

Supplementary Materials

Successes and struggles with computational reproducibility: Lessons from the Fragile Families Challenge*

David M. Liu and Matthew J. Salganik

1 Replication status for each paper

- **Ahearn and Brand:** We were able to reproduce the three prediction.csv files within our error tolerance, and we were able to reproduce Table 2. We did not attempt to reproduce the other tables and figures because we did not deem them to be essential to reproducing the main results in the paper, and the relevant files were not included in the replication materials. One thing to note is that while the values in the prediction.csv files match nearly exactly, the row order differed between the provided and generated prediction.csv files. As an example, rows 13 and 21 are swapped for model 1 (imputed data).
- **Altschul:** We were able to reproduce all 3 prediction.csv files exactly. Further, we were able to reproduce Tables 1, 2, and 3 (with some tiny differences likely due to rounding). We note that there is some manual post-processing required to take the output from `table_figures.txt` and create Table 1. We did not deem the figures necessary to reproduce.

*The Docker images described in this article are available at: https://hub.docker.com/r/2018dliu/fragilefamilieschallenge_socius_reproducibility.

- **Carnegie and Wu:** We ran out of time to verify the results. However, the code is available, and we have created a Docker image with the necessary dependencies.
- **Compton:** We ran out of time to verify the results. However, the code is available, and we have created a Docker image with the necessary dependencies.
- **Davidson:** We ran out of time to verify the results. However, the code is available, and we have created a Docker image with the necessary dependencies.
- **Filippova et al.:** All 25 prediction.csv files are computationally reproducible. Initially, we were able to exactly match 15 of the prediction.csv files, but not all of them. For the files that did not match the differences were minor (differing by at most 0.284) and did not alter the substantive conclusions of the paper. Working with the authors, we decided that these deviations were likely caused by differences between the authors' computing environment and our computing environment. Therefore, we decided that the final results would be those that came from our computing environment, which is readily available (as opposed to the authors' computing environment which is only available to researchers at a specific university and will likely change over time). Further, we were able to reproduce Figures 2, 3, 4, and 5 (Figure 1 did not need to be reproduced).
- **Goode et al.:** We were able to reproduce the one prediction.csv file. There are 3 tables, but they were created manually and we didn't attempt to reproduce them.
- **McKay:** We were able to reproduce the one prediction.csv file within our error tolerance. Further, we were able to reproduce Table 1 and Figures 1 and 2. We did not attempt to reproduce Table 2 because we did not deem it to be essential to reproducing the main results in the paper, and the relevant files were not included in the replication materials.
- **Raes:** We were able to reproduce all 11 prediction.csv files within our error tolerance. Further, we were able to reproduce the tables. For Table 1, we were able to reproduce all rows related to submitted predictions. For Table 2, we did not attempt to reproduce the results

because the table includes leaderboard and holdout scores. For Table 3, we only attempted to reproduce the results related to model 6, and we were successful.

- **Rigobon et al.:** We ran out of time to verify the results. However, the code is available, and we have created a Docker image with the necessary dependencies.
- **Roberts:** We ran out of time verifying the results from phase 1, but given those results, we were able to match, within an error tolerance of 10^{-15} , the 150 MSE scores calculated in phase 2. We did not deem the tables and figures necessary to reproduce as they were based upon the leaderboard and holdout scores.
- **Stanescu et al.:** We were able to reproduce the single prediction.csv file. We were able to reproduce Tables 1 and 2 and Figures 1 and 2.

2 Amazon Web Services specification

Table 1 shows the software and hardware that we used to execute the code for eleven of the twelve papers.¹ All of the machines were accessed through Amazon Web Services and used a Linux operating system known as the Amazon Linux AMI. The AMI ID specifies the exact operating system version we used. We determined the hardware configuration based on the computing demands of each submission and a trial-and-error process. As a default, we instantiated machines of type t2.2xlarge, which was the most powerful computing category in Amazon Web Services' General Purpose T2 series. For submissions that needed more computing resources, we used the c5.9xlarge instances.

¹Ahearn and Brand (2019) required an installation of Stata, which is not open source software. Therefore, we could not include Stata inside a Docker image. Ultimately, we executed the code on a private computing cluster with the following specifications: Intel Xeon CPU E7-4850 v3 @ 2.20 GHz (4 processors) with 512 GB RAM, using a 64-bit operating system, operating Windows Server 2012 R2 Standard, running Stata 14.

First Author	Instance Type	AMI ID
Altschul	t2.2xlarge	amzn-ami-hvm-2017.09.0.20170930-x86_64-gp2 (ami-8c1be5f6)
Carnegie	t2.2xlarge	amzn-ami-hvm-2017.09.0.20170930-x86_64-gp2 (ami-8c1be5f6)
Compton	t2.2xlarge	amzn-ami-hvm-2017.09.0.20170930-x86_64-gp2 (ami-8c1be5f6)
Davidson	c5.9xlarge	amzn2-ami-hvm-2.0.20181008-x86_64-gp2 (ami-0922553b7b0369273)
Filippova	c5.9xlarge	amzn2-ami-hvm-2.0.20181008-x86_64-gp2 (ami-0922553b7b0369273)
Goode	c5.9xlarge	amzn2-ami-hvm-2.0.20181024-x86_64-gp2 (ami-013be31976ca2c322)
McKay	t2.2xlarge	amzn-ami-hvm-2017.09.0.20170930-x86_64-gp2 (ami-8c1be5f6)
Raes	t2.2xlarge	amzn-ami-hvm-2017.09.0.20170930-x86_64-gp2 (ami-8c1be5f6)
Rigobon	c5.9xlarge	amzn2-ami-hvm-2.0.20181008-x86_64-gp2 (ami-0922553b7b0369273)
Roberts	t2.2xlarge	amzn-ami-hvm-2017.09.0.20170930-x86_64-gp2 (ami-8c1be5f6)
Stanescu	t2.2xlarge	amzn-ami-hvm-2017.09.0.20170930-x86_64-gp2 (ami-8c1be5f6)

Table 1: All of our Amazon Web Services instances ran a distribution of Linux known as Amazon Linux AMI. The “AMI ID” specifies the exact version. We varied each instance’s hardware specification (“Instance Type”) based on the code’s computational load.

3 Reproducibility memo

The following is a copy of the memo that we sent to all authors along with our reviews of their initial manuscripts.

Background behind reproducibility guidelines

First, we’d like to step back from the details to describe the high-level goal. We want your articles to be computationally reproducible, which means that another researcher could regenerate the results in your paper using the Challenge data, your code, and any additional data that you have created. Computational reproducibility will increase the impact of your work individually, and it will increase the contribution of the Challenge collectively.

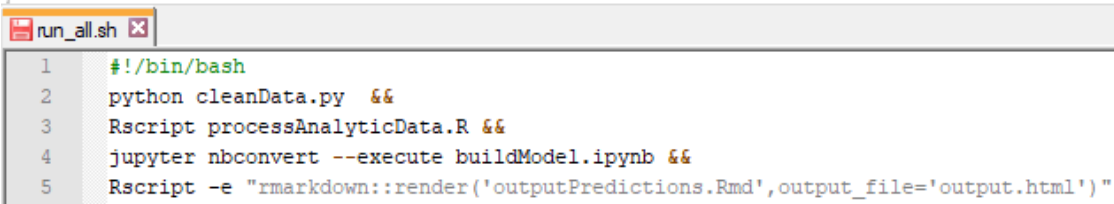
As we’ve learned during this first round of reviews, the goal of computational reproducibility is widely shared by scientists, easy to state, and tricky to achieve. Based on what we’ve learned from your code, our thinking on how to achieve this goal has evolved. In particular, we’ve been very influenced by the idea of a research pipeline described by Peng and Eckel (2014), which is nicely captured by this figure: <http://bit.ly/2qrTWXK>.

The goal of this document is to provide you with guidelines that support computational

reproducibility of your entire research pipeline, which goes from raw data to final output. You don't have to follow these guidelines exactly; if you devise a system that you think is better, you are welcome to use it. But, if you have no system in place, we are going to strongly encourage that you adopt these guidelines.

The Guidelines

The most important thing to keep in mind is that we are asking you to create one single script named `run_all` that executes all the necessary files to go from the raw data to the final results. One way to do this is to write a shell script that calls the submission files in sequence. An example of a simple shell script is shown below:



```
1  #!/bin/bash
2  python cleanData.py &&
3  Rscript processAnalyticData.R &&
4  jupyter nbconvert --execute buildModel.ipynb &&
5  Rscript -e "rmarkdown::render('outputPredictions.Rmd',output_file='output.html')"
```

Running the above script will execute each line, one after another. Note that the screen shot includes examples for many common languages. More background information on writing bash scripts is available at [this link](#). Of course, you may write the `run_all` script in the language of your choice so long as it can be executed from the command line.

While you are creating this script, we think it will be helpful to organize your input files, intermediate files, and output files into a standard directory (i.e., folder) structure. We think that this structure would help you create a modular research pipeline; see Peng and Eckel (2014) for more on the modularity of a research pipeline. This modular pipeline will make it easier for us to ensure computational reproducibility, and it will make it easy for other researchers to understand, re-use, and improve your code.

Here's a basic structure that we think might work well for this project and others:

data/

code/

output/
README
LICENSE

In the data/ directory you can include:

- background.csv (this should not actually be included because of privacy constraints, but we will put it here)
- train.csv (this should not actually be included because of privacy constraints, but we will put it here)
- Supplemental materials such as metadata files, the constructed-data dictionary, the machine-readable codebook.
- Data that you have collected or created, such as a csv file that you manually created that has your MSE scores on the holdout data and/or an analytic dataset created by your code.

In your code/ directory you can include:

- Executable “run-all” script that when run goes from raw inputs all the way to final outputs (for this script we encourage you to think about the research pipeline idea from Peng and Eckel 2014)
- Source code files each with a useful header (see FAQ).
- Package requirements
 - For python submissions, please include a requirements.txt file. More information [here](#).
 - For R submissions, please list all libraries utilized in a file named requirements_r.txt. Include each library name on a new line.

In your output/ directory you can include:

- prediction.csv
- A subdirectory for tables
- A subdirectory for figures (we also recommend including all data files that can be used to recreate the figures; see rule 7 of Sandve et al. 2013)

In addition to these three main directories, you should also include a README file and LICENSE file. We have more information about these files in the FAQ below. We hope that these guidelines are help, and please let us know if you have any questions.

Code resubmission process

Once you think you are ready to resubmit, here's a checklist that you can follow to help ensure that your work will be computationally reproducible:

- I have written the kind of README file that I would like to read (see FAQ below)
- Each code file that I've written has a header that will be helpful (see FAQ below)
- I've run the submission and I can get from raw files to final output using only materials in my directories. Then, I've done this again and I get the same result. This second step helps check for problems with seeding.
- I've considered refactoring my code (see FAQ below)

Finally, when you resubmit, we ask that you include a revision memo about the code, just as you will about the manuscript. This revision memo should summarize changes that you have made. In this revision memo, please also include a rough estimate of the cumulative amount of time it took you to comply with these guidelines. We are asking for this time estimate because one objection to computational reproducibility is that it is too burdensome for authors and we would like to assess this empirically. Finally, please include any suggestions for how this process could have been easier or more efficient.

F.A.Q.

What should go in the README file?

The README file should provide an overview of your code. For example, it could include a diagram showing the different pieces of their code, their inputs and their outputs. If relevant, please include expected warnings when executing the code. Mention any provided intermediate results readers can utilize to decompose the submission into smaller pieces.

The README should also include something about your computing environment and expected run time; general terms are appropriate here. For example: I ran this on a modern laptop (circa 2016) and it ran in a few minutes. or This code ran on high-performance cluster and took one week. Finally, please clearly cite any open sourced content utilized in the submission, such as resources shared in the FFC blog or more general packages distributed in the computation community.

What headers should be included at the top of each piece of code?

Based on the ideas in Nagler (1995), we think the following elements should be included at the top of each piece of code:

- Purpose (in 140 characters or less)
- Inputs
- Outputs
- Machine used (e.g., laptop, desktop, cluster)
- Expected runtime (e.g., seconds, minutes, hours, days, etc)
- Set the seed at the beginning of each file (see rule 6 of Sandve et al. 2013)²
- All the package include statements (e.g., “library(ggplot2)” in R)

²Note added afterwards: This is incomplete. It also helps to set seed inside of functions that have randomness. Also, there are different kinds of seeds. For example, in Python numpy uses a different seed from the standard python seed.

If you would like to deviate from this standard, please contact us.

How can I make my code easier to read?

It is hard to offer general advice, but one thing that we can recommend is at the end of the process take some time to refactor your code). In our experience, code evolves over the course of a project, and at the end it can be helpful to refactor in order to clean up the structure, improve variable names, and promote modularity.

Even if you don't refactor your code, please include additional comments to helper functions and code segments that may be obscure to new readers.

What is our standard for computational reproducibility for the special issue?

Our standard for computational reproducibility for this special issue is that we should be able to take whatever code and data you submit, add the Fragile Families Challenge data file, and then reproduce all of the figures in your paper, all of the tables in your paper, and your predictions.csv file.

What is not included in our standard for computational reproducibility for the special issue?

We will not attempt to completely recreate your analysis from the written materials. Also, we will not verify that your description in the paper matches the code. For example, if the paper says that you use logistic regression to generate your predictions, we will not verify that the code also uses logistic regression. Further, we will not verify the information that you have provided from external sources. For example, if you write in the paper that your submission was 10th on the leaderboard, we will not verify this fact. Finally, we will not verify any of the numbers that are included in the text of the manuscript. For example, we would not verify a claim in the text such as: dropping variables with no variation removes 10% of variables. As we hope this list illustrates, our standard of computational reproducibility is in fact quite limited.

What license should I use?

We strongly recommend the MIT license. You can find it here. Simply replace with 2018 and with the name of all co-authors of the paper, in the order they are listed in the paper. If you would like

to use some other license, please contact us.

What should I read to learn more about computational reproducibility?

Here's a partial list. If we've left off a good resource, please let us know (fragilefamilieschallenge@gmail.com).

Nagler (1995) "Coding Style Good Computing Practices" *PS: Political Science & Politics*.

Peng and Eckel (2009) "Distributed Reproducible Research Using Cached Computations" *Computing in Science & Engineering*.

Sandvae et al (2013) "Ten Simple Rules for Reproducible Computational Research" *PLOS Computational Biology*.

Stodden et al (2016) "Enhancing reproducibility for computational methods" *Science*.

4 Instructions for executing the code

The instructions below will walk you through executing the code for any Fragile Families Challenge submission featured in the special issue of *Socius*.

1. Install Docker

You can install Docker at: <https://www.docker.com/>. Simply navigate to the "Get Docker" tab.³ To verify that Docker is installed properly, open your favorite command line tool (e.g., Terminal for Mac or Command Prompt for Windows). You should be able to run `docker info` and see an output beginning with:

```
[ec2-user@ip-172-31-17-213 ffchallenge]$ docker info
Containers: 0
Running: 0
Paused: 0
Stopped: 0
```

2. Pull the FFC Docker Repository

Once you have installed Docker, you will be able to download and run the FFC code. Each of the

³Docker should run smoothly for machines running Linux, Mac OS, and Windows 10 Pro. If you are running an earlier version of Windows, we would recommend installing Docker onto an Amazon Web Services instance.

submissions has already been packaged into a Docker image containing all of the code as well as software / packages needed to execute. The images are labeled with the following tags:

First Author	Docker Image Tag
Rigobon, Daniel	rigobon_winning_2019
Filippova, Anna	filippova_humans_2019
Compton, Ryan	compton_data_2019
McKay, Stephen	mckay_when_2019
Carnegie, Nicole	carnegie_variable_2019
Stanescu, Diana	stanescu_using_2019
Raes, Louis	raes_predicting_2019
Altschul, Drew	altschul_leveraging_2019
Goode, Brian	goode_imputing_2019
Roberts, Claudia	roberts_friend_2019
Davidson, Thomas	davidson_black_2019

The instructions below will proceed with the image tag *stanescu_using_2019*. However, the reader should replace every instance of *stanescu_using_2019* with the desired Docker tag from the table above.

Download the desired image with the command:

```
docker pull 2018dliu/replication:stanescu_using_2019
```

This should give the output shown below:

```

[[ec2-user@ip-172-31-17-213 ~]$ docker pull 2018dliu/replication:stanescu_using_2019
stanescu_using_2019: Pulling from 2018dliu/replication
31d9d91516b8: Pull complete
d329eef6788a: Pull complete
0191ff0db9ad: Pull complete
bde9645eabc7: Pull complete
a453e71e5295: Pull complete
9f7a01d6fde4: Pull complete
0415a57c294d: Pull complete
30da06bf9510: Pull complete
0325c337472a: Pull complete
912a5c3ab3ce: Pull complete
e4addfd31bd9: Pull complete
e03a3143764a: Pull complete
0b0586f50e93: Pull complete
14a8fc0fb8c6: Pull complete
d60a244e19c8: Pull complete
0d8ecb5c7aea: Pull complete
94dd043a3370: Pull complete
Digest: sha256:e8aa9cb1a6597d2a4e61934d5aa38c66ddd8fc373c47f36a58d6a11359587c7a
Status: Downloaded newer image for 2018dliu/replication:stanescu_using_2019

```

3. Run the Image

The next step is to run the Docker image, which will launch a Docker container, an isolated computing environment, on your machine. Doing so simply involves typing:

```
docker run -t -d 2018dliu/replication:stanescu_using_2019
```

Those running the image tagged *davidson_black_2019* should use this command instead:

```
docker run -t -d -p 8888:8888
2018dliu/replication:davidson_black_2019
```

This command will expose one of the container's ports for Jupyter's notebook UI. Furthermore, those running the image tagged *rigobon_winning_2019* should use the following command:

```
docker run -t -d -p 8888:8888 --user root
2018dliu/replication:rigobon_winning_2019
```

4. Explore the submitted code

You can now dig inside the running container to access its contents. The first step is identifying the name of the container. To do so simply type the command: `docker container ls`.

You will receive output similar to:

```
[ec2-user@ip-172-31-17-213 ~]$ docker container ls
CONTAINER ID        IMAGE                                     COMMAND                  CREATED
3d4de315125f        2018dliu/replication:stanescu_using_2019  "sh"                    20 seconds ago
```

In the above output, the ID of the container is: 3d4de315125f. To access the container, run `docker exec -it [insert-container-ID] bash` (e.g.: `docker exec -it 3d4de315125f bash`) . After you run this command, you will be inside of the container and have access to all of the necessary code and dependencies. You can run "ls" to view all of the files that were submitted to the Challenge.

The README files contain instructions for executing the code. For several of the submissions, we have supplemented the author's README with one of our own, providing additional execution instructions. In all cases, you must first move the Fragile Families Challenge data files into the container. You can move the files using the command `docker cp`, which will copy the data files from the local file system into the container. Then execute the `run_all` script to trigger the computing pipeline. This file can be found in either the `root` or `code` directory.