024. "The Data Availability Policies of Political Science Journals." OSF.

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

———. 2023. *Groundhog: Make Your r Scripts Reproducible by Replacing Library(pkg) with Groundhog.library(pkg,date)*.

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.

# A Appendix

## A.1 Rebuilding the oinal development environment using Docker

Section Section 3 outlined what steps a researcher starting a new project should take to set up a reproducible environment using Docker and `renv`. However, what should researchers who 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[8] 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 this assumption, we then used the Posit Package Manager[9], 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,

[8] https://github.com/r-lib/rig
[9] https://packagemanager.posit.co/client/#/repos/2/overview

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-execute 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 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 used 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, the mismatch of the operating system does not seem to be an issue here.

It should also be noted that strictly speaking, using Docker is not completely reproducible either: this is because the underlying base image, upon which we build our complete analysis, also changes through time. In the examples above the image we used, `rhub/rig:latest` is based on Ubuntu 22.04, which is, as of writing, the latest long-term support release of the Ubuntu operating system. This version of Ubuntu thus still gets updated: so when building the image for our project, the underlying base image will be different today than 6 months from now. And

when the next Ubuntu long-term support will be released, in April 2024, then `rhub/rig:latest` will this time be based upon this new release of Ubuntu. So building in May 2024 will yield a different image than now. To avoid this, it is possible to use a so-called *digest* which ensures the same image will forever be used as a base. For this paper, we used the following image: `rhub/rig@sha256:0667321839be0c2ade5c14fd8c8bbb311a843ddf690081c5d0dd80dcc98bd6c6`.

Finally, to ensure completeness, instead of using the Posit Package Manager with a fixed date, we could have used the `{groundhog}` package by Simonsohn (2023). This package makes it possible to download packages as of a certain date as well, and it does not require users to change any of `R`'s default configurations.