Introduction

LDI Replication Lab

THE Manual

This document is meant to guide the **training** of students in assessing the reproducibility of articles in the social sciences, and in particular in economics. It is also a **reference** to come back to while those same students are working on specific reproducibility checks.

How to read this document

If you are a casual reader of this document, start with the pre-requisites. If you are a student participating in our training, then start with the pre-training tasks. Read the rest sequentially.

> **Important**
>
> Once trained, jump right to the **AEA Jira Workflow!**

Slides

For training purposes, we use a set of slides.

Printable version

A printable version of this document is available here .

> **Warning**
>
> We discourage printing the full document, as it is over 250 pages long.

Improving this document

File an issue! (Github account required) You can do this from the GitHub issues page, or, for more precision, from each top-right corner of the page (click on the Octocat icon).

Re-using this document

Please feel free to adapt this document, or parts of it, for your own purposes.

Background

On July 16, 2019, the AEA announced an updated "Data and Code Availability Policy" [American Economic Association, 2019, American Economic Association, 2020]. Henceforth, replication materials were to be made available to the AEA *prior* to acceptance - previously, it was prior to publication, but *after* acceptance. Computer code should be provided for all stages of the data cleaning and data analysis (code for the data cleaning portion was previously optional). Raw data must be uniformly made available, when permissions allow (also for author-collected survey data, data from experiments). For restricted-access or proprietary data, to the extent permissible, the data must be made available to the AEA Data Editor for verification, even if the data cannot be published by the author.[1] Enforcement of an existing but unenforced **data citation requirement** as per AEA Citation Guidelines.

We also test licenses, access restrictions, and sometimes question the ability of authors to publish the data. We have had cases both ways: data provided after initial refusal, and data rejected by us because the license did not allow distribution.

We also check for obvious personally identifiable information (PII). However, the ultimate responsibility lies with authors.

All data and code must be available in a "trusted repository," which in most cases means the AEA's Data and Code Repository at openICPSR, for better transparency and findability. ZIP files are no longer accepted as supplementary packages, and we check that. Supplements are tagged with JEL codes, other keywords (e.g., "`Current Population Survey`" or "`behavioral study`"), and optionally with methodological information (time period, geographic region, survey method used). Each deposit gets its own DOI—a permanent unique identifier. Deposits can be found through various search engines, such as the native search engine on openICPSR, through Google Data Search, or through DOI registries such as DataCite.

To implement all this, we built a system using Jira, and connect to it from ScholarOne (system used for manuscript submission) and openICPSR.

Activities of the LDI Replication Lab

The LDI Replication Lab conducts reproducibility checks in two way.

Pre-Publication Evaluation of Reproducibility and Quality of Supplemental Materials

The Lab performs pre-publication evaluation for the American Economic Association, i.e., prior to publication. Think of it as a "reviewer" of the data supplement.

- Analyze the provided materials - see Verification guidance
- Verify data citations
- Verify ability to post data (do the authors have the right to post the data?)
- If possible, attempt replication

Post-Publication Evaluation of Reproducibility

We download articles and supplements, assess to what extent an undergraduate student can run the code that produces the analysis reported in the paper. We have also in the past done an evaluation of the response of authors when the code is only available "upon request". Not currently an active project.

This is a secondary goal, as time permits or as research goals suggest. It is quite similar to the first goal, but there is no interaction with authors, and no method to improve authors' files.

> Interested parties might visit ACRE for how to incorporate this kind of activity into a class curriculum.

Learning goals

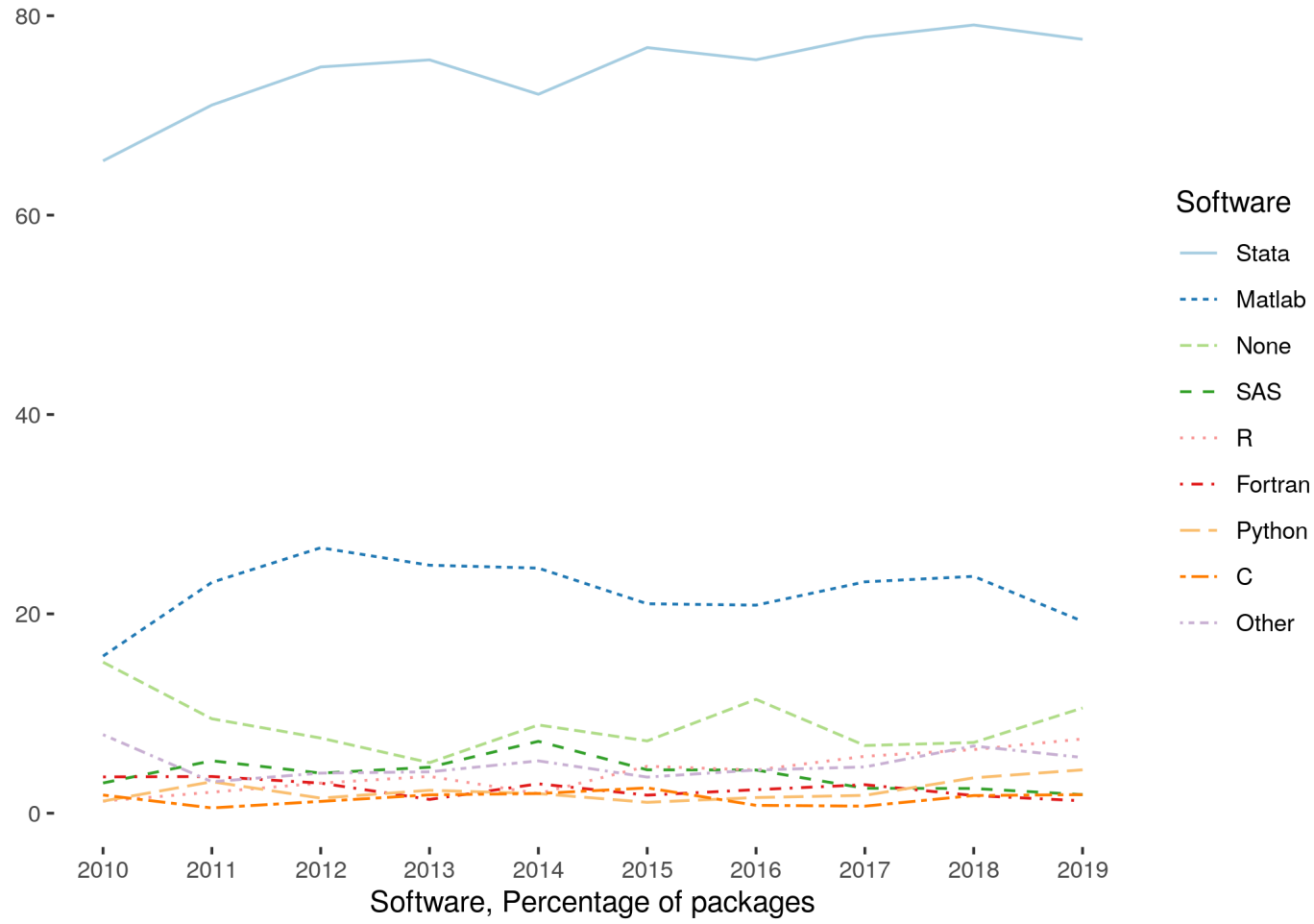
In this training, you will learn how to verify compliance with the policy. This means going through a variety of checklists, obtaining data, and running code, as per instructions provided by authors. You will not be required to actively program, but you will learn a lot about how to program (and sometimes, how not to program). You will gain an appreciation for a well-structured empirical analysis, which you will benefit from for your own studies (now) and in your work (after graduation).

---

[1] We regularly and successfully do so.

Pre-requisites

Most students with some **prior experience with statistical software** can effectively reproduce and assess articles. In economics, most articles use either Stata or Matlab.



Students should be comfortable working on a variety of computers, including their own, and be flexible with respect to the location of computing. If you don't know what that means, you'll find out during training!

Pre-training tasks

We ask that trainees accomplish a few tasks prior to the first training session. Please do the following:

Read

- We frequently meet virtually - at least one, and most of the time both of our weekly meetings are on Zoom. Review our Video Etiquette rules (they are useful beyond our group as well). When training is virtual, these rules apply as well.
- Review our Privacy Policy, where you will recognize how we handle your privacy, and the privacy of authors.

View

- View my talk on the background of the lab, including what we do, and why we do it.

Install

- Go through our Setup Checklist and install necessary software

Try-out

- Try out command line, Markdown, Git (we'll come back to it on the first day).

Toolkit

In this chapter, we will introduce you to a few of the toolkits that we will use for this project, and that you may find useful for most future data and code work.

Command line

The shell (command line) is a power tool that allows you to do complex things with just a few keystrokes. You can combine and chain things together. Later in your studies, you may find that it is a fundamental tool used in combination, or even core, to other powerful tools and computing resources (including "high-performance computing" supercomputers).

For our purpose, there are a few key features that are important:

- it is present on all the computers you will be working on (with the possible exception of the Bash shell on Windows, which we will get to shortly).
- it allows us to show you how things work without paying attention whether you are on a Mac, Linux, or Windows, or if you prefer one or the other fancy windowed program.

We will leverage the excellent lessons provided by the Software Carpentries, somewhat optimized for our purposes.

> - Go to our customized shell lesson
> - See the full lesson at Carpentries

Note: we will walk you through the basics very quickly, but you will want to try this out later, on your own time, before we get to the Git lesson later.

Markdown

(based on this Carpentries tutorial)

Markdown is a lightweight markup language, i.e. **a convention for adding style information to textual content**.

Having a rather minimalistic syntax, text formatted in Markdown is comparably readable. This might be one reason for Markdown having become the language of choice for formatted user input on websites like, for example:

- Stack Exchange
- GitHub
- GitLab.

Where to Start Writing Markdown?

A lot of tools for rendering Markdown source code exist. Rendering is the process of generating a nice view of the content using the style information included in the source text. Chances are high, your editor can do this. Chances are, many websites use and render it. It is the native way to add formatting on Github, Bitbucket, etc.

In our case, we will use Visual Studio Code to write and preview Markdown. All of our reports will be written in Markdown. This tutorial is written in Markdown!

> - Open up Visual Studio Code,
> - Add the line `# My Title`
> - and save as a new file called 'README.md'

Let's add `Some **bold** font` and see what happens when we preview it using the preview button.

> Where is the Preview button? Read the introduction to the Visual Studio Code: Writing Markdown article to find out!

If new sections were added you will also find green vertical bars visually highlighting the new content.

Writing Markdown

Now that we know about the editing interface and preview tab of our projects `README.md` we can use it as a text editor and investigate selected Markdown features.

Our `README.md` already contains vanilla text and two formatting features:

- Heading `# My Title`
- Emphasis using `**bold**`.

Let's learn some more Markdown by adding some formatting and see what happens when we preview it in the preview tab. Add the following to your `README.md` file.

```
# My Title

Repo for learning how to write Markdown

## Learning Markdown

Vanilla text may contain *italics* and **bold words**.

This paragraph is separated from the previous one by a blank line.
Line breaks
are caused by two trailing spaces at the end of a line.

[Carpentries Webpage](https://carpentries.org/)


### Carpentries Lesson Programs:
- Software Carpentry
- Data Carpentry
- Library Carpentry
```

You can see in the preview tab again to see how the formatting renders.

> ## Markdown trailing spaces are meaningful
>
> In the example above there are two spaces at the end of `Line breaks  `. These introduce what is called a **hard line break**, causing that paragraph to continue in the next line by adding a `<br/>` to the generated HTML.
>
> If you break the line in a markdown file but don't include the two trailing spaces the generated HTML will continue in the same line **without** introducing a `<br/>`. This is called a **soft line break**.
>
> In some cases you may find that **soft line breaks** do introduce a `<br/>`. This can happen when using different [markdown flavors].
>
> See for instance:
>
> ```
> Soft line
> break
>
> Hard line
> break
> ```

Exercise: Try Out Markdown

Online Markdown Tutorial

Four reasons you should learn Markdown:

1. Less formatting than HTML
2. Easy to read even with formatting
3. Commonly used for websites and software development
4. We use it in our reports!

Git

Git is a version control system, which we use extensively in the LDI Replication Lab.

Why a version control system?

You can review history of code or text to find out:

- *Which changes were made?* Identify the lines that have been changed.
- *Who made the changes?* If multiple people work on a project, who made the particular change. Also tabulate contributions to the project by person.
- *When were the changes made?*
- *Why were changes needed?* The ability to attach *log messages* provides a lot of information.

Key Git Concept: repository

A *repository* = *Git project* is a collection of files and folders, along with each file's revision history. There are

- *commits*: collections of additions or modifications, a snapshot
- *branches*: versions or variants of the main repository
- *clones*: Each copy of the repository is complete - fully functional
- *staging area*: an area on your computer where files and changes are collected prior to finalizing a *commit*

Key Git Concept: Github or Bitbucket?

There are multiple services that host repositories, and that can serve as a central reference point for these repositories.

- You *push* to another repository - that could be on your buddy's computer, to a central entity (Github, Bitbucket) - however it is defined.

- You *pull* changes from another repository.
- A *pull* + *push* is sometimes referred to as a *sync* (synchronisation)

Basic Git Commands

(also see the Cheatsheet for Git)

- `git clone` creates a local copy of a project that already exists remotely. The clone includes all the project's files, history, and branches.
- `git add` stages a change. Git tracks changes to a working copy, but it's necessary to stage and take a snapshot of the changes to include them in the project's history. This command performs *staging*, the first part of that two-step process. Any changes that are staged will become a part of the next snapshot and a part of the project's history.
- `git commit` saves the snapshot to the project history and completes the change-tracking process. In short, a commit functions like taking a photo. Anything that's been staged with `git add` will become a part of the snapshot with `git commit`.
- `git status` shows the status of changes as untracked, modified, or staged.
- `git pull` updates the local line of development with updates from its remote counterpart. You use this command if a teammate has made commits to a branch on a remote, and they would like to reflect those changes in their local environment. Or you made a commit on your laptop, and now want to continue the work on a remote desktop server.
- `git push` updates the remote repository with any commits made locally to a branch. Think of it as publishing your changes.

A Tutorial

We will again leverage the excellent lessons provided by the Software Carpentries, somewhat optimized for our purposes.

> - Go to our customized Git lesson
> - See the full lesson at Carpentries

Some additional resources

- General resources
- Git tutorial

Basic Concepts

The goal of the reproducibility activity is

- to verify provenance of all data sources,
- to verify the availability of all computer code to produce analysis files from data sources, and to produce tables, figures, and in-text numbers from analysis files,

- to verify that the available code actually reproduces the analysis files, tables, figures, and in-text numbers.

We therefore need to define a few concepts:

- what is a data source, and how can one verify its provenance?
- what is analysis data?
- how can we verify computational reproducibility?

The subsequent chapters will explore each of these in additional detail.

Reproducible practices

At the Lab, we evaluate researchers' adherence to reproducible practices, and their ability to produce a publishable and **reproducible** package encapsulating their work. Many researchers are challenged with this. The Lab, to be credible, and for clarity of our findings, also needs to work with **reproducible** practices. You will be challenged by this.

> **Note**
>
> Slides on this topic. are available.

Some additional recommended readings and presentations

These are primarily directed at authors and researchers, but you should be aware of what is feasible and practicable, since often, you will be in the role of recommending such practices to authors in order to fix problems you identified.

- Applying reproducible practices from Day 1
- Self-checking reproducibility (Presentation)

The README template

Researchers are encouraged (sometimes required) to adhere to a particular documentation standard. We have found that providing a template is useful to allow for complete coverage of all the topics a researcher should address in their README, which describers what to do with their package. The typical replication package needs information on data (provided or not), code (how to run it, and what is required), and the computational infrastructure necessary to be able to run the code.

> **Note**
>
> Presentation of this section

Additional reading

Peruse the template README provided by a collective of data editors. You will hold researchers to this standard.

- README template (archived version)

- Writing a clear README file

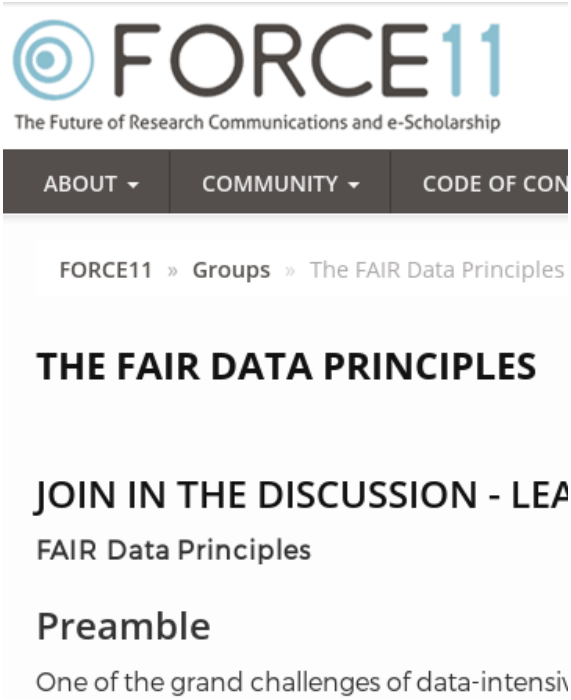
Data citations and data availability statements

Most students and researchers have been trained to cite their sources. Mostly, this is meant to be literature sources, but the basic idea applies just as much to data: If you use somebody else's data, you should acknowledge that.

> **Note**
>
> See the relevant presentation slides.

A very short history of data citations

The almost complete absence of data citations from the literature has lead to issues when data creators and data providers attempt to assess their impact on science. Typically, they revert to manually or algorithmically scouring the literature, trying to find instances where their data is used.



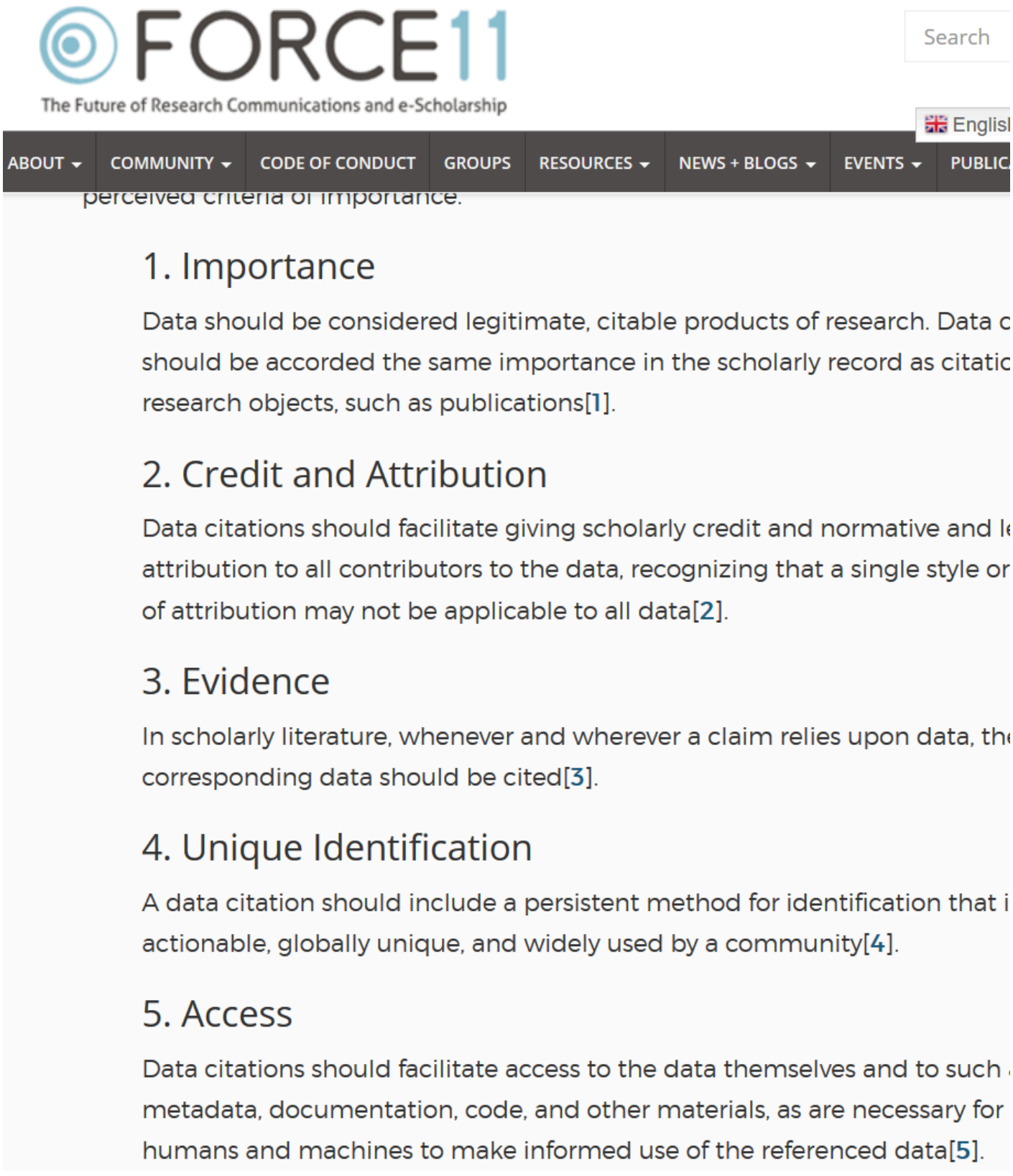
In 2016, a number of scientists and publishers from many domains got together and issued the FAIR Data principles [FORCE11, 2016]. "FAIR" is an acronym for

- Findable,
- Accessible,
- Interoperable, and
- Re-usable

These are principles which help the furthering of increased reproducibility - by making data accessible, reproducibility checks can be conducted. By making them interoperable and re-usable, others can reproduce, replicate, and expand on the original

research. Findability remains an issue: how do you discover data that is hidden in a ZIP file on a journal website as part of an (otherwise) perfectly reproducible package?

In fact, the same group had previously laid the groundwork to that. In 2014, they issued the "Data Citation Principles (DCP)" [Martone, 2014]. These laid out the ethical need for such citations, the content such citations should have, and some desirable attributes of data citations. Both the DCP as well as FAIR suggest that data be assigned unique identifiers, allowing them to be clearly referenced, and found. The most common identifier in the social sciences is the Digital Object Identifier (DOI), but others exist, in particular in the biological or physical sciences.



The DCC note that (emphasis added):

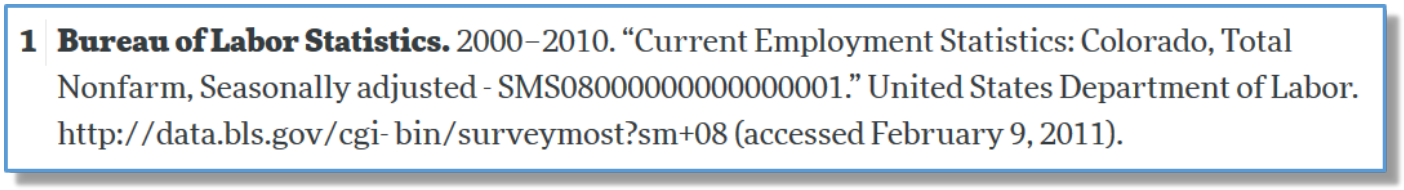
> Sound, reproducible scholarship rests upon a foundation of robust, accessible data. For this to be so in practice as well as theory, data must be accorded due importance in the practice of scholarship and in the enduring scholarly record. In

> other words, data should be considered legitimate, **citable products of research**. Data citation, **like the citation of other evidence and sources**, is good research practice and is part of the scholarly ecosystem supporting data reuse.

Data citation increases the findability, accessibility, interoperability, and re-usability of research data (FAIR). Through data citations, data providers can link to articles (sometimes automatically), allowing them to show the academic value of the data and continue providing the services around data creation. Finally, data citations open a new path to finding relevant science, by reaching the linked articles through data search interfaces, like openICPSR, Data-Pass, and Google Dataset Search.

These principles are imperfectly implemented even today. Many data providers, including some of the biggest statistical agencies, do not provide unique identifiers to a particular data file. Some might uniquely identify a data series. Others may heuristically refer to certain versions ("V2"). Few have robust archival quality unique identifiers.

Journals' stylistic considerations also reduce the impact of the DCC. Citations - in the sense of appearing in a bibliography at the end of the document - do not provide much sense of any access method other than a download. The AEA follows the Chicago Manual of Style (CMOS), with several additions on the AEA website.


1 **Bureau of Labor Statistics.** 2000–2010. "Current Employment Statistics: Colorado, Total Nonfarm, Seasonally adjusted - SMS08000000000000001." United States Department of Labor. http://data.bls.gov/cgi- bin/surveymost?sm+08 (accessed February 9, 2011).

As the CMOS states, one of the criteria for a useful citation is conveying authority and permanence:

> Electronic content presented without formal ties to a publisher or sponsoring body has the authority equivalent to that of unpublished or self-published material in other media.

FAIR data helps convey such information, but often, assessing what "publisher" or "sponsoring body" is reputable or reliable is tricky. And the citation fails to convey many important facets of data access. What if the cited resource requires a login, even if it is free? What if payment is required? What if a long-winding application process is required? Citations do not communicate that amount of detail. As we shall see later, there are ways to augment data citations with the needed information.

How do you create a data citation

ICPSR [ICPSR, n.d.] notes that a citation should include the following items:

- Author
- Title
- Distributor
- Date
- Version
- Persistent identifier

where any data source has the first five elements, and the ideal FAIR curated data source also has a persistent identifier.

An imperfect real example

Consider the BLS citation shown earlier:

> Bureau of Labor Statistics. 2000–2010. “Current Employment Statistics: Colorado, Total Nonfarm, Seasonally adjusted - SMS08000000000000001.” United States Department of Labor. http://data.bls.gov/cgi-bin/surveymost?sm+08 (accessed February 9, 2011).

The *author* or creator is clearly `Bureau of Labor Statistics`. The *distributor* here is `United States Department of Labor`, which happens to be the government department housing the BLS. *Title* is arguably `Current Employment Statistics: Colorado, Total Nonfarm, Seasonally adjusted`, though some might argue that state, coverage, and seasonal adjustment identify *versions* of the survey. *Date* are the (original) dates of publication, or here, approximated by the date range covered by the series. But generally, *versions* relates to a chronologically released version - which is less clear in this case, and only captured by `(accessed February 9, 2011)`. There is no clear *persistent identifier*, the closest approximation is provided by a combination of the series identifier `SMS08000000000000001`, the URL `http://data.bls.gov/cgi-bin/surveymost?sm+08`, and the date accessed `(accessed February 9, 2011)`. Note that this is imperfect, because, unless you can time-travel, you cannot obtain the same dataset a second time.

Attribute	Value
Author:	`Bureau of Labor Statistics`
Title:	`Current Employment Statistics: Colorado, Total Nonfarm, Seasonally adjusted`
Distributor:	`United States Department of Labor`
Date:	2000-2010
Version:	`(accessed February 9, 2011)`
Persistent identifier:	`SMS08000000000000001` + `http://data.bls.gov/cgi-bin/surveymost?sm+08` + `(accessed February 9, 2011)`

An ideal (?) data citation

What should the ideal data citation look like? ICPSR suggests

> Barnes, Samuel H. Italian Mass Election Survey, 1968. Ann Arbor, MI: Inter-university Consortium for Political and Social Research [distributor], 1992-02-16. https://doi.org/10.3886/ICPSR07953.v1

so that

Attribute	Value
Author:	`Barnes, Samuel H.`
Title:	`Italian Mass Election Survey, 1968`
Distributor:	`Inter-university Consortium for Political and Social Research`
Date:	`1992-02-16`
Version:	`V1`
Persistent identifier:	`10.3886/ICPSR07953.v1`

Note that the date `1968` describes the survey, but its date of publication was 1992. The version is implicit in the DOI. The persistent identifier is just `10.3886/ICPSR07953.v1`, but the display guidelines for DOI suggest including the full URL that yields a resolution.

Data citation template

In general, therefore, as long as you can fill out the table

Attribute	Value
Author:	
Title:	
Distributor:	
Date:	
Version:	
Persistent identifier:	

you can create a data citation:

> [AUTHOR], “[TITLE]”, [DISTRIBUTOR], [DATE], [VERSION] + [Persistent Identifier]

A not quite serious example

Many authors initially neglect to add data citations, or do not know how to add a data citation. Often, we see authors cite papers with supplementary data, but not databases or other data:

> We use data acquired from the NHL, dates of power outages collected by Tremblay et al (2018), augmented with information on the language and grammar skills of hockey players provided by the Ethnologue database.

(note absence of citation for NHL and Ethnologue data). In the above example, three datasets are used, but only one is cited in some fashion.

Better

The above example can be improved as follows:

> We use data acquired from the NHL (NHL, 2018), dates of power outages collected by Tremblay et al (2018, 2019), augmented with information on the language and grammar skills of hockey players provided by the Ethnologue database (Eberhard et al, 2019).

with the reference list having the following entries:

> - Eberhard, David M., Gary F. Simons, and Charles D. Fennig (eds.). 2019. *Ethnologue: Languages of the World.* Twenty-second edition. Dallas, Texas: SIL International. Online version: http://www.ethnologue.com.
> - National Hockey League. 2018. *NHL Game Database 1917-2018*. National Hockey League Hall of Fame, Toronto, ON. Accessed February 29, 2019.
> - Tremblay, Réjean, Ken Dryden, and José Theodore. 2018. "The impact of power outages on goal-keeping in the NHL", *Journal of National Hockey Leagues*, vol 32, iss. 1.
> - Tremblay, Réjean, Ken Dryden, and José Theodore. 2019. "Power outages during NHL games (updated)", *Canadian Hockey Dataverse*, doi:10.1234/nhl.lnh.haha

Assess why the latter is better.

Data distributed as supplementary data

The CMOS provides examples of how to cite supplementary materials that are attached to a specific article:

> Suárez-Rodríguez, M. and C. Macías Garcia. 2014. "There Is No Such a Thing as a Free Cigarette: Lining Nests with Discarded Butts Brings Short-Term Benefits, but Causes Toxic Damage." *Journal of Evolutionary Biology* 27, no. 12 (December 2014): 2719–26, https://doi.org/10.1111/jeb.12531, data deposited at Dryad Digital Repository, https://doi.org/10.5061/dryad.4t5rt.

The AEA guidance used to provide an example, in which the citation links to the article landing page:

> Romer, Christina D., and David H. Romer. 2010. "The Macroeconomic Effects of Tax Changes: Estimates Based on a New Measure of Fiscal Shocks: Dataset." *American Economic Review.* https://doi.org/10.1257/aer.100.3.763.

Many authors, however, would only cite the article itself, and not the data. Note however that modern data citation guidance suggest that both the article and the data used by the article should be cited, and this can lead to confusion. With the 2019 move of the AEA to a data archive, the correct citation for the above supplement would be:

> Romer, Christina D., and David H. Romer. 2010. “Replication data for: The Macroeconomic Effects of Tax Changes: Estimates Based on a New Measure of Fiscal Shocks.” *American Economic Association [publisher]*, *Inter-university Consortium for Political and Social Research [distributor]*, https://doi.org/10.3886/E112357V1

with the article also cited as:

> Romer, Christina D., and David H. Romer. 2010. “The Macroeconomic Effects of Tax Changes: Estimates Based on a New Measure of Fiscal Shocks” *American Economic Review.* no. 3 (June 2010): 763–801. https://doi.org/10.1257/aer.100.3.763.

Producer

Often, the creator of a dataset is an organization. The same way that an organization as a work’s author can be cited:

> ISO (International Organization for Standardization). 1997. *Information and Documentation—Rules for the Abbreviation of Title Words and Titles of Publications*. ISO 4:1997. Paris: ISO.

an organization can be cited as the creator of a dataset:

> Standard and Poor’s (S&P). 2017. *Compustat-Capital IQ*. S&P Global Market Intelligence.

Distributor

In many cases, the data are not distributed by the creator. This means the *distributor* takes on the role of a *publisher* (of a book, of data). In the BLS example, the two differed only because the (higher-ranking) department counts as the distributor. In the case of Compustat, one might have obtained access through the Wharton Research Data Services, and cite as

> Standard and Poor’s (S&P). 2017. *Compustat-Capital IQ*. Wharton Research Data Services. https://wrds-www.wharton.upenn.edu/pages/about/data-vendors/sp-global-market-intelligence/

If using the S&P 500 data, there may be multiple providers:

> S&P Dow Jones Indices LLC, *S&P 500 [SP500]*, retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/SP500, January 24, 2020.

> S&P Dow Jones Indices LLC, *S&P 500*, provided via Haver Analytics Data Subscription, February 24, 2018.

with hopefully the same content. Note that often, such data is subject to copyright and redistribution restrictions (see the page at FRED on SP500 and discussion in the later section).

Dates

In some cases, it isn't clear when the dataset was *published*, though it may be clear what time period the dataset covers (as in the BLS case). One way to address this may be by using the "n.d." abbreviation for the date of publication and including the date of coverage in the title:

> Standard and Poor's (S&P). n.d. *Compustat-Capital IQ (1982-2017)*. Wharton Research Data Services. Accessed April 6, 2018. https://wrds-www.wharton.upenn.edu/pages/about/data-vendors/sp-global-market-intelligence/

A related issue may arise when the dataset is comprised of multiple years, each of which has its own DOI. For instance, when accessing multiple years of American Community Survey data on ICPSR, each has its own DOI:

American Community Survey (ACS): Public Use Microdata Sample (PUMS),	1998	(ICPSR 3888)	2008-05-21
American Community Survey (ACS): Public Use Microdata Sample (PUMS),	1997	(ICPSR 3886)	2008-05-21
American Community Survey (ACS): Public Use Microdata Sample (PUMS),	2003	(ICPSR 4117)	2009-12-01
American Community Survey (ACS): Public Use Microdata Sample (PUMS),	2004	(ICPSR 4370)	2008-10-14
American Community Survey (ACS): Public Use Microdata Sample (PUMS),	2009	(ICPSR 33802)	2013-04-04
American Community Survey (ACS): Public Use Microdata Sample (PUMS),	2008	(ICPSR 29263)	2011-11-08

One approach to this is to create a composite citation, with additional information available in an online data appendix or a Data Availability Statement:

> Bureau Of The Census. 2009. "American Community Survey (ACS): Public Use Microdata Sample (PUMS), 1997-2009." *United States Department Of Commerce* [publisher]. ICPSR - Interuniversity Consortium for Political and Social Research. [distributor] DOIs listed in data appendix.

or

> Bureau Of The Census. 2009. "American Community Survey (ACS): Public Use Microdata Sample (PUMS), 1997-2009." *United States Department Of Commerce* [publisher]. ICPSR - Interuniversity Consortium for Political and Social Research. [distributor] https://www.icpsr.umich.edu/icpsrweb/ICPSR/search/studies?q=american+community+survey (accessed November 21, 2019)

(and listing of exact DOIs in an appendix table).

Offline access mechanism

Many datasets are available only under license, memorandum, contract, etc., and do not have a formal online presence. This is quite similar to traditional offline archives, for instance manuscript collections. For such collections, CMOS suggests:

> Kallen, Horace. *Papers*. YIVO Institute for Jewish Research, New York. Merriam, Charles E. *Papers*. Special Collections Research Center, box 26, folder 17. University of Chicago Library.

and usage in the text as

> Alvin Johnson, in a memorandum prepared sometime in 1937 (Kallen Papers, file 36), observed that …

Similar citations can be constructed for offline databases:

> Bloom, Nick. 2019. *Confidential survey data on Cameroon business processes*. Stanford Secure Access Center (file "cameroon-bloom.zip"). Stanford University.

Confidential databases

Similar forms may be used for confidential databases when no DOI exists:

> Internal Revenue Service. (YEAR). *Corporate Income Tax Returns [database]*. Department of Treasury, Washington DC, accessed YYYY-MM-DD.

where the data, in this case, were accessed via the "Department of Treasury," acting as a *secure* distributor (of access, not downloads). If the same data had been accessed via a secure research data center, the reference should have instead noted that access mechanism:

> Internal Revenue Service. (YEAR). *Corporate Income Tax Returns [database]*. Federal Research Data Centers, last accessed YYYY-MM-DD.

Confidential data with DOI

If a DOI exists, of course, the formal citation generated from that DOI should be used:

> Forschungsdatenzentrum der Bundesagentur für Arbeit. 2020. "Betriebs-Historik-Panel (BHP) – Version 7518 v1." *Institut für Arbeitsmarkt- und Berufsforschung (IAB)*. https://doi.org/10.5164/IAB.BHP7518.DE.EN.V1.

No formal access mechanism

In some cases (not infrequently), access to data is through informal means. The CMOS allows for citation of such information, without inclusion in the references.

> (A. P. Møller, unpublished data; C. R. Brown and M. B. Brown, unpublished data)

We would deviate from that suggestion, ask for inclusion in the reference list, and simply suggest using *unpublished data* as the locator, similar to a URN, in the reference list:

> Møller, A. P. n.d. “Data on Crocodile Sightings in Manhattan.” unpublished data. Accessed February 29, 2019.

Unknown or confidential author

In some cases, the authors might not be able to name the data creator, due to a non-disclosure agreement. One suggestion may then be

> Anonymous Firm, n.d. “Data on financial transactions.” Accessed under Non-disclosure Agreement, extract obtained on January 20, 2016.

Where to cite

In all cases, data and code should be cited in the main manuscript. They should also be referenced in the data availability statement (some journals) or the README (other journals). However, in some cases, data is only used in an online appendix, and it is acceptable to cite the data there as well, and not in the main manuscript’s bibliography.

Furthermore, as data citation are still a relatively new concept, many authors will have substantially completed their manuscript, without including data citations. Adding them to the README is then acceptable practice (for now).

Data availability statements

The academic publishing community’s response are “data availability statements (DAS).” While mostly, these are pointers from the journal to where the data can be found. In the case of data supplements, this is almost trivial when the journal has a robust data availability policy, though some journals allow for self-declared but unverified DAS.

Summarily, a data availability statement describes not just where the data can be obtained from, but also how the data can be obtained, conditions for obtaining it, and any additional restrictions.

> Some examples are provided by Springer/Nature and Hindawi:
>
> - Springer
> - Hindawi

Some examples of DAS

Example for public use data included in data archive:

> The paper uses data obtained from IPUMS (Ruggles et al, 2017). IPUMS-CPS does not currently provide the ability to store or reference custom extracts, but allows for redistribution for the purpose of replication. The archive contains the extracted data, codebook in the folder “data/IPUMS”. The data citation in the main article has the full URL.

Example for public use data not included in data archive:

> Data from the Socioeconomic High-resolution Rural Urban Geographic Dataset on India, Version 1.0 (Asher and Novosad, 2019) is used in this paper. The full dataset and documentation can be downloaded from https://doi.org/10.7910/DVN/DPESAK.

Example for public use data with required permission:

> The paper uses IPUMS Terra data. IPUMS-Terra does not allow for redistribution, except for the purpose of replication archives. Permissions as per https://terra.ipums.org/citation have been obtained, and are documented within the “data/IPUMS-terra” folder.

Example for confidential data:

> The data for this project are confidential, but may be obtained with Data Use Agreements with the Massachusetts Department of Elementary and Secondary Education (DESE). Researchers interested in access to the data may contact [NAME] at [EMAIL], also see www.doe.mass.edu/research/contact.html. It can take some months to negotiate data use agreements and gain access to the data. The author will assist with any reasonable replication attempts for two years following publication.

Example for Government registers

In some cases, governments have a list of their (named) registers. For instance, Statistics Denmark provides the full list of registers at http://www.dst.dk/extranet/forskningvariabellister/Oversigt%20over%20registre.html. These can be used to craft data citations (see Data citation guidance). Data availability statements should describe how each such register can be accessed:

> The information used in the analysis combines several Danish administrative registers (as described in the paper). The data use is subject to the European Union’s General Data Protection Regulation(GDPR) per new Danish regulations from May 2018. The data are physically stored on computers at Statistics Denmark and, due to security considerations, the data may not be transferred to computers outside Statistics Denmark. Researchers interested in obtaining access to the register data employed in this paper are required to submit a written application to gain approval from Statistics Denmark. The application must include a detailed description of the proposed project, its purpose, and its social contribution, as well as a description of the required datasets, variables, and analysis population. Applications can be submitted by researchers who are affiliated with Danish institutions accepted by Statistics Denmark, or by researchers outside of Denmark who collaborate with researchers affiliated with these institutions.

(Example taken from Fadlon and Nielsen, 2020 (forthcoming as of June 2020).

S&P 500

The S&P 500 is one of the most widely known stock indexes. And yet, it is subject to copyright, and restrictions on redistribution. Some authors who use the S&P 500 numbers may have downloaded it via FRED https://fred.stlouisfed.org/series/SP500, others through other data services (Haver Analytics, Bloomberg). The FRED website mentions that the data are

> (C) S&P Dow Jones Indices LLC. Reproduction of S&P 500 in any form is prohibited except with the prior written permission of S&P Dow Jones Indices LLC ("S&P").

If obtained through FRED, the suggested citation is

> S&P Dow Jones Indices LLC, *S&P 500 [SP500]*, retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/SP500, January 24, 2020.

An analogue if accessing it through, say, Haver Analytics, might be

> S&P Dow Jones Indices LLC, *S&P 500*, provided via Haver Analytics, January 24, 2020.

Note that both citations do not provide (complete) information on how others might obtain the data, and what restrictions are imposed on the data (unless they visit the site, or read the terms of use of, say, their Haver Analytics description). A tentative Data Availability Statement might be:

> S&P 500 data is (C) S&P Dow Jones Indices LLC. Reproduction of S&P 500 in any form is prohibited except with the prior written permission of S&P Dow Jones Indices LLC ("S&P"). It is thus not available as part of the replication archive. Users may access the past 10 years via FRED at https://fred.stlouisfed.org/series/SP500, or purchase longer access via Haver Analytics (http://www.haver.com/databaseprofiles.html#indicators). Note that reproduction of our manuscript's tables requires data from [YYYY]-[ZZZZ].

Generic data workflow

In a very generic, all empirical analysis goes through the following steps:

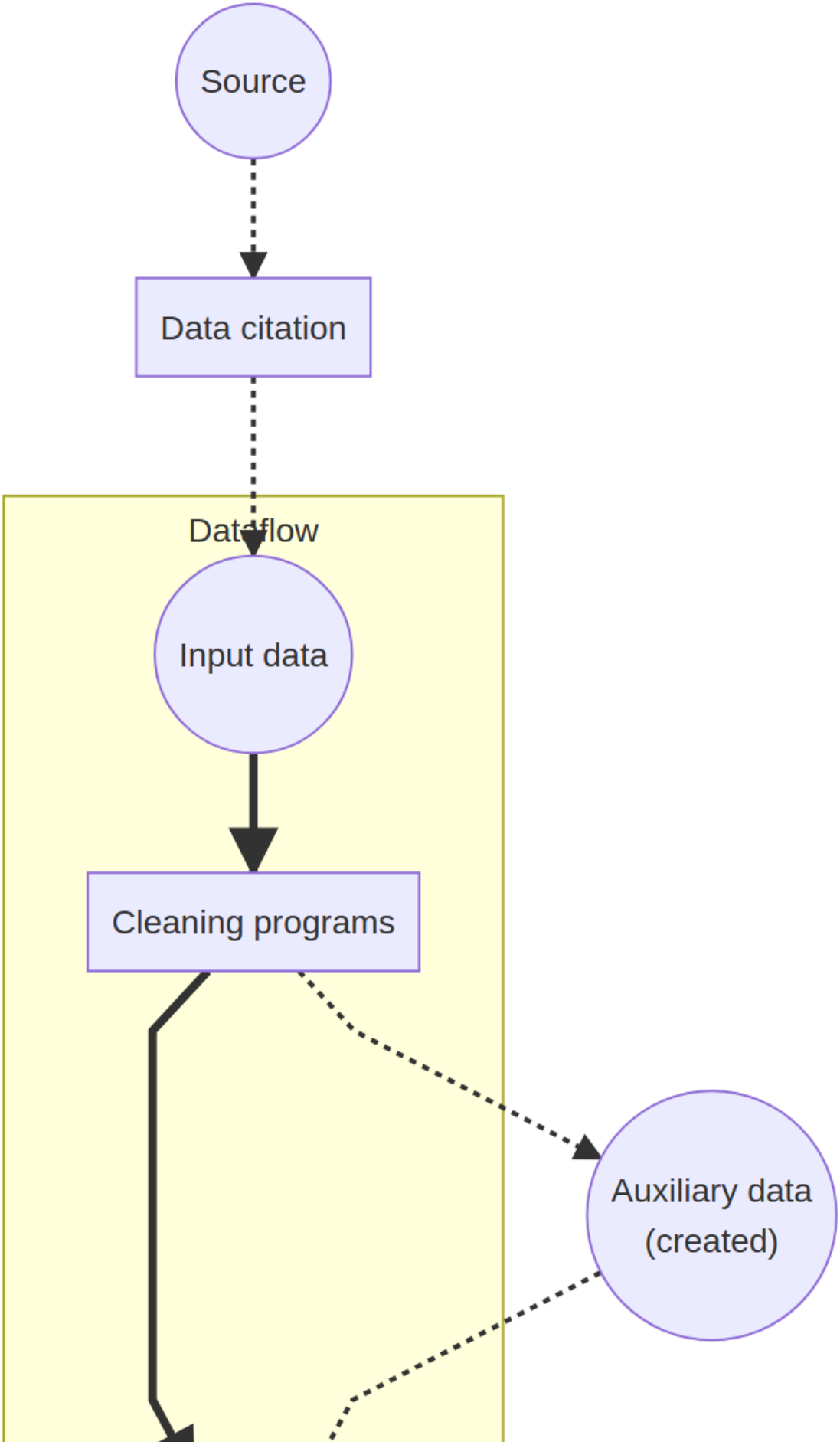
- acquiring data (primary or secondary)
- cleaning data (reformatting, standardizing, dropping) to create analysis data
- analyzing data and reporting results in tables, figures, and in-text numbers.

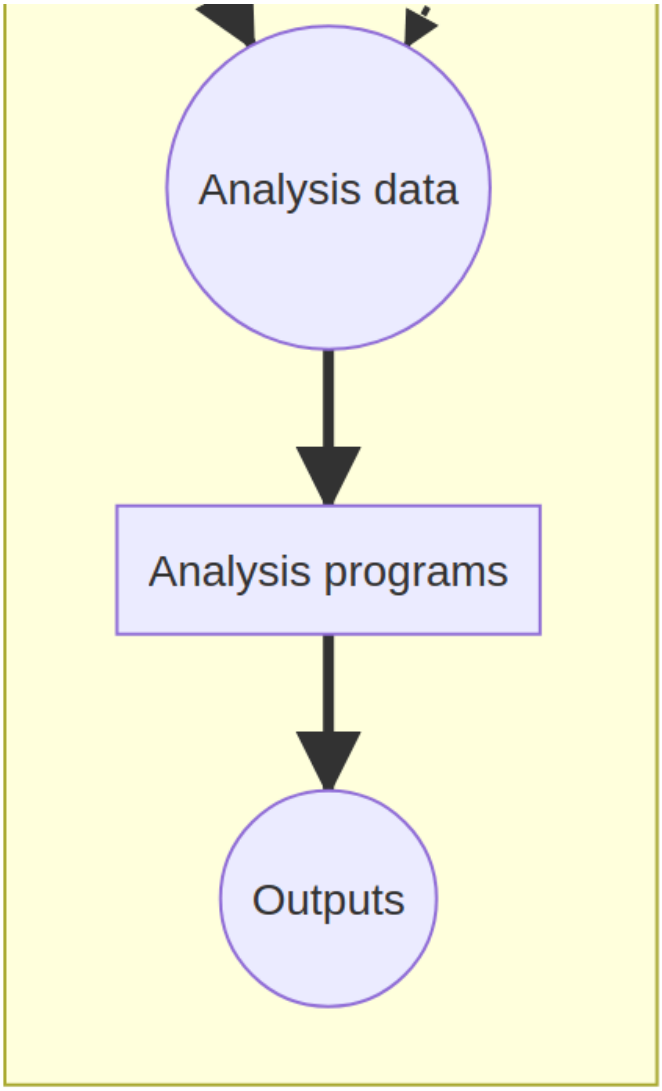
Details on data workflows

In a reproducible workflow, *instructions* to perform all of the above are provided. Most of the time, those instructions will be *computer code*, to be interpreted using statistical software, in some cases to be compiled into executables (C++, Fortran).

Sometimes, the data acquisition might also be coded - using software packages or scripts to download, or through computer-assisted surveys or experiments. But more often, acquiring data entails manual actions. For instance, secondary data access might be described in Data Availability Statements, whereas primary data acquisition might be described in survey documentation, field guides, and in experiment instructions.^[Note that when the authors generate or collect primary data, subsequent users of the same data are secondary data users, but replication might also involve re-executing the experiment, or collecting data anew with the same survey instruments.] Once analysis data is created, the analysis programs generate output that is then embedded in the article. Often, that output might be ready-made tables and figures. But sometimes authors will manually transcode output from log files into tables. Modern reproducible documents will capture such output and embed it into a document directly.

The following diagram illustrates the generic flow:


Source
Data citation
Dataflow
Input data
Cleaning programs
Auxiliary data
(created)



Real articles are often more complex. A simple example will be discussed next.

A simple example

At the Github repository [labordynamicsinstitute/simple-example](), we have prepared a very simple example of a data analysis workflow. It illustrates the concepts above.

> **Tip**
>
> Read the README before going on.

We focus on the information provided in the README first:

- `Source`: Census of Population and Housing, 2000

The `source` data is identified, and a `data citation` is provided as part of the README. The README notes that the data is considered public-use, but a login is required at ICPSR, which is the distributor of the data. The data acquisition, as described, is a manual process (because it requires a login). The data is also manually unzipped, and the final location described.

> Note: this could be scripted to make part of the data acquisition code-driven.

- `Cleaning programs`: `01_dataclean.do`

A `cleaning program` is identified and described.

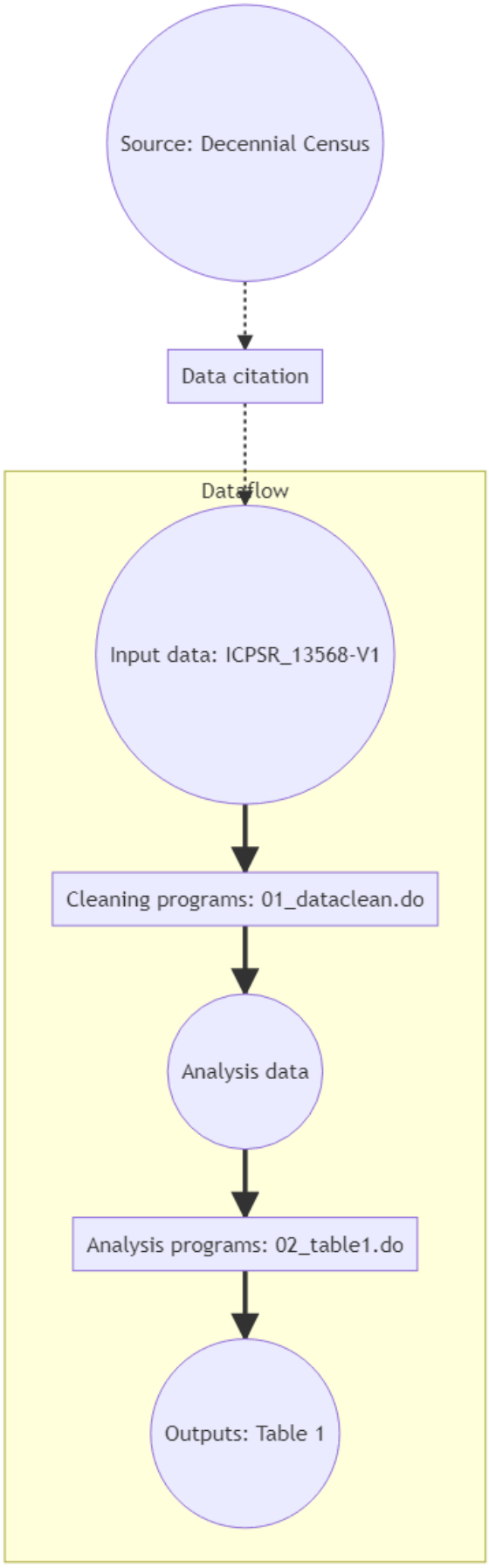
- `Analysis data`: is verbally described in the README (“clean merged dataset”)

> **Note**
>
> Could this be improved?

- `Analysis program`: `02_table1.do`
- `Outputs`: Table 1

An `analysis program` is also listed, and identified as creating `output` Table 1. An inspection of the “manuscript” shows that only a single table is included.

So a flow diagram for the simple example might be:

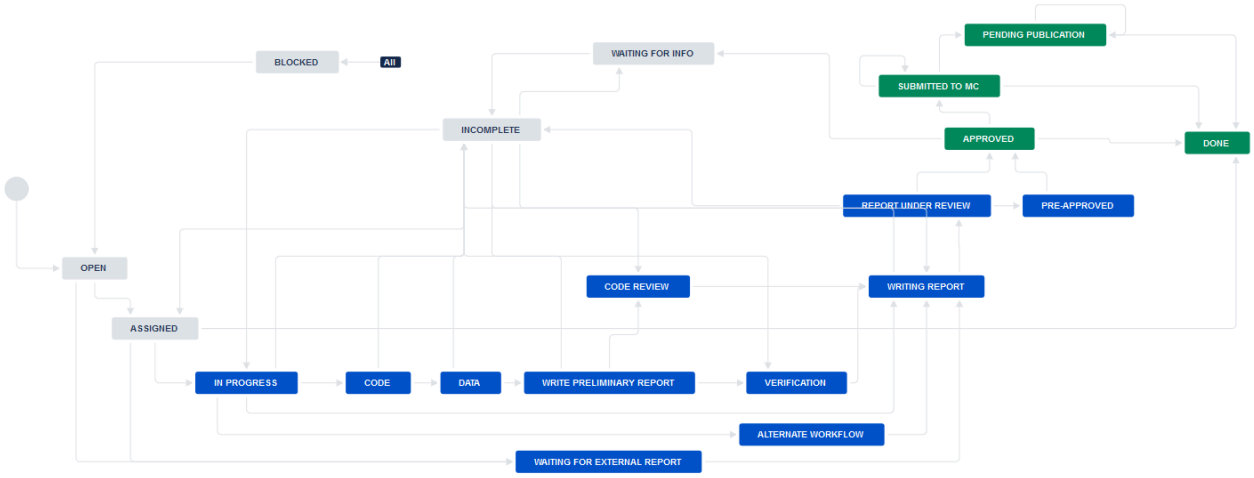

Source: Decennial Census
Data citation
Dataflow
Input data: ICPSR_13568-V1
Cleaning programs: 01_dataclean.do
Analysis data
Analysis programs: 02_table1.do
Outputs: Table 1

Assessing computational reproducibility

The basic workflow is simple:

- obtain the (pre-publication) manuscript and any materials
- analyze the README, and possibly appendixes and data sections, to identify all data sources, and all processing instructions
- obtain all code
- obtain all data
- execute all code
- compare the output with the manuscript tables, figures, and in-text

In practice, this quickly becomes more complicated, and requires a multitude of if-then instructions. At the LDI Lab, we have implemented the generic workflow within a system called Jira, which replicators use to go through the steps above.



At each step, additional information is requested, mostly through in-app fields and questionnaires, but also via some external tools if the Jira app did not provide the functionality. At the end, a report is created by the replicator, identifying all steps taken, which steps succeeded, and the final outcome.

We will first describe the prototypical report and its elements, and then walk through the detailed steps of the replicator's journey through the system. In our live training, replicators will also work on examples to "get the hang of it."

A guided walk through the Replication Report

In order to work through a replication report, you will need

- Access to computers
- Reviewed the Template REPORT

We have examples of various actual reports (slightly anonymized):

- Example 1
- Example 2

- Example 3 and its revision

Some high-level concepts

On data documentation

The Social Science Data Editor's page on Data documentation provides guidance:

- Identifying all data
- What is great / good / just-good-enough data documentation
- Citing data!

On code documentation

The Social Science Data Editor's page on Code and documentation provides guidance:

- What do we consider to be "code"
- Assessing the quality of the code documentation

To come

How to modify code for replicability/verification

What's in a replication report

The template used by the Lab can be found on Github here. The table of contents looks like this:

- SUMMARY *
- Data description *
- Data deposit *
- Stated Requirements
- Code description
- Data files provided
- Computing Environment of the Replicator
- Replication steps
- Findings
    - Missing Requirements
    - Data prep, tables, figures, and in-text numbers
- Classification

The sections marked with * are in "Part A" of the report. They lead to a preliminary report, which is a "dry" assessment of the completeness of the replication package. The remaining sections are in "Part B" of the report, and are filled out as a replicator attempts to run the code. In some cases, it may turn out that after describing the "Stated Requirements", we find that it is not

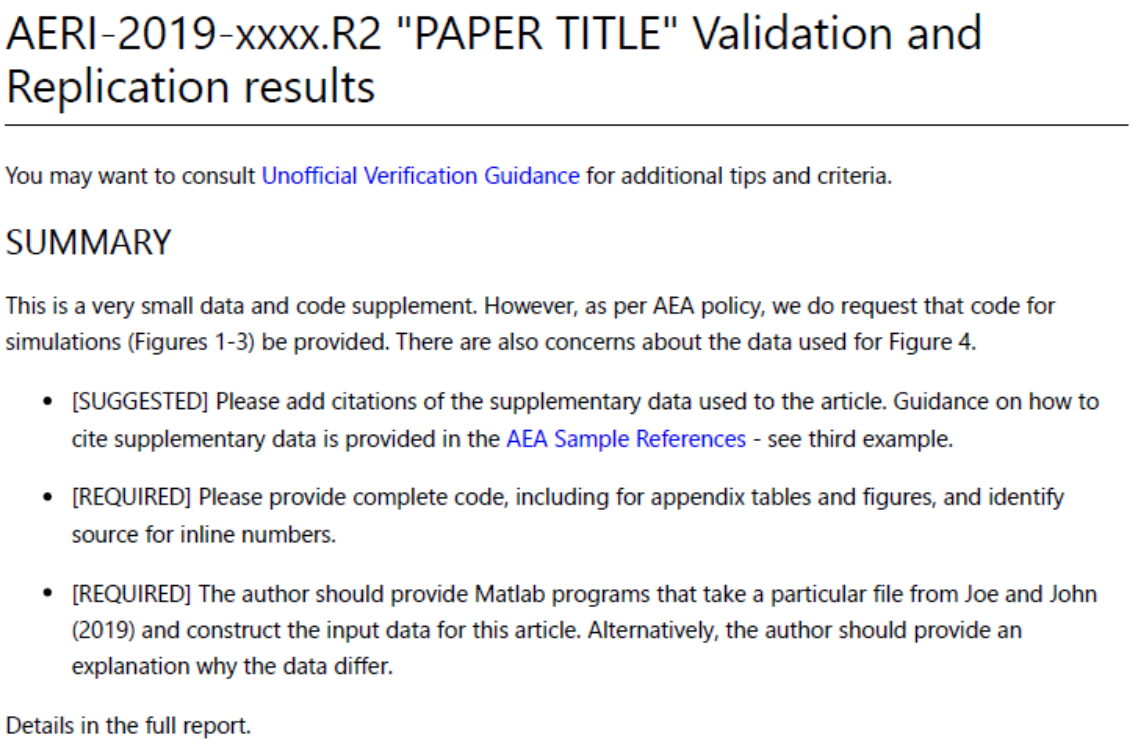
possible to run the code. Finally, “Findings” and “Classification” wrap up the report, once both Part A and B are completed, and involve having a look at what didn’t work, was not well documented, what figures or tables are not the same.

In most sections, when elements are missing, wrong, or do not work, we use a standardized set of action items to highlight this. This is present in your default template repository.

Report Part A

Summary

The SUMMARY is intended for a quick glance by journal editor and authors. It should be short and succinct, with a bulleted (unduplicated) list of action items for the authors, drawn from the rest of the report. It is the first thing editors and authors will see.



AERI-2019-xxxx.R2 "PAPER TITLE" Validation and Replication results

You may want to consult Unofficial Verification Guidance for additional tips and criteria.

SUMMARY

This is a very small data and code supplement. However, as per AEA policy, we do request that code for simulations (Figures 1-3) be provided. There are also concerns about the data used for Figure 4.

- [SUGGESTED] Please add citations of the supplementary data used to the article. Guidance on how to cite supplementary data is provided in the AEA Sample References - see third example.
- [REQUIRED] Please provide complete code, including for appendix tables and figures, and identify source for inline numbers.
- [REQUIRED] The author should provide Matlab programs that take a particular file from Joe and John (2019) and construct the input data for this article. Alternatively, the author should provide an explanation why the data differ.

Details in the full report.

Data description

The data description can require substantial time to complete. The replicator is asked to identify all `input` (original) data sources used by the authors. It sometimes is useful to create a working list (spreadsheet) and commit the list together with the report. The ACRE project has a useful template, and te template report has an example.

An essential part in writing the data description section is identifying the data used in the analysis. While the dataset used for the main analysis is often explained in the README or in the manuscript, the description of other datasets (e.g. datasets used in the appendix, introduction, or in a figure describing the study settings) are sometimes omitted in the provided documents.

Once the preparations above are completed, a summary should be written in the “Data Description” section.

What data need to be described?

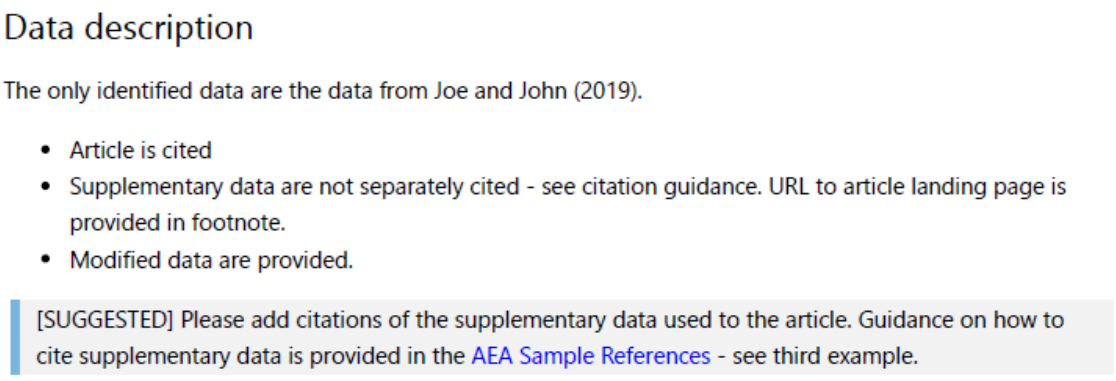
All "`input` (original) data sources" and "Analysis data files" should be listed.

- Data needs to be listed include:
  - Any data used to produce tables, figures, and in-text numbers that presents the estimated results, summary statistics, or any other numbers that are calculated from the data
  - Data used to create maps.
    - The data source of the geographical information is a source data if the map is created by the authors.
- Data need not be listed include:
  - Source data for the numbers or estimates directly quoted from other aritcles
  - Souce data for the parameters used for calibrations, unless they are estimated within the article.

Source data

For each data source, list

- presence or absence of source data (data files),
- presence or absence of codebook/information on the data, and summary statistics. Summary statistics and codebook may not be necessary if they are available for public use data. In all cases, if the author of the article points to an online location for such information, that is OK.
  - The information of the source location of the data should instruct the replicator how to access the source data.
  - A replicator should validate the provided description of the access information by visiting the link, downloading the dataset from the link, and compare the downloaded dataset with the provided dataset.
- whether the *data* is cited (see the section on data citations. Note that when authors cite data supplements, both the article and the data supplement should be cited - often, the latter is missing.



Data description

The only identified data are the data from Joe and John (2019).

- Article is cited
- Supplementary data are not separately cited - see citation guidance. URL to article landing page is provided in footnote.
- Modified data are provided.

> [SUGGESTED] Please add citations of the supplementary data used to the article. Guidance on how to cite supplementary data is provided in the AEA Sample References - see third example.

Analysis data files

Analysis data files are the data files from which output tables and figures are produced directly. If any analysis data files are provided and found, they are listed. Analysis data files are produced by code in the deposit from data sources. Not every deposit will have these, and in some cases, there may be ambiguity if a data source is not clearly defined. In some cases, replicators will identify surplus data - data not associated with any source and any program. Authors are then asked to clarify this information.

Metadata checks on deposit

Most replication packages received by the LDI Replication Lab will have been deposited in the AEA Data and Code Repository, but some may be on other trusted repositories (Dataverse, Zenodo, etc.).^[See the Social Science Data Editor website for a list of trusted repositories.] The replicators are asked to verify compliance with an AEA-specific list of required elements:

- ☐ JEL Classification (required)
- ☐ Manuscript Number (required)
- ☐ Subject Terms (highly recommended)
- ☐ Geographic coverage (highly recommended)
- ☐ Time period(s) (highly recommended)
- ☐ Collection date(s) (suggested)
- ☐ Universe (suggested)
- ☐ Data Type(s) (suggested)
- ☐ Data Source (suggested)
- ☐ Units of Observation (suggested)

If all required elements are provided, then the deposit passes. Many of the recommended elements are not applicable to all data deposits - for instance, a simulation has no “geographic coverage” or “collection date”, but a survey clearly does.

ICPSR data deposit

Requirements

- ☑ README is in TXT, MD, PDF format
- ☑ openICPSR deposit has no ZIP files
- ☑ Title conforms to guidance (starts with "Data and Code for:" or "Code for:", is properly capitalized)
- ☑ Authors (with affiliations) are listed in the same order as on the paper

Deposit Metadata

- ☑ JEL Classification (required)
- ☑ Manuscript Number (required)
- ☐ Subject Terms (highly recommended)
- ☐ Geographic coverage (highly recommended)
- ☐ Time period(s) (highly recommended)
- ☐ Collection date(s) (suggested)
- ☐ Universe (suggested)
- ☐ Data Type(s) (suggested)
- ☐ Data Source (suggested)
- ☐ Units of Observation (suggested)

> [NOTE] openICPSR metadata is sufficient.

> [SUGGESTED] We suggest you update the openICPSR metadata fields marked as (suggested), in order to improve findability of your data and code supplement.

For additional guidance, see https://aeadataeditor.github.io/aea-de-guidance/data-deposit-aea-guidance.html.

> **This completes Part A.**
>
> Note that no code needed to be run. Yet you will find that this can take considerable time. You may need to verify if links that authors provide actually lead to the relevant information.

Report Part B

Stated Requirements

The authors should have specified specific requirements in terms of software, computer hardware, runtime, add-on packages. The replicator should list them here. This is **different** from the *Computing Environment of the Replicator*, which you will describe later. If all requirements are listed, check the box "Requirements are complete". This section is important to assess the feasibility of the reproducibility attempt. A reproduction that requires "20,000 core compute hours", or that "runs for weeks", or that requires custom software that needs to be acquired, may not be feasible.

- ☐ No requirements specified
- ☐ Software Requirements specified as follows:
  - Software 1
  - Software 2
- ☐ Computational Requirements specified as follows:
  - Cluster size, etc.
- ☐ Time Requirements specified as follows:
  - Length of necessary computation (hours, weeks, etc.)
- ☐ Requirements are complete.

Code description

All deposits should have code. In line with the basic data flow, there should be both data cleaning or preparation code, as well as analysis code. The replicator will review the code (but will not run it yet).

- Identify programs that create "analysis files" ("data preparation code").
- Identify programs that create tables and figures.

From the README, the replicator should be able to identify code to create all **figures, tables, and any in-text numbers**. If not listed in the README, comments in the code should enable the replicator to find this. The replicator will create a list, mapping each of figure, table, and in-text number to a particular program and line number within the program. A template spreadsheet is provided. Note that the code description might already observe that setup programs are missing, but most missing code will be identified in the [findings][findings] section.

Data files provided

Normally, this simply lists the files that are part of the replication package in the deposit, and should be filled out automatically by scripts. However, in some cases, those scripts will fail. Then, it is the replicator handling Part B who needs to run the necessary scripts manually.

> **Note**
>
> At this point, you *still* have not run any code, but you have collected everything needed to discuss this case with the group.

Computing Environment of the Replicator

Just as the original authors have a particular computing environment that a replicator needs to know in order to properly implement a reproducibility attempt, the replicator's own attempt is important. This section should *not* describe the laptop the replicator uses to write the report =- that is irrelevant if we use the remote resources at the Lab, unless you use the laptop to actually run code! – but should provide as complete a list of details as possible describing the computer where the computational component of the reproducibility check was conducted. Some of these details can be found as follows:

- (Windows) by right-clicking on "My PC"
- (Mac) Apple-menu > "About this Mac"
- (Linux) see code in `tools/linux-system-info.sh`

Examples of computing environments used at the Lab are pre-printed in the template report, but can be amended or augmented.

- CCSS Cloud Windows Server AMD EPYC 7763 64-Core Processor 2.44 GHz
- BioHPC Linux server, Rocky Linux 9.0, AMD Opteron Processor 6380 / 16 cores/ 125 GB memory (adjust as necessary from output of linux-system-info.sh)
- WholeTale (describe the environment chosen)
- CodeOcean (describe the type of capsule chosen) Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz (16 cores), 128 GB Memory
- Bitbucket Pipelines, "Ubuntu 20.04.2 LTS", Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz, 8 cores, 9GB memory
- Codespaces, "Ubuntu 20.04.2 LTS", Intel(R) Xeon(R) Platinum 8272CL CPU @ 2.60GHz, 2 core, 4GB/ 4-core, 8GB/ 8-core, 16 GB/ 16-core, 64GB (choose appropriately)

The list should also list the software the replicator used, with the specific version used (even if the author did not list that information). Examples include:

- Stata/MP 16
- Matlab R2019a
- Intel Compiler 3.14152

but always check what version you are actually using on the server you are using. For instance, as of Oct 2023, the version of Stata installed on CCSS Cloud is Stata 18, but on BioHPC, the standard version is still Stata 16.

Replication steps

For every replication or reproducibility attempt, the list of steps a replicator undertakes is important to be listed. In principle, these steps should be specified in the README, but while the README contains instructions, this section should contain what you actually did. It should include details as to under what name the replicator saved a dataset downloaded from a website (if not the suggested name), or what minor edits were made to programs.

- DO NOT include trivial details (“I downloaded the code and saved on my Desktop”).
- DO describe actions that you did as per instructions here (“I added a config.do”) or as per the authors’ README (“Ran main.do as per the README”)
- DO describe any other actions you needed to do (“I had to make changes in multiple programs”). It is often helpful to provide additional detail, such as an excerpt of the error message, and the fix applied, allowing the author to read the report and fix the issues. Remember that the author will not have access to our Bitbucket repository, nor to the log files that might contain the errors.

The description should allow the Data Editor and the authors to understand that everything was done as instructed. Deviations need to be described with enough detail that somebody else can reproduce the deviation!

Replication steps

1. Attempted to run RBCModel.mod, recieved an error:

```
(ERROR LINE 15: Undefined function or variables 'dates')
```

1. Added the config.do which generates system information and creates log files. Included packages mentioned in CONSTRUCT_data.do to the config.
2. Downloaded data mentioned in README from FRBSF and placed into same folder as CONSTRUCT_data.do
3. Ran CONSTRUCT_data.do (after running config.do in the same DATA_CONSTRUCTION folder)
4. Ran ANALYZE_data.do
5. Ran Matlab_plots.m from Section 1 of the README
6. Ran replicateSection3.m
7. Ran replicateSection5.m
8. Ran replicateSection4.m

> “[REQUIRED] Please provide debugged code, addressing the issue identified in this report. ”

Findings

Once everything is put in place, the replicator can report on findings, both positive and negative. This should include enough detail to allow a reader - a Data Editor and the authors - to understand what went wrong when something went wrong. For each **Data Preparation Code, Figure, Table, and any in-text numbers**, the section should provide information on success or failure to reproduce (the previously filled out code-check.xlsx can be re-used to drive the list). When errors happen, the replicator’s description should be as precise as possible. For differences in figures, the replicator should provide both a screenshot of what the manuscript contains, as well as the figure produced by the code you ran, with differences highlighted. For differences in numbers (in tables or in-text), the replicator should list both the number as reported in the manuscript, as well as the number replicated.

Tables

- Table 1: unable to reproduce, even after examining our version of the log file `ANALYZE_data.log` for the output
- Table 2: unable to reproduce, even after examining our version of the log file `ANALYZE_data.log` for the output
- Table 3: not produced by the code
- Table 4: looks the same, `calibration_targets.xls`
- Table 5: looks the same, `calibrated_parameters.xls`
- Table 6: very slight numerical discrepancies: Row 1 col 2 is 1.61 in paper vs 1.62 in replication; Row 4 col 5 is 0.86 in paper vs 0.87 in replication

Missing Requirements

If it turns out that some requirements were not stated or are incomplete (software, packages, operating system), the replicator should list the *complete* list of requirements here. Some of this maybe immediately obvious (e.g., there is no mention at all of what kind of computer you need), in others, you will need to run the programs to find out that things are missing. This is usually amended as the reproducibility attempt progresses.

Classification

The replication report template used by the LDI Replication Lab uses a simplified scheme to summarize the reproducibility of the materials:

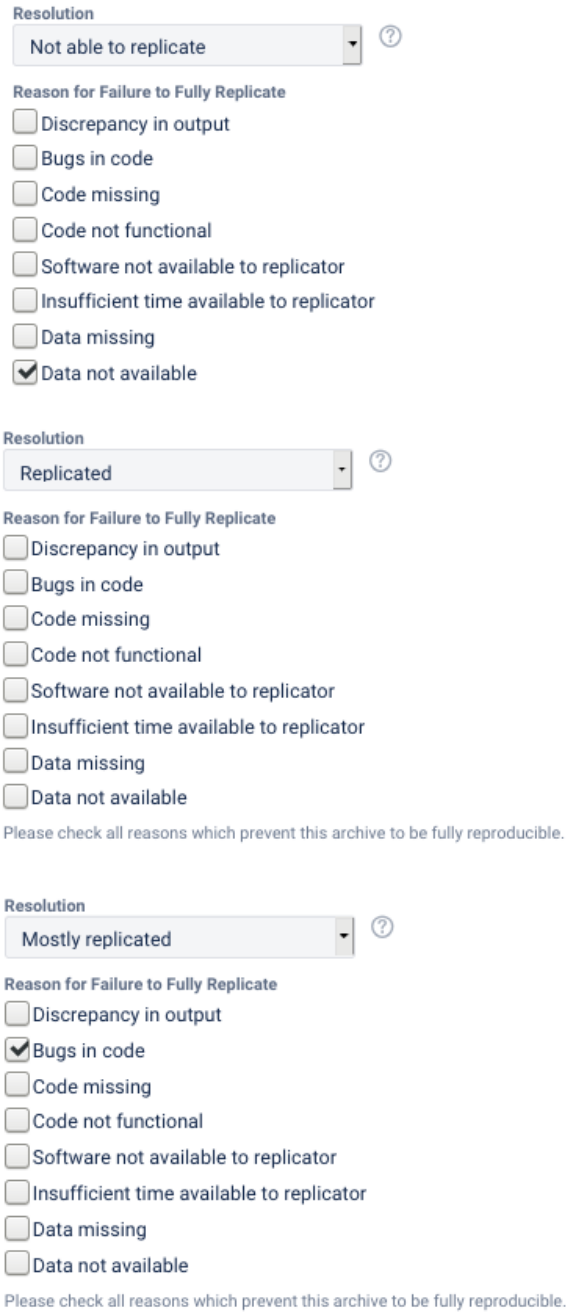
- Full replication can include a small number of apparently insignificant changes in the numbers in the table. Full replication also applies when changes to the programs needed to be made, but were successfully implemented.
- Partial replication means that a significant number (>25%) of programs and/or numbers are different. Note that if any data is confidential and not available, then at best a partial replication can be achieved.
- Failure to replicate implies that only a small number of programs ran successfully, or only a small number of numbers were successfully generated (<25%). Most replication packages that rely on confidential data will also be in this category.
- ☐ full replication
- ☐ full replication with minor issues
- ☐ partial replication (see above)
- ☐ not able to replicate most or all of the results

The Lab does not (yet) use a more refined reproducibility score, such as the one developed as part of the BITSS ACRE Project. The emphasis is on a summary measure, combined with detailed reasons why full reproducibility is not achieved.

The reasons for not being able to fully reproduce the materials can be multiple, and should be noted in the report (they are captured through a multiple-choice field in the LDI Lab's JIRA system):

- ☐ `Discrepancy in output` (either figures or numbers in tables or text differ)
- ☐ `Bugs in code` that were fixable by the replicator (but should be fixed in the final deposit)
- ☐ `Code missing`, in particular if it prevented the replicator from completing the reproducibility check
- ☐ `Code not functional` is more severe than a simple bug: it prevented the replicator from completing the reproducibility check
- ☐ `Software not available to replicator` may happen for a variety of reasons, but in particular (a) when the software is commercial, and the replicator does not have access to a licensed copy, or (b) the software is open-source, but a specific version required to conduct the reproducibility check is not available.
- ☐ `Insufficient time available to replicator` is applicable when (a) running the code would take weeks or more (b) running the code might take less time if sufficient compute resources were to be brought to bear, but no such resources can be accessed in a timely fashion (c) the replication package is very complex, and following all (manual and scripted) steps would take too long.
- ☐ `Data missing` is marked when data *should* be available, but was erroneously not provided, or is not accessible via the procedures described in the replication package
- ☐ `Data not available` is marked when data requires additional access steps, for instance purchase or application procedure.

Note that absence of full replication is not necessarily a reason to reject the replication package. The AEA regularly accepts replication packages that are not reproducible by the AEA Data Editor because the “data are not available” or because there is “insufficient time”, as long as the replication package could plausibly be reproduced by somebody with extra time or with access to the data. The determination is made by the AEA Data Editor, based on the report.



AEA Jira workflow

For pre-publication verification, we use a Jira-based workflow to guide the replicator through the process of filling out the report (called `REPLICATION.md`).

> **Note**
>
> The link to JIRA is https://aeadataeditors.atlassian.net/jira (requires login).

New

> **Warning**
>
> For most Jira issues created after 2024-01-30, we use a new workflow. In this, the report is initially split into `REPLICATION-PartA.md` and `REPLICATION-PartB.md`, and subtasks are created for preparing parts A and B. Once both parts are ready, they are merged back into `REPLICATION.md`.

Scope

Your supervisor will assign you to this workflow. This workflow covers code and data, even when data may not be accessible. Supervisor, see other document for details.

- This workflow **DOES NOT** cover simple metadata assessment of openICPSR deposits, for instance for AEA Papers and Proceedings deposits. See Instructions for Papers and Proceedings.

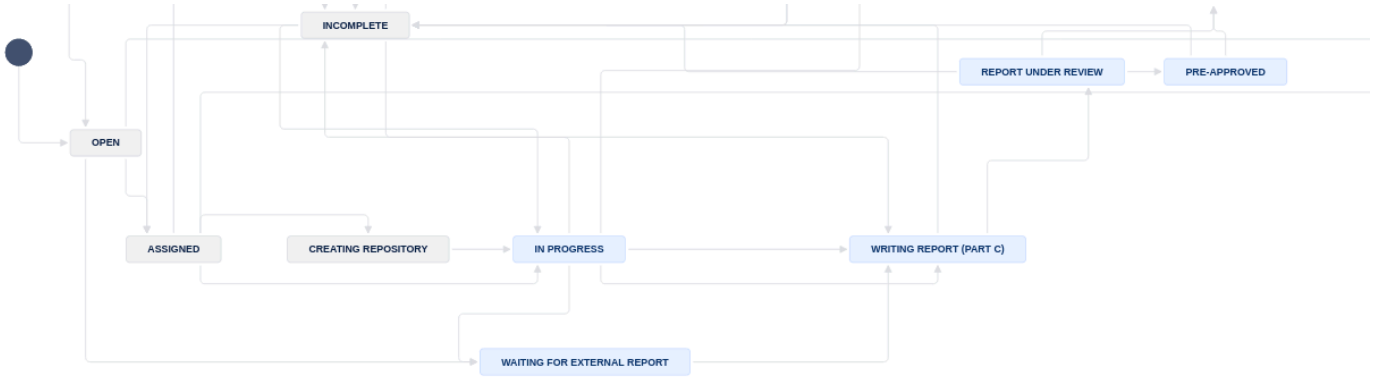
Overview

Prior to assignment

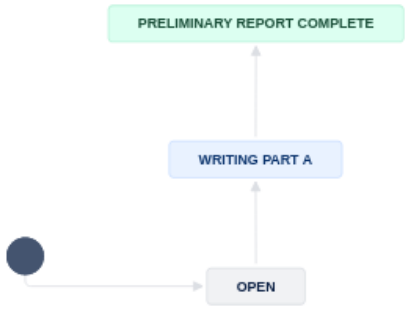
AEA manuscripts are checked for compliance with the AEA Data and Code Availability Policy after having been `conditionally accepted` by the journal's Editor-in-Chief (EIC) or Co-Editor (which we'll call "the editor" collectively).[1] Once authors have received the editor's "Conditional Acceptance" email, they send their manuscript materials to the AEA Editorial Office, together with the AEA Data and Code Availability Form (DCAF). It is the **DCAF** which identifies where the authors have initiated a deposit of their replication package. Usually, this is at the AEA Data and Code Repository @ openICPSR, but it can be at any trusted repository.

Once the AEA Editorial Office has verified that everything is complete, they assign the manuscript to the LDI Replication Lab. This assignments happens in the manuscript management system "Scholar One", and triggers an email. This email enters our JIRA system system, and becomes an "open JIRA issue". That's where we start.

High-level view of flow through our process



An `Open` issue is first evaluated by the a supervisor (the AEA Data Editor or their assistant). The supervisor takes into account the complexity of the case, the known software or data acquisition skills of the replicators, the urgency of the case (for instance, if one of the authors is soon up for tenure), etc. Some of this is already coded into JIRA fields.

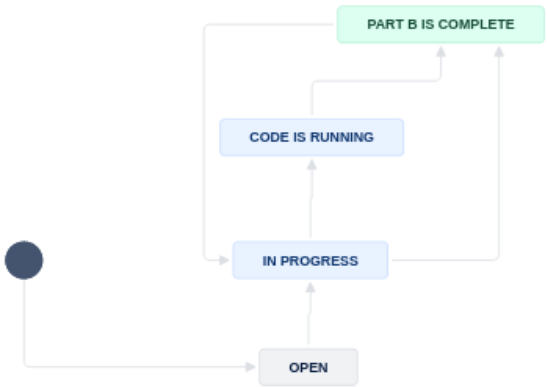


Once `Assigned` a new case, the replicator will create a new git repository, if a brand new case, or reference an existing repository (if the issue is a revision) in order to move to `In Progress`. Each manuscript is associated with a single git repository, which tracks the evolution of the author's code, and of our reports and attempts to run it, over time.

The replicator then copies the author's code (but not the data!) into the git repository, usually by using automated scripts, then adds the manuscript and DCAF as provided by the AEA Editorial Office. The first task is to make an extensive assessment of the replication package in the repository, what we call a Preliminary Report. This identifies the three key elements of a reproducibility attempt:"

- is the guiding document to the replication deposit (the README) sufficiently complete and clear to undertake a complete assessment? The replicator is the first reader of this, but the ultimate reader is a future researcher who wants to use the data and code.
- can we access all the data? Is there additional data that needs to be obtained, and how complicated is that? (Some of this may have been signalled by the authors via the DCAF and coded by the supervisor.)
- can we run the code? Do we have access to the software, to the computing hardware to run the code? If not, can we acquire such access in a reasonable time frame?

Once the Preliminary Report is complete, the replicator may discuss this at one of the bi-weekly meetings with the AEA Data Editor, if there are questions.



If a decision is made to attempt a computational reproduction of the paper, the replicator then proceeds to the next step, running code. This involves setting up the computing environment, the project environment, and running the code. Unfortunately, this almost always involves some debugging, and replicators get really good at that!

If there is too much debugging, or the debugging does not work, we may return it to the authors at this point, for correction of the identified issues, and resubmission. Our report tries to provide as much detail about what the replicator attempted and didn't work, to assist the authors, but is not a full "code consultation", as that is not the role of the journal.

If the debugging was successful (or not necessary), the replicator will, at this time, have a collection of tables and figures produced by the code. It is time to compile all this into a report. Tables and figures are compared to the manuscript, to assess for completeness and accuracy. We expect that all results are exactly reproduced, down to the last decimal, unless the README clearly states that this should not be expected (and explains why).

Once the report is completed, it is sent back to the authors via the ScholarOne interface, to the AEA Editorial Office. The report will contain the AEA Data Editor's decision, which can be:

- `Accept` - no changes need to be made anywhere
- `Accept with changes` - there are minor changes that need to be made to the manuscript and/or the deposit, but these are not substantive, and do not require re-running code. Manuscript changes are implemented at the copy-editing stage.
- `Conditional accept` - there are substantive changes that need to be made to the manuscript and/or the deposit, and these changes require a full review by the Data Editor, including possibly re-running code.
- `Revise-and-resubmit` - when issues are found that put into doubt the results in the manuscript, the manuscript may need to be reviewed by the editor, or the referees, again. This is never chosen without a prior discussion with the editor, and extremely rare. As of 2024, none such cases have lead to a terminal rejection of the manuscript.

Details

The following table illustrates the flow and transitions. The `transition` field identifies the button that will appear in the interface that needs to be clicked in order to progress an issue from the `From` state to the `To` state. The `Condition` field identifies which form field needs to be filled out in order to be able to make the transition. `Blocked` is always an option, and leads to a "waiting state" until a resolution can be found.

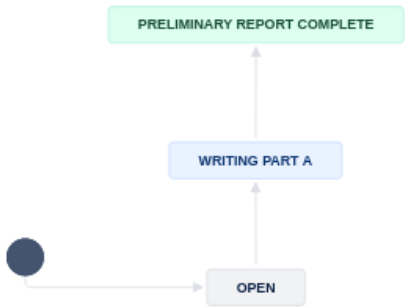
> **Warning**
>
> We regularly tweak the process to respond to needs and changes. The following detailed description may lag behind those tweaks!

Workflow for Main Task

From	Transition	→ To	Condition
Assigned	Create new repository	→ Creating repository	Issue is not a `Revision`
	Bypass repository creation	→ In Progress	Issue is a `Revision` , `Git working location` has been filled out
Creating repository	Start Task	→ In Progress	`Git working location` has been filled out
In Progress	Assign to external replicator	→ Waiting for external report	`External Replication` = `Yes`
In Progress	Write report	→ Writing Report (Part C)	Subtask Part A= `Preliminary Report Complete`, Subtask Part B = `Part B is complete`
Writing Report (Part C)	Submit for review	→ Report Under Review	`Report URL` is not empty.

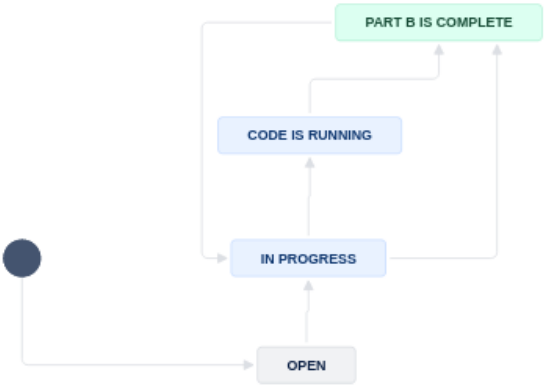
At this point, there will be two subtasks: `Prepare Part A` and `Run Part B`. The following transitions are available for each of these subtasks, unless the issue has been assigned to an external replicator.

Workflow for Part A Subtask



From	Transition	→ To	Condition
Open	Writing preliminary report	→ Writing Part A	
Writing Part A	Finished Part A	→ Preliminary Report Complete	`Part A` has been filled out

Workflow for Part B Subtask

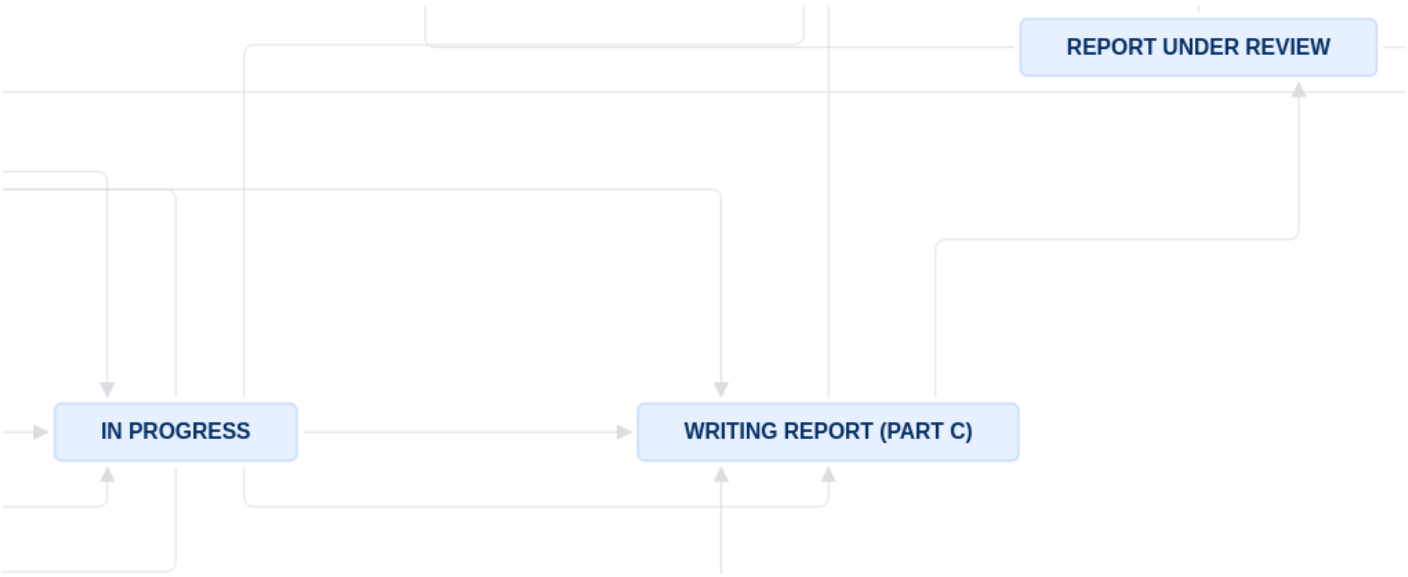


From	Transition	→ To	Condition
Open	Prepare to run code	→ In Progress	Working area has been prepared
In Progress	Start running code	→ Code is Running	Some data is available
In Progress	No code can be run	→ Part B is complete	No data is available
Code is Running	Code is done running	→ Part B is complete	

It is possible that a Pre-approver or an Administrator move the issue back to `In Progress` when additional debugging or code runs are necessary.

Once both parts are complete, the issue can be merged into `REPLICATION.md` and the final report compiled. This is also true once the external report has been received.

Continuation of Workflow for Main Task



From	Transition	→ To	Condition
In Progress	Write report	→ Writing Report (Part C)	`Report URL` is not empty
Waiting for external report	Write report	→ Writing Report (Part C)	`Report URL` is not empty
Writing Report (Part C)	Submit for review	→ Report Under Review	`Report URL` is not empty.
Multiple	Need information	→ Incomplete	when information is missing
Incomplete	Restart task	→ In Progress	
Incomplete	Prepare report	→ Writing Report (Part C)	

Other Workflows

The following are only relevant for “Approvers”, “Pre-Approvers” (if you have not been told you are a “(Pre-)Approver”, you are not.)

From	Transition	→ To	Condition
Under Review	Approve	→ Approved	Can only be done by **approvers**.
Pre-approved	Approve	→ Approved	`MCRecommendationV2` is filled out Can only be done by **approvers**.
Under Review	Pre-Approve	→ Pre-Approved	Can only be done by **pre-approvers**.

The following are only relevant for “Publishers” (if you have not been told you are a “Publisher”, you are not.)

From	Transition	→ To	Condition
Approved	Submit to MC	→ Submitted to MC	`MCRecommendationV2` is filled out
Submitted to MC	Wait for response on ICPSR	→ Pending openICPSR changes	`MCRecommendationV2` is `Accepted with changes` and notes to be added on openICPSR
Submitted to MC	Prepare for publication	→ Pending Publication	`openICPSRVersion` is filled out, issue is NOT an `R&R`
Pending openICPSR changes	Prepare for publication	–> Pending Publication	`openICPSRVersion` is filled out, Changes have been satisfied on openICPSR
Pending Publication	Publish	→ Published	`openICPSRDOI` is set

The following are relevant for a few other roles (usually not accessible to replicators):

From	Transition	→ To	Condition
Open	Assign	→ Assigned	Can only be done by **assigners**.
Assigned	Fast Track	→ Writing Report (Part C)	
.	Incomplete	→ Incomplete	n.a.
Approved	Done	→ Done	n.a.
Blocked	Reopen	→ Open	n.a.

Notes

- In the **Issue form**, please also fill out other fields, as noted.
- At any point, you can move the issue to `Incomplete`: more information/action is required before you can proceed. You should also notify us of the situation ASAP
- When committing, you can use Smart Commits, e.g.

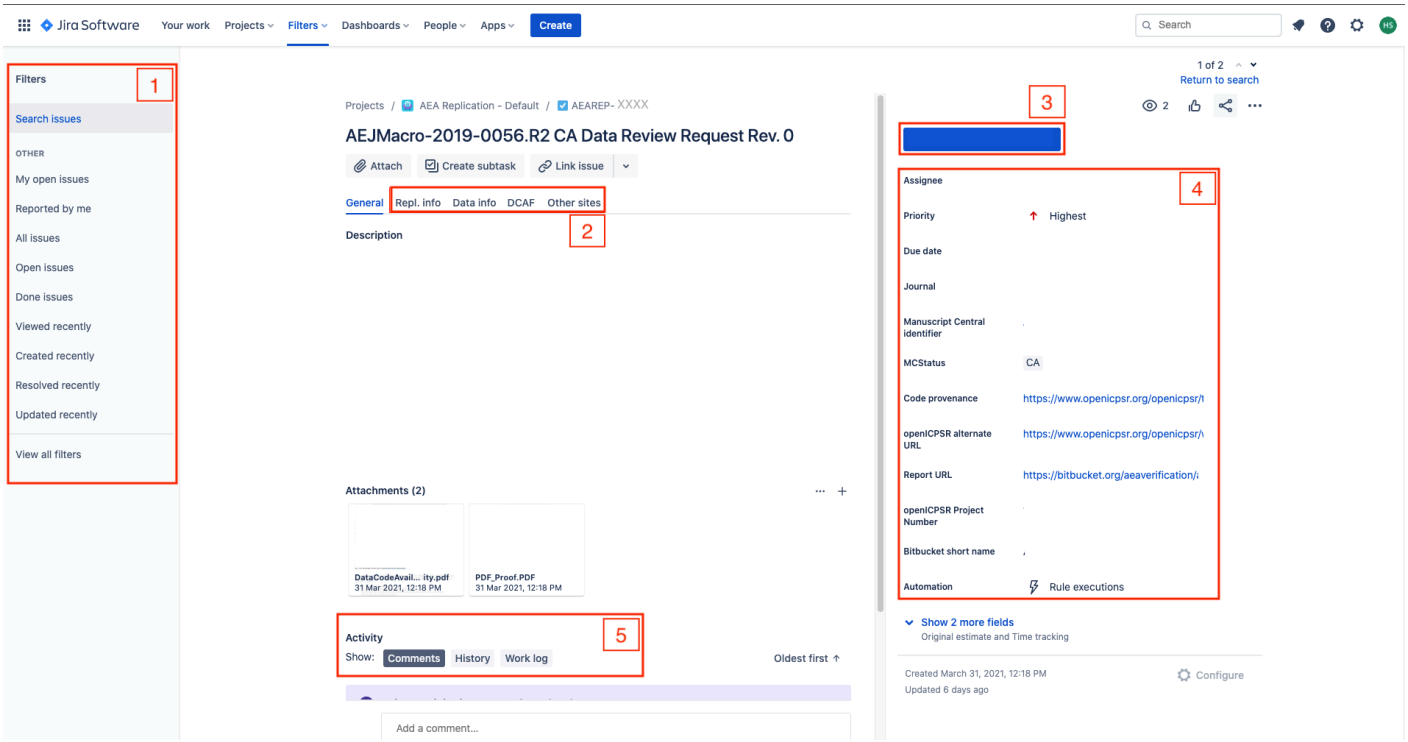
> AEAREP-1234 #comment corrected indent issue

- Use JIRA comments to communicate with your supervisor as issues arise, including code that takes a long time to run.

Details

Additional details for each of the key stages are provided here. Below is a screenshot of a Jira ticket. Some things to note:

- The blue `In Progress` box in the upper right - area 3 of the screen - is how you “advance” the Jira ticket. When you are first assigned a replication, this box will say `Open`.
- The tall grey bar on the left side - area 1 of the screen - contains several handy links that you will use throughout the process.
    - Sometimes this box is not visible. To make it visible, edit the URL for the Jira ticket so that there are no characters after the ticket number (e.g. AEAREP-123). You may have to refresh the page after doing so.



[1] The American Economic Journal: Applied Economics (AEJ:AE) is an exception, because they regularly assign us papers during the “revise-and-resubmit” phase, prior to final acceptance, for a preliminary or early check.

Assigned

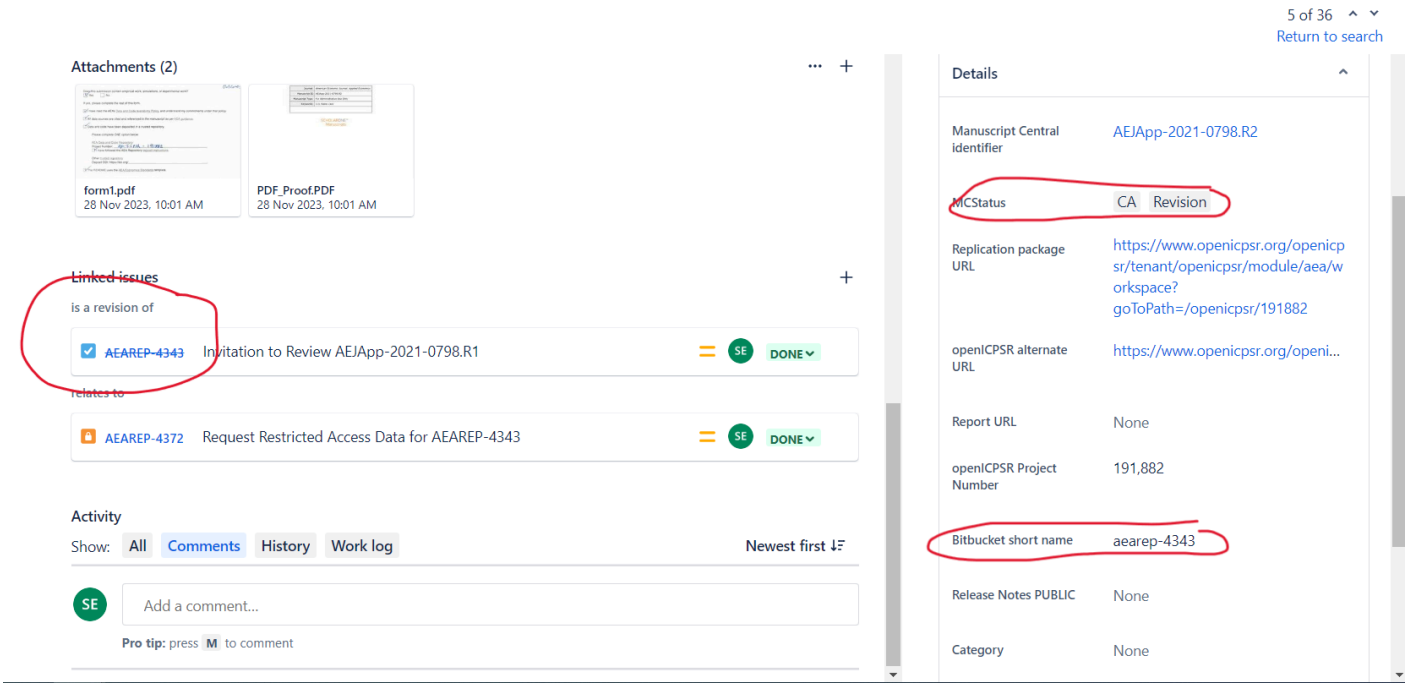
When you are first assigned an issue, it will in the `Assigned` status.

> **Note**
>
> The link to JIRA is https://aeadataeditors.atlassian.net/jira (requires login).

Identifying whether the issue is a revision or not

> **Warning**
>
> Is the current Jira issue an **original report** (first time we see the manuscript) or **is it a revision** (we’ve seen the manuscript before)?

Signs that this case is a revision



- ☐ Check the `MCStatus` field:
  - If it says "`RR`" or "`CA`", then it is an "original report" - proceed to the next step (**`Creating repository`**).
  - If it says "`CA` `Revision`", then it is … a revision!
    - Follow the instructions at "Revision reports after author resubmission".
    - In particular, **do NOT create a new repository** - you will re-use the previous repository. Enter the previous repository "stub" into the `Bitbucket short name` field.
    - In particular, **do NOT create "update" or "new" directories** The current state of the repository should always correspond to the author's structure. Overwrite files, delete files. The previous state is preserved in Git. This will also tell you what files have changed.
    - When running a second replication on the same archive, please be sure to have the committed "REPLICATION.md" be accurate when you commit it - do not let it contain holdover data from a previous replication attempt, as this can lead to confusion.
    - Once you have entered the previous repository "stub" into the `Bitbucket short name` field, you can proceed to **`In Progress`**.

> **Warning**
>
> If a field on Jira is already filled out, **do not edit it**. The only exception is if software is missing from the `Software used` field.

If this is not a revision

The next thing you must do, if this is **not a revision** is to either create a new repository (**`Create repository`**) in order to move to `In Progress`.

- This lets us know that you have started working on replication.

If this is a “split” case

It will happen that the usual order (first Part A, then Part B) is not followed. This may happen if the reproducibility check was done automatically, by a third party, or for various other reasons.

- One person will be assigned the overall responsibility for the case. This is the person the case is **Assigned** to, and they take responsibility for tracking Parts A, B, and C, all the way through to submission for review.
- Usually (but not always), that same person will be assigned Part A.
- If Part B is assigned to a different person, then often the FIRST person to touch the case will have (`created a repository`).

> ⚠ **Warning**
>
> Whoever creates the repository first should leave a note in the comments, and be sure to fill out the `Bitbucket short name` field on Jira, so that the second person can find the repository and use it for their work.

- The person who is assigned Part B, if not the primary assignee, **should ALWAYS work in a branch** of the repository (typically called `part-b`) and should not touch any of the documents usually used by Part A. See notes in **Preparing to Run Code**.

Creating a repository and running pipeline

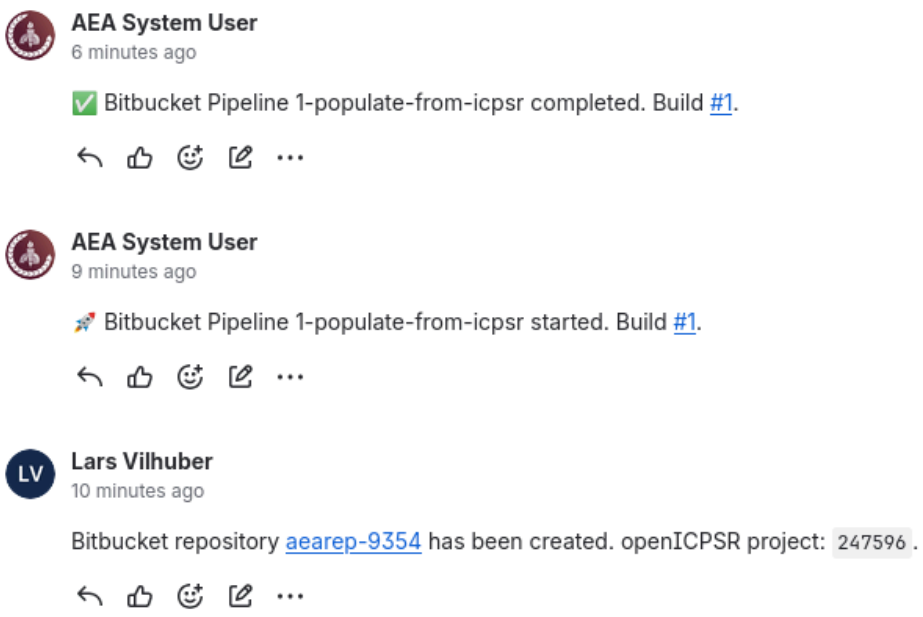
Warning

In some instances, somebody else has already created a repository. Always check first if the `Bitbucket short name` is already filled out. If it is, **do not** run the pipeline scripts again. This can overwrite or destroy changes already made by someone working on the case.

If the `Bitbucket short name` is filled out, skip this section and go to Collecting information!

What this looks like in Jira Click to hide

If the scripts have executed, then the Jira ticket will have one or more of these entries:



Advanced users Click to hide

You can also use the `editor-scripts` toolbox to do this from the command line, see `aeagit-create`.

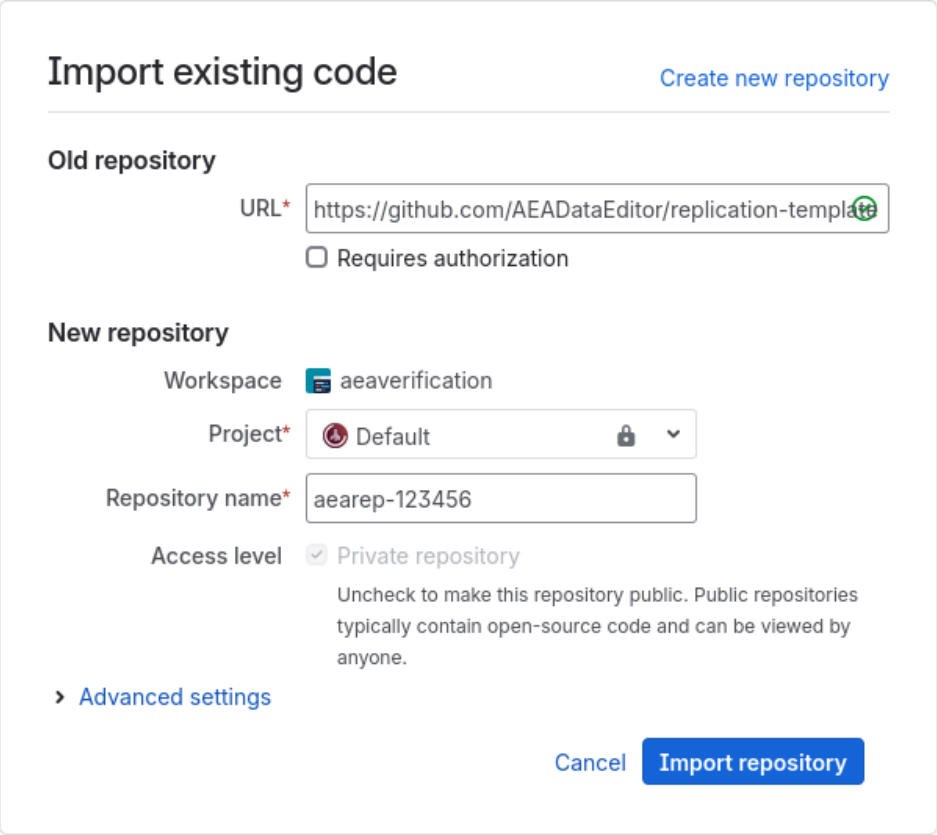
Creating a repository

- ☐ start by creating a repository using the import method
  - copy-paste from this URL to the URL field (this is also available in the Jira dropdown “Shortcuts”)

```
https://github.com/AEADataEditor/replication-template
```

  - the repository name should be the name of the JIRA issue, in **lower case** (e.g., `aearep-123`)
  - Be sure that `aeaverification` is always the “owner” of the report on Bitbucket.

- The Project should be the `DEFAULT` project, unless you are explicitly asked to create it in a different one (currently only “JEP” and “JEL”)
- Keep the other settings (in particular, keep this a **private** repository).
- Click `Import Repository`
- Keep this tab open!



- We have now created a Bitbucket repo named something like `aearep-123` that has been populated with the latest version of the LDI replication template documents!

Ingesting author materials

We will now ingest the authors’ materials, and run a few statistics. Typically, the materials will be on a (private) openICPSR repository. Sometimes, the materials will be at Dataverse, Zenodo, or elsewhere.

- If at openICPSR, the fields `Replication package URL`, `openICPSR alternate URL`, and `openICPSR Project Number` will be filled.
- If at Zenodo or Dataverse, the `Replication package URL` will have the DOI of the replication package, `openICPSR alternate URL` and `openICPSR Project Number` will be empty.

Note

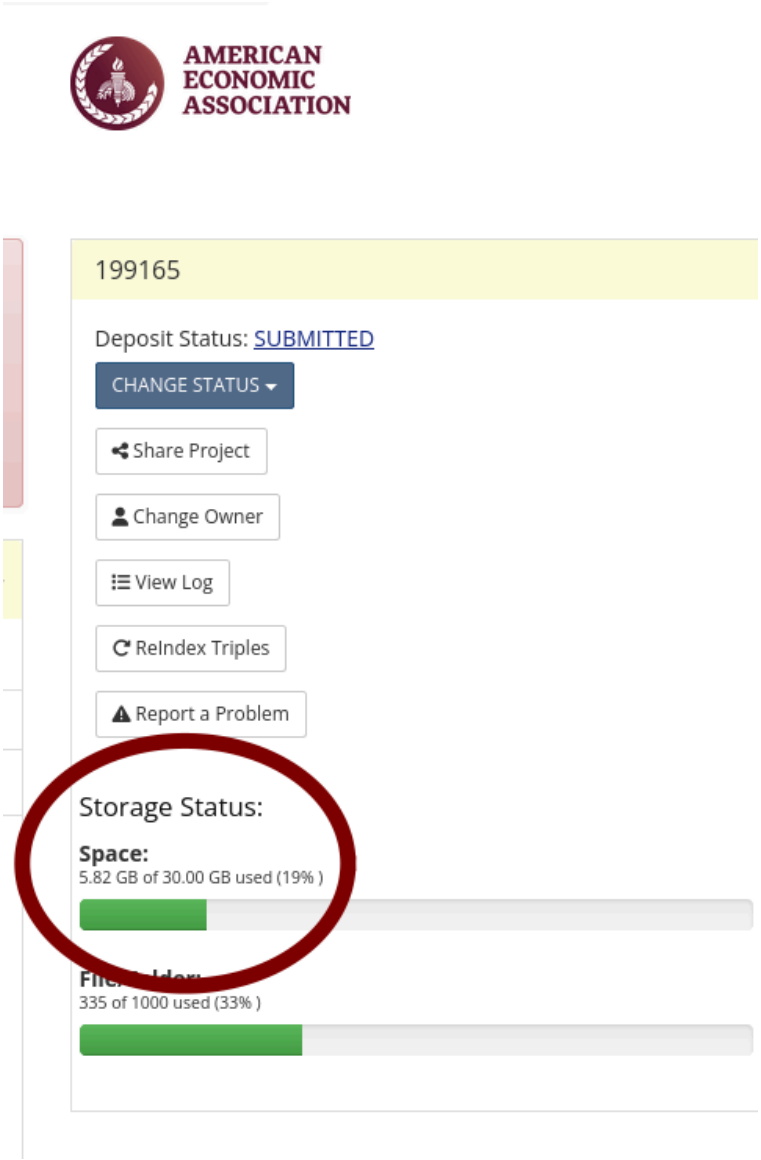
⚠️ This currently works reliably only for openICPSR. This documentation will be updated when it works for Dataverse and Zenodo as well. In the meantime, follow Manual Steps taking into account repository-specific guidelines.

Inspect the deposit

First, click on the `openICPSR alternate URL` URL (or `Replication package URL` if it contains a DOI and the other fields are empty). Inspect the deposit. The information may be in different locations at other repositories.

openICPSR | **Zenodo**

You will see the size of the deposit on the right:



Note

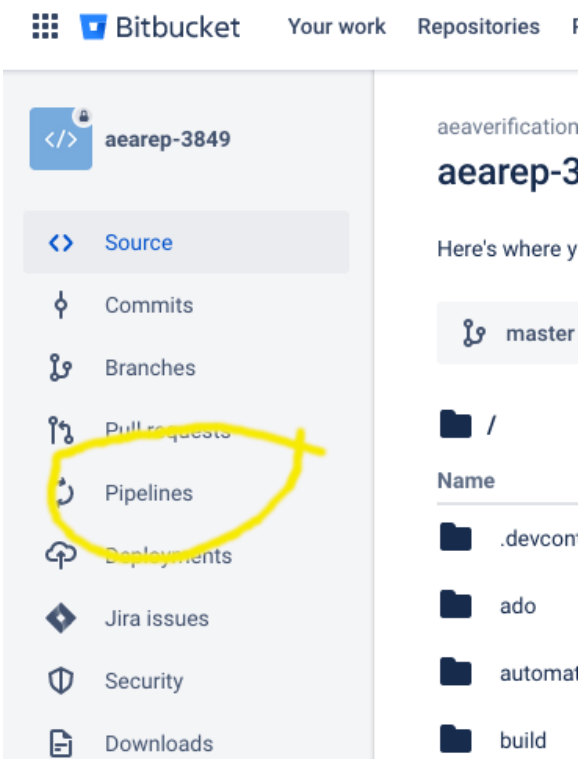
Make a note of the size of the deposit! Bitbucket pipelines have a limit of 1GB for technical reasons, but the size you see is not the size of the ZIPed download, it is the size before compression. The Zipped size is hard to predict.

Running the pipeline

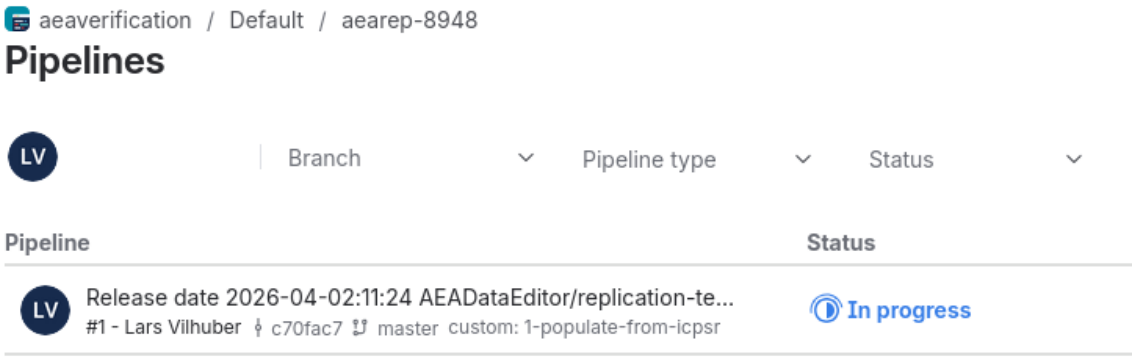
You will now run what is called a `*Bitbucket Pipeline*`. Similar tools on other sites might be called `Continuous integration`, `Github Actions`, etc. If you have encountered these before, this will not be news for you, but it isn't hard even

when this new.

- First, in the repository you just created, navigate to the `Pipelines` tab



- Verify that somebody else is not already running a pipeline!



If yes … Click to hide

Skip this step, or wait until the pipeline has completed. Do not re-run the same pipeline twice as it can cause problems. Contact the person (via the Jira comments) who is running the pipeline.

- Because this is new, you will see the “Run initial pipeline” page. Click on `Run initial pipeline`.

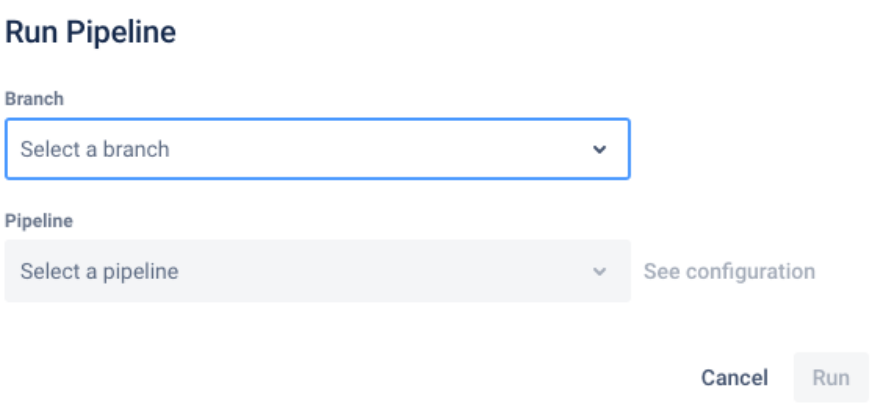
Build, test and deploy with Pipelines

Easy to set up Integrated CI/CD for Bitbucket Cloud that helps you automate your code from test to production in the Cloud or using your own infrastructure.

Looks like you already have a bitbucket-pipelines.yml file in this repository

Review pipeline | Run initial pipeline

- You will now need to select a “pipeline” to run.

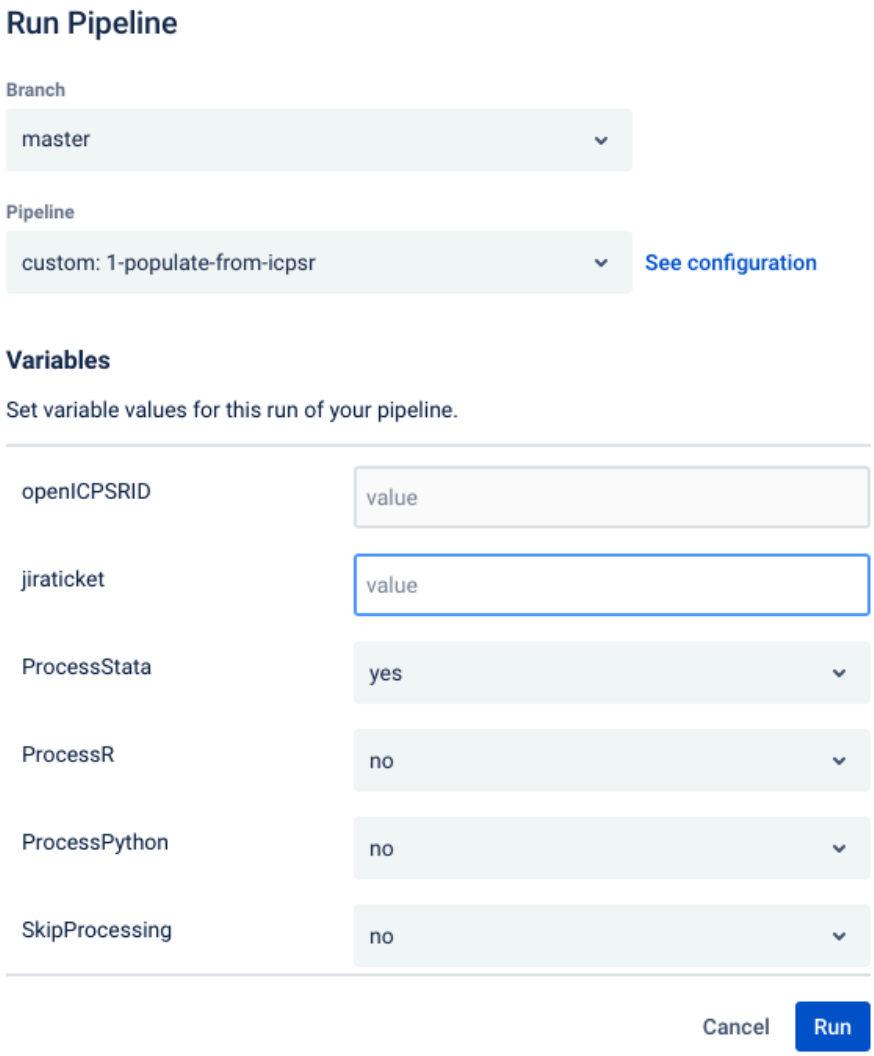


Note

This is where the information about the size of the deposit matters! Choose the option that best matches the size of the deposit.

If the deposit is less than 3 GB… Click to hide

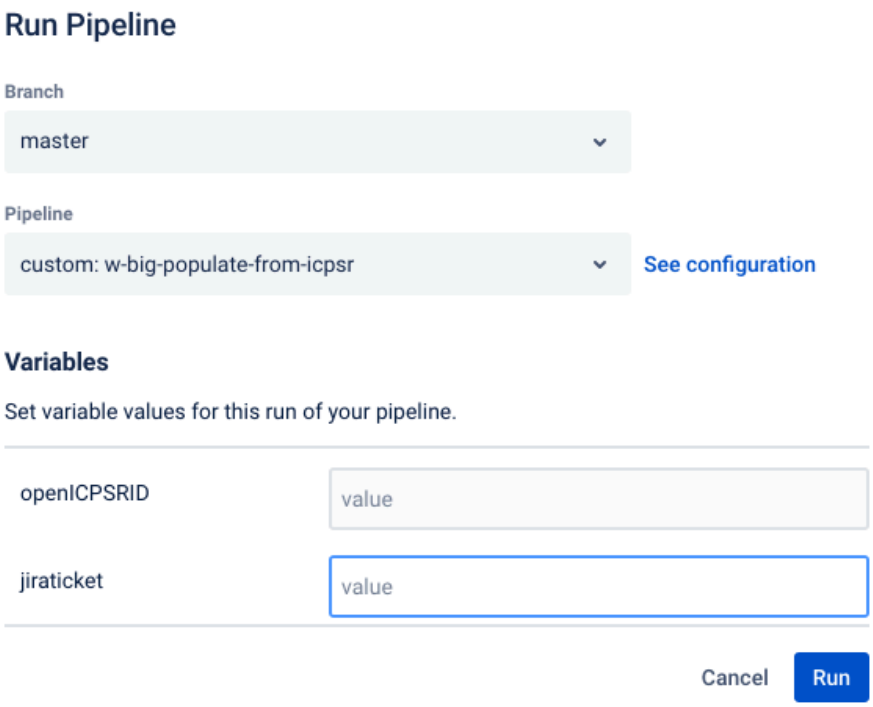
- Choose "`1-populate from ICPSR`" (might change in the future), and fill in the ID for the relevant source of the replication package (here: openICPSR ID = `123456`), and hit `Run`.



However, closely monitor the pipeline output - it might still fail because the resulting ZIP file is too big. See ZIP file is too big below for how to diagnose this.

If the deposit is more than 3 GB… Click to hide

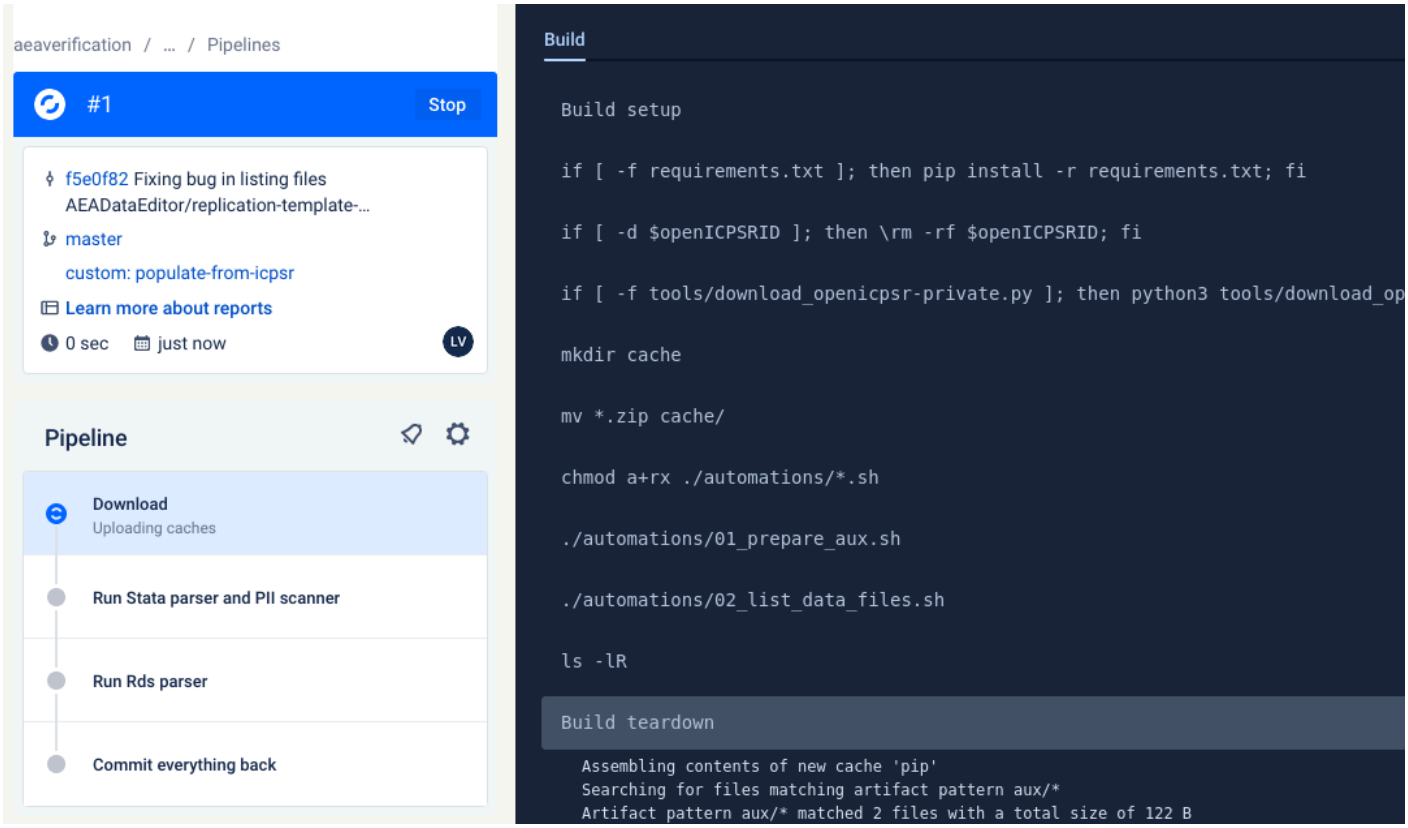
- Choose "`w-big populate from ICPSR`" (might change in the future), and fill in the ID for the relevant source of the replication package (here: openICPSR ID = `123456`), and hit `Run`.



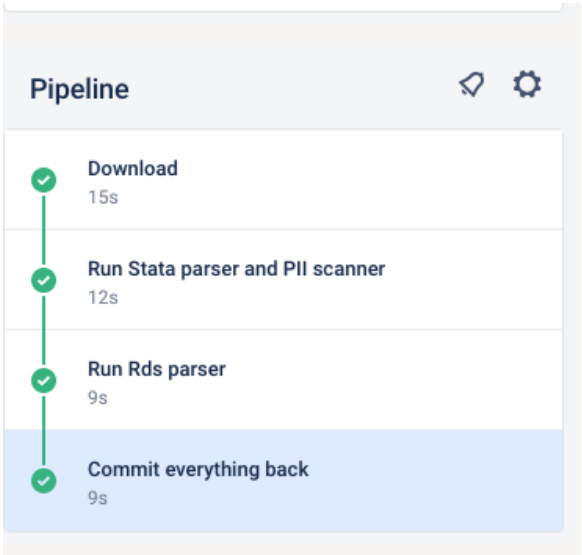
Note that if you choose this pipeline, certain information is not generated (Stata scan, R package scan), and you may need to augment these manually. Try to avoid this pipeline if possible, and make a note in the Jira comments if you had to run this.

Monitoring the pipeline

- Your pipeline will start, working through various steps. This might take a while! Do the next step (Collecting Information) then come back here.



- [ ] Once your pipeline is done, check that it is green.
    - If for some reason, it fails, the logs are available for your supervisor to inspect, and to help you. Check out the possible fixes below. You, or the person assigned to `Part B`, may then need to do the manual steps later.

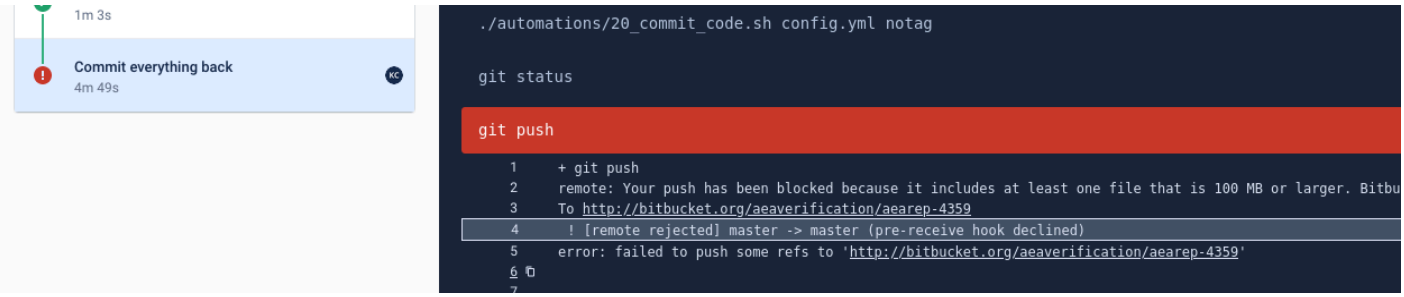


Possible errors for pipeline failure

Files too big

Bitbucket might complain in the `Commit everything back` step that

> remote: Your push has been blocked because it includes at least one file that is 100 MB or larger.



Solution

Investigate which files are being captured that are too big.

You can see the list of large files in the Bitbucket pipeline logs. Find the line in the logs that reads `./automations/01_check_file_sizes.sh $projectID` and click on it to expand:

```
./automations/01_check_file_sizes.sh $projectID
1     + ./automations/01_check_file_sizes.sh $projectID
2     247052/10K/out/mda_item_filtered.csv
3     247052/10K/out/item7_filtered.csv
4     247052/10K/out/item7a_8_p4.csv
5     247052/10K/out/item7a_8_filtered.csv
6     247052/10K/out/item7_p4.csv
7     247052/10K/models/word2vec/w2v_model_trigram.model.wv.vectors.npy
8     247052/10K/models/word2vec/w2v_model_trigram.model.syn1neg.npy
9     247052/10K/models/phrase_models/trigram_10_10
10    247052/10K/data/word2vec/all_tokens_spacy.pkl
11    247052/10K/data/sample_edgar_index.csv
12    247052/External-Data/HS92_allyears.dta
13
```

The list of file endings that Git should ignore is kept in the `.gitignore` file. You can search through that file for any extensions listed in the above listing. Some files do not have extensions, though.

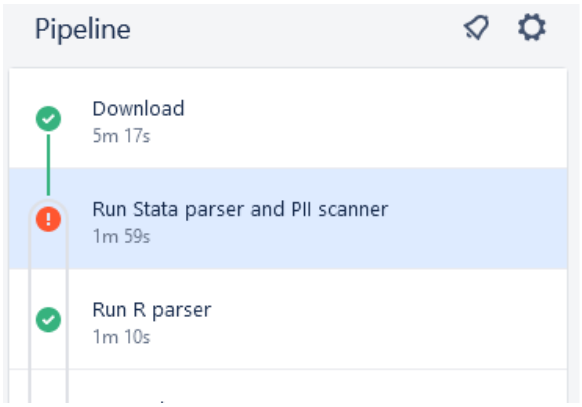
Once you have figured out which files are causing the problem, you should exclude them.

- For instance, if it is a previously unlisted extension, list `*.extension` in the `.gitignore` file.
- If it is an extension-less file, list the entire file (in the example above, you might list `trigram_10_10`).
- If it is a file with an extension you do want to generically keep (e.g., `*.pdf`), then listing the entire path to the file should also work (e.g., if there were a huge PDF, you could list `247052/10K/data/sample_edgar_index.pdf`)

You should make those changes

- in your repository, by adding them into the repository-specific `.gitignore`
- in the template `.gitignore` file, by suggesting an edit. Click on this link, then choose "`Edit`", and add the extension to the file (you will need a Github account to create a pull request).

Memory or CPU usage to high

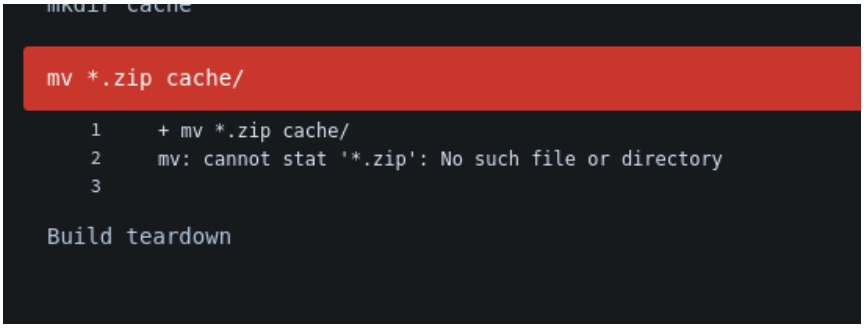


If your pipeline fails in the Stata step, click on the failed step, and scroll to the error message. If you see this:

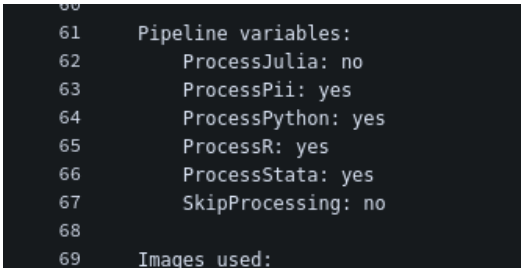
```
./automations/10_run_stata_scanner.sh: line 64:   122 Killed                  stata-mp -b do ../PII_stata
```

It is likely that the PII scan failed because the in-memory dataset is too large (too much memory was run, and the pipeline was killed). Try running the pipeline again with the "`w-big populate from ICPSR`" (see above).

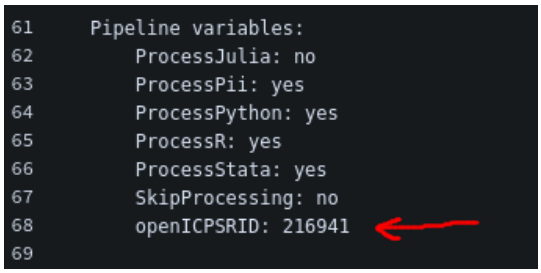
`Cannot stat zip`



This is an indication of an earlier error, typically forgotten variables. To check, click on the `Build setup`, and check the `Pipeline variables`. If it looks like this:



then you forgot to specify the `openICPSRID` variable. Rerun the pipeline with the correct variable entered. It should look like this:



ZIP file is too big

When the ZIP file downloaded from the repository is too big, parts of the pipeline will run, but ultimately, the full pipeline may fail.

To diagnose this part,

- Go back to the list of Pipeline runs
- Click on the failed run
- Click on the `Download` part.
- At the very bottom of the log, you will see `Build teardown`. Click on it to expand. You are looking for lines mentioning `cache/**`:
- A successful run will look something like this:

```
Searching for files matching artifact pattern cache/**
Artifact pattern cache/** matched 1 files with a total size of 26.1 MiB
Compressed files for artifact "cache/**" matching pattern cache/** to 25.8 MiB in 1 seconds
Uploading artifact of 25.8 MiB
Successfully uploaded artifact in 1 seconds.
```

- A **failed** run due to the size of the ZIP file will look like this:

```
Searching for files matching artifact pattern cache/**
Artifact pattern cache/** matched 1 files with a total size of 1 GiB
Compressed files for artifact "cache/**" matching pattern cache/** to 1 GiB in 47 seconds
Compressed artifact size is 39.6 MiB over the 1 GiB upload limit, so the artifact will not be uploaded.
```

If this is the case, try again with the "`w-big populate from ICPSR`" pipeline, which will not try to upload the ZIP file, but will still run the other steps.

Download failed

Since Dec 2025, we have seen occasional failures, with our script being blocked by ICPSR. If you see an error like this:

```
if [ ! -z $openICPSRID ]; then python3 tools/download_openicpsr-private.py $openICPSRID; fi
```

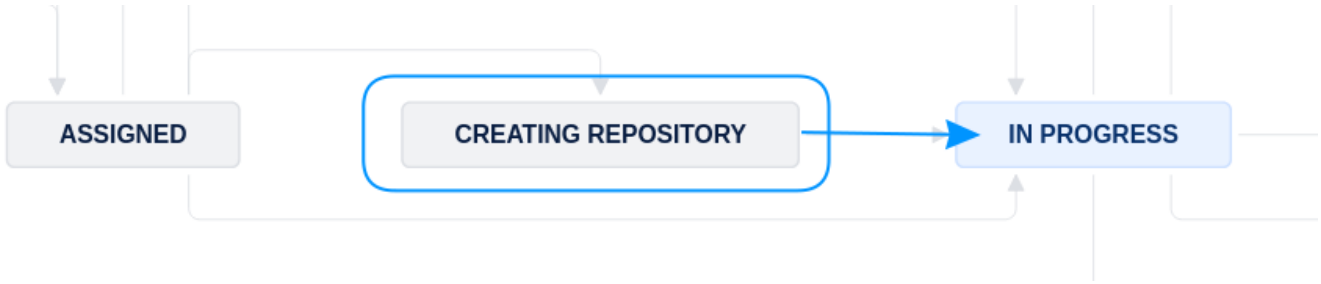
```
+ if [ ! -z $openICPSRID ]; then python3 tools/download_openicpsr-private.py $openICPSRID; fi
openICPSR downloader v2025-12-15
No debug:None

============================================================
System Information:
  Hostname:   6e030ad2-f10a-4475-bb9e-8faa006787f5-r78bg
  Local IP:   10.38.99.30
  Public IP:  35.170.61.140
  Cloudflare Token Used: TRUE
============================================================

============================================================
Starting download for openICPSR deposit: 238704
Traceback (most recent call last):
Save path: .
============================================================

Getting session cookies...
  File "/opt/atlassian/pipelines/agent/build/tools/download_openicpsr-private.py", line 300, in <module>
    req.raise_for_status()
  File "/usr/local/lib/python3.12/site-packages/requests/models.py", line 1026, in raise_for_status
    raise HTTPError(http_error_msg, response=self)
requests.exceptions.HTTPError: 403 Client Error: Forbidden for url: https://www.openicpsr.org/openicpsr/

```

then the download got blocked. There is no need to report it, and usually, it will work after 2-3 retries. Simply run a new pipeline with the same settings (do not use the `Rerun` button).

Next step

Once everything has completed to your satisfaction,

- fill out the `Bitbucket short name` field (e.g., `aearep-1234`)
- Move the Jira issue forward to `In Progress`.



In Progress

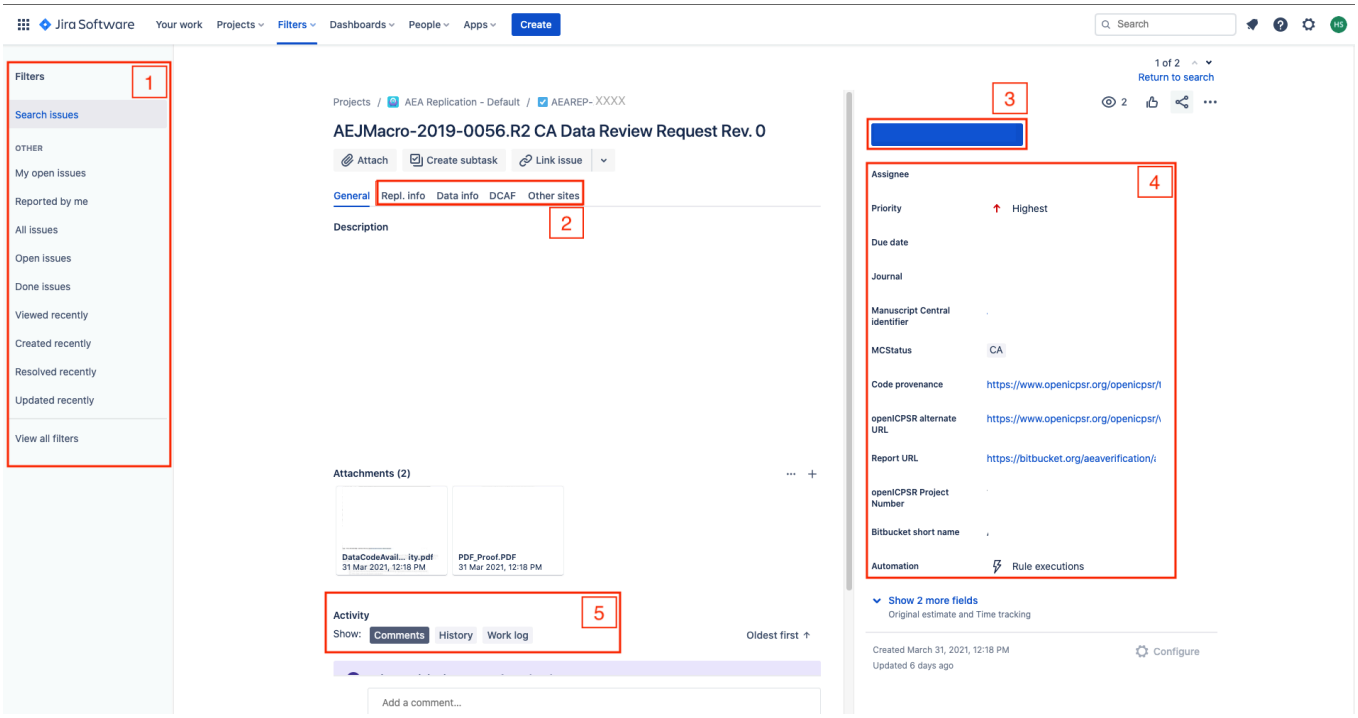
Collecting information

At this stage, the repository exists. You are now collecting information.

- ☐ Fill out the following fields in the Jira ticket (some may be pre-populated):

- [x] `Replication package URL` In almost all cases, this is the openICPSR repo for which you will have received a notification email.
  - If code and/or data are provided by email, `Replication package URL` should be filled out with "https://email", otherwise with a DOI or URL.
- [x] `Journal`
- [x] `Manuscript Central identifier`
- [ ] `Bitbucket short name` (e.g., `aearep-123`)
  - this should then auto-fill `Git working location`.

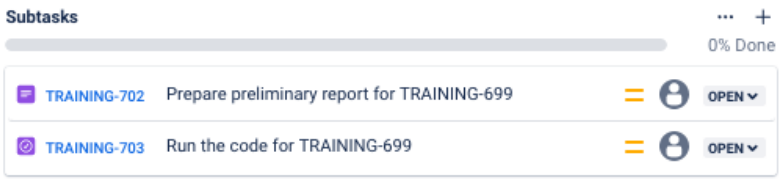
- The following fields, located in the **REPL. INFO** tab area 2 of the screen, must also be filled out:



  - [ ] `Empirical Article`: "Does the article contain empirical work, simulations, or experimental work?"
    - typically the answer should be "Yes". You should answer "No" only if you read the article and find that it is entirely theoretical, no simulations or empirical work at all.
  - [ ] `RCT`: Is the paper about a randomized control trial? This should be immediately obvious from the abstract.
    - `RCT NUMBER`: If it is an RCT, fill in the associated RCT registration number (typically in the title page footnote)

Choosing the next steps

From here, there should be two sub-tasks.



There are two possibilities now:

You complete the subtasks sequentially

- [ ] First, complete the `Prepare preliminary report (Part A)` subtask.
- [ ] Then, `run the code` and complete the Part B report.

You and somebody else complete the subtasks in parallel

In some cases, you will be working with somebody else.

- One person does `Part A`: _____
- Another person does `Part B`: _____

In general, the person doing `Part A` should then also, once `Part B` is completed, merge in the information from `Part B` and write the final `Findings` section (`Part C`).

We will describe Parts A and B in sequence, but as needed, you should skip ahead.

Preparing Part A - Preliminary Report

> **Note**
>
> - Link to JIRA: https://aeadataeditors.atlassian.net/jira (requires login).

Start the report

To signal that you are starting the report, you will now transition the JIRA subtask to "`Writing Part A`".

Prepare your working area

> **Note**
>
> For **Part A**, you do NOT need the data, but you do need the working copy of the repository. It is not important what computer you prepare this part, you can do it on CCSS or your laptop.

Pull everything together

Changes in May 2026 Click to hide

In May 2026, we started adding the manuscript and DCAF to the repository as part of the automated processing. You probably no longer need to manually add the manuscript and DCAF.

CCSS | **Manual steps** | **BioHPC**

- [ ] Clone the Bitbucket repository onto the CCSS server you are working on

```
git clone https://yourname@bitbucket.org/aeaverification/aearep-xxx.git
```

- [ ] Alternatively: If you have done the full Bash setup, you can run

```
aeagit xxx
```

- [ ] Verify that the code is present, i.e., that the automated scripts run during the Code Ingest worked. Also check that the manuscript and DCAF were added. **If they did not, you need to switch to the "Manual steps"!**

Be sure to use the `REPLICATION-PartA.md` for this section! Click to hide

As part of the automated processing, the `REPLICATION.md` is split into two parts, `REPLICATION-PartA.md` and `REPLICATION-PartB.md`. Somebody else may be working on Part B at the same time as you are working on Part A. Please be sure to use the correct file for your work.

- The root of the repository should contain only our files (i.e., `REPLICATION-PartA.md`, `REPLICATION-PartB.md` etc.), the manuscript files (main manuscript, any online appendices and README files provided through the JIRA ticket), and the code.
  - Example:

```
111234/
code-check.xlsx
config.do
PDF_Proof.PDF
PII_stata_scan.do
DataCodeAvailability.pdf
REPLICATION-PartA.md
REPLICATION-PartB.md
REPLICATION.md
```

Verifying completeness of the README

Now that you have pulled everything together, you will inspect the authors' README, and do a first analysis of the necessary data.

Where is the README? Click to hide

The README should have been pulled in when you did the `aeagit` or `git clone` command, and be in the `12345` folder corresponding to the openICPSR repository number.

If it is not there, there is a chance that it is called something non-standard, and was excluded. Check against what you can see on openICPSR! If it is not there, contact your supervisor.

The first section of the Replication report is titled "General". Here you want to verify how complete the README is.

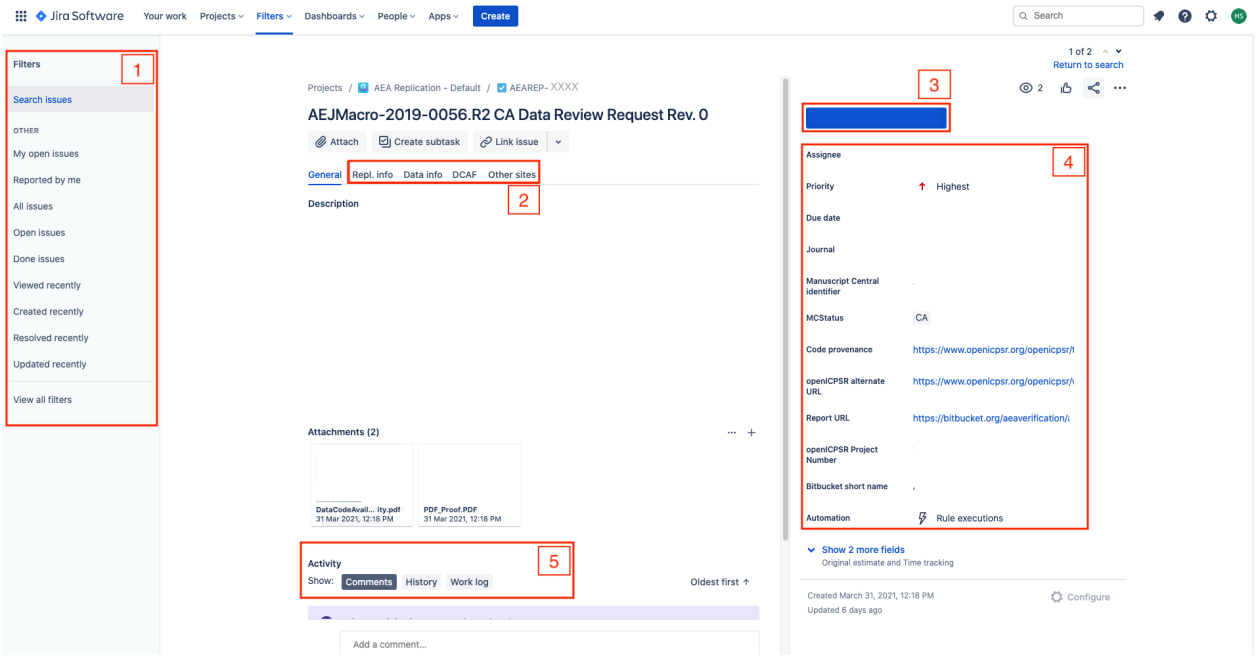
- Go through the entire README, and compare both to the template README and the sections in the **General** section.
  - Check off the sections for which you appear to have information.
  - Leave blank the sections that are either not provided, or that are empty
- In JIRA, identify the question called "`DCAF_README_compliant`".
  - If all or most of the sections are in the same order, and are filled with meaningful information, then check `Fully`
  - If information is provided, but in different order, or only mostly, mark "`Content only`".
  - If very little of the information is provided, mark it as "`No`"

List of Data Sources

Now you will establish a **list of data sources used**.

- Please check the **Data and Code Availability Form**.
  - The form should be attached in the JIRA ticket.
  - In area 2 of the screen, choose `DCAF`.
    - Open the **Data and Code Availability Form**, and check if all blanks are filled out.
    - Once you checked the form, and in particular, if the checkbox next to "README" is checked, choose "`Yes`" from the dropdown menu of `DCAF_README_checked` cell.
    - If the answer to the following question at the bottom of the form is "Yes", then, choose "`Yes`" from the dropdown menu of `DCAF_Access_Restrictions`. Otherwise, choose "`No`". "Is any of the data used in this manuscript

subject to access restrictions?”



- From the **README** provided by the authors, the **data section of the article itself**, or an **appendix**, establish a list of data sources used in the article. For **each data source**
  - ☐ write the corresponding `Data description` section of `REPLICATION-PartA.md`. This should provide detail about the datasets
    - If data are cited, copy and past the citation to the replication report, clarify which one you are referring to. Be sure to check AEA Sample References and the additional guidance to be sure it is a **data citation**, and not a citation to an article or a document describing the data!
  - ☐ check any provided URL, and verify if there is a **“Data Use Agreement”, “Citation requirement”, “License”** on the web page. Check any such data use agreement for conditions. These may require that the authors cite a particular paper, or cite the data in a particular way (check this), or that the authors may not actually redistribute (provide) the data (check this!). If you have doubts, check with your supervisor.
  - ☐ Check that there is enough information to obtain the data in the README. Based on the README, you should be able to find, on the linked website, the data that you would need. (Ignore at this point that the data might be provided)
- ☐ Add the list of data sources to the repository by committing the preliminary version of the `REPLICATION-PartA.md` (`git add`, `git commit`, `git push`)
- ☐ Fill out the `DataCitationSummary` field indicating how many data citations are in order: `all`, `some`, or `none`.
- ☐ Fill out the `Data Provenance` section
  - Are the data in the openICPSR repository, or are they someplace else? “Various” is a legitimate answer if data are in various locations.
- ☐ Please refer to A guided walk through the Replication Report for more details about which data sources to include and how to assess the provided information.

Note

What is the difference between a "**data source**" and a "**dataset**" or "data file"?

- A **data source** is more general. A data source may provide multiple data files. One example is the "Current Population Survey".
- A **dataset** or **data file** is a single file that appears in the replication package. An example is "`data_cps.dta`", which is presumably the data file containing the Current Population Survey.

For this section, you should list **data sources**.

Warning

When assessing the data, please take care to distinguish

- data that is part of the openICPSR deposit
- data that the README tells you to download or otherwise access
- data that you are provided on the L-Drive, which is typically provided under an agreement with the authors, and cannot be redistributed.

Assess the openICPSR deposit

If the data are in openICPSR, you will now assess the deposit. The form in the template report should give enough guidance on what to check!

If the data are restricted-access

If the data are restricted-access, you will still fill out this part. However, you should **not** attempt to request access to the data yourself. Contact your supervisor. See Requesting Restricted-Access Data from Authors for details.

Review the Part A report again

At this stage, you go back and review the Part A report again. Identify any actions you think the author should make to bring the deposit into compliance with our requirements.

- There is sample language for commonly encountered problems in the sample-language-report.md, which is part of the replication package.
  - Select an appropriate tag, and copy-paste into the `REPLICATION-PartA.md`

Commit Part A to the Bitbucket repository.

```
git add REPLICATION-PartA.md
git commit -m "Preliminary report"
git push
```

Completing JIRA fields

You are now ready to complete the subtask. In the transition screen, you will be asked to fill out the following fields:

- [ ] `DATA PROVENANCE` Where, specifically, are you accessing the data? Typically you can write here "`openICPSR`", but it may also be a user-provided URL or DOI, or a different source
    - if the data is in multiple places, enter "Multiple" here, and record the details in the REPLICATION.md

You can now proceed to change the status to `Preliminary Report complete`. You will be asked to provide additional information:

- [ ] `DATASETSINCLUDED` Are all datasets included as part of the replication package (on openICPSR or, if not using openICPSR, on the other repository)?
- [ ] `DATAAVAILABILITYACESS` Do the data require users to apply for access, purchase, or otherwise sign or enter into agreements to access the data? This could be a license agreement, or even a click-through acknowledgement. (This should be mentioned in the Readme PDF or in the article)
- [ ] `DATAAVAILABILITYEXCLUSIVE` Are there data that are **only** accessible to the author (nobody else)?
- [ ] `REASON FOR NON-ACCESSIBILITY OF DATA` Fill this out for **any** data that is not accessible/ not included as part of this archive; check **all** that apply. This should be clear from the authors' descriptions (in the README)
    - `Too big`: The data can be accessed elsewhere, but they are too big for this replication package
    - `Application process`: In order to access the data, an application needs to be made to an institution (not a purchase).
    - `Cost`: It costs money to obtain the data. This may be because it has to be purchased, or because there is a fee for the application process.
    - `Confidential data`: The data are sensitive / confidential and are therefore not made available in this replication package. They can be available elsewhere, subject to conditions.
    - `Proprietary data`: The data "belong" to somebody - a company, or in rare cases, a specific author, and cannot be redistributed.
    - `Licensed data`: A license must be obtained. This can be different than an application process (generally, less complicated).
    - `Redistribution not authorized`: Often, even if data are not confidential, not proprietary, etc., there may be redistribution restrictions. An example are some IPUMS data, as well as many others.
    - `Other download site provided`: When data can be downloaded elsewhere, possibly due to `licenses` or `application process`. In other cases, even if they could be provided, they may already be archived elsewhere, and are not included here.
    - `Not found`: This should be checked when data cannot be found as per the instructions by the author. This is rarely a final finding for pre-publication verification.
- [ ] `NUMBEROFDATASETS` How many datasets are used in the article (whether or not they are included in the replication package you downloaded)? This is meant to include datasets that you are asked to download, or that you were given access to via the "L:" drive, or "CRADC", or some other secure mechanism.

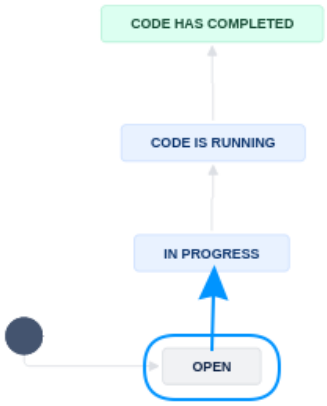
Next steps

You should now move the subtask to "`Preliminary Report Complete`" to signal that this part of the processing is done.

Preparing to Run Code (Part B)

Signal that you are working on Part B

To signal that you are starting to run the code, you will now transition the JIRA **subtask** to "`In Progress`".



If you are the replicator who was previously assigned Part A Click to hide

In this case, your working area is already prepared, and you can skip to Get the data.

If you are NOT the primary assignee of this case Click to hide

then you should do some additional steps to prepare your work:

- do not edit any of the files that relate to Part A. These are the responsibility of the Part A assignee.
- **do create a branch of the repository**, called `partb`, and make all your commits to that branch. We describe how to do that further below.

Prepare your working area

Before we can verify code, data, and documentation, we need to first get the code, then the data into your "working area". Let's start with the **code**.

Note

You now need to decide on what computer you are going to do the data analysis - that should be the place you do the next few steps. See Access Computer for details. This is because the git setup we use does not allow you to include the data files in the Bitbucket repository, so when you download the replication package from openICPSR or elsewhere, they do not get added to the Bitbucket repository.

Get the code via the existing Bitbucket repository

CCSS | **BioHPC** | **Github Codespaces (CS)**

- ☐ Access the CCSS environment: shortcut to Cloud, for other access, see Appendix.
- ☐ Ensure that you have set up your CCSS environment (see appendix)
- ☐ **Get the code:** Clone the Bitbucket repository onto the CCSS server you are working on

```
git clone https://yourname@bitbucket.org/aeaverification/aearep-xxx.git
```

- ☐ Alternatively: If you have done the full Bash setup, you can run

```
aeagit xxx
```

Verify that the code is present

With the first step, you obtained a copy of the Bitbucket repository. You should now have a folder called `aearep-123` (or whatever the repository number is). All subsequent steps should be done from there.

Important

If you do not see a folder like `123456` or `dropbox-xyz` in the repository, then the Bitbucket Pipeline likely did not work. You will then need to populate the code (and data) manually. You will do this AFTER downloading the data, in the next step.

If you are ONLY doing Part B Click to hide

now is the time to create a branch for Part B.

- Create a branch called `partb` and switch to it.

```
git checkout -b partb
```

- Push that branch to the remote repository, so that it is available for others to see and use.

```
git push -u origin partb
```

Verify that you can actually run the code

Before you spend time obtaining the data, now is a good time to assess (again) if you have everything needed to run the code.

Have a look at the **README** again.

- [ ] Is there information about **Requirements**?
  - Is there information about the **software**?
  - How long does the author say the code will run? Is it a reasonable time, or do we maybe need to run it on a more powerful computer?
  - How much memory, or processors, does the code need? Again, is the computer you intended to choose sufficient, or do we need to get access to a more powerful computer, or even a cluster of computers?
  - What operating system (Mac, Windows, Linux) does the author appear to have used?

Now fill out the **Stated Requirements** section of **Part B** of the report.

- Check if the deposit has a 'main' or 'master' file and fill out the 'MainFile' field in Jira under the 'Repl. info' tab.
- Mark the **operating system (OS)** that the authors used in the field `Original OS`. Leave blank if you do not know.

> **Be sure to use the `REPLICATION-PartB.md` for this section!** Click to hide
>
> As part of the automated processing, the `REPLICATION.md` is split into two parts, `REPLICATION-PartA.md` and `REPLICATION-PartB.md`. Somebody else may be working on Part A at the same time as you are working on Part B. Please be sure to use the correct file for your work.

Prepare the code-check

Now is a good time to understand the code in a bit more detail:

- [ ] In the template, you will find *code-check.xlsx*.
  - Use this to create a list of all Tables and Figures in the paper
  - You will use this to guide later to tabulate your findings!
- [ ] Fill out the "**Code Description**" section of the `REPLICATION-PartB.md`
  - Provide some information about the program files (are there 3 Stata files? Are there 5 Matlab programs?). You will use this information to fill out the `Software Used` (in the main task) later as well, but provide details here.
    - You can use the file "`generated/programs-list.txt` to help you here.
  - Did you have difficulty aligning the README with the files? Does the sequence suggested by the programs differ from what's written in the README?
  - Are there files in the archive not explained in the README?
  - Copy-and-paste the *code-check.xlsx* into the code description part, listing the programs. Omit the "Reproduced?" Column in doing so. Use the Excel-to-Markdown plugin for VSCode.
    - This table will be pasted in under "Findings" again, with "Reproduced?" column, once code has been run.

Verify that you can actually run the code

> **Do you think you know how to run the code in the software mentioned?**
>
> You may not have the right experience, talk to your supervisor!
>
> If both of you agree that **nobody** will run the code, then
>
> - Move the **subtask** to "Part B is complete" via `No code was run`
> - Go straight to Part C, writing the **Findings** section

Describe the provided data

The automated scripts should have filled out the "**All data files provided**" section, but if not, please do so here.

If the list is VERY long, put it into an appendix, but make a note in this section that there is an appendix with this info.

> **What if there are no data provided at all?**
>
> If **there are no data at all**, for instance, when data are confidential and only available through some computer system at the Census Bureau or in Sweden?
>
> Then you will skip getting the data and running code, and you are done with **Part B**!
>
> - Move the **subtask** to "Part B is complete" via `No code was run`
> - Go straight to Part C, writing the **Findings** section

If there is ANY data at all present, then continue to get the data.

Get the Data

If you think that you are ready to run the code, you need to get the data. When getting the data, please take care to distinguish

- data that is part of the openICPSR deposit
- data that the README tells you to download or otherwise access
- data that you are provided on the L-Drive, which is typically provided under an agreement with the authors, and cannot be redistributed.

Here, we will describe the most likely first step: getting the data from openICPSR. Any data you download should also be stored on this computer. We do not explicitly describe this here. **CCSS** is the most likely place where you do this, but double-check with your supervisor.

What if the data are not on openICPSR? Click to hide

Sometimes, data are provided on other repositories: OSF, Dataverse, Zenodo are the most frequent ones.

You may need to adapt the process below to those circumstances. See the following mapping:

Deposit	Name of download folder	Name of script	Tested?
openICPSR	`111234`	`tools/download_openicpsr-private.py`	Yes
OSF	`osf-ZX123A`	`tools/download_osf.sh`	Partially
Zenodo	`zenodo-1234567`	`tools/download_zenodo.sh`	Maybe
Dataverse	`dv-2YWLWG`	`tools/download_dv.py`	Maybe

In all cases, you should obtain the data from the deposit, and otherwise follow the same principles as are described below for openICPSR.

CCSS | **BioHPC** | **Github Codespaces (CS)** | **Manual steps**

- ☐ Access the CCSS environment: shortcut to Cloud, for other access, see Appendix.
- ☐ In some cases, you may be asked to use (restricted) data on the S: drive or L: drive. Follow instructions as you receive them.
- ☐ Download the openICPSR data (if available).
  - Try to do this first using scripts. See the details in the appendix.

```
python tools/download_openicpsr-private.py 111234
```

```
unzip -n 111234 -d 111234
```

  which should unpack the data files only, not overwriting anything else. **If this fails, do the "Manual steps."**

- ☐ attempt to download data from various sources indicated by the authors, but ONLY if no sign-up/ application process is involved.

Did the Bitbucket Pipeline scripts work?

In cases where the package downloaded from openICPSR is too big, or where the data do not come from openICPSR, the automated **Bitbucket Pipeline** scripts will not work. In this case, you will need to run some additional steps to run the automated scripts. **This is best done on BioHPC or CS**.

The Pipeline worked! | **Running on CCSS** | **Running on BioHPC** | **Running on Github Codespaces (CS)**

You don't have to do anything additional, you can move on.

Ready!

You are now ready to run code.

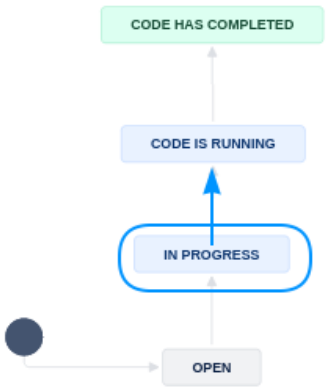
Running Code (Part B)

In this stage, you will run the code, by using the provided data. The `REPLICATION-PartB.md` is the report!

> **Be sure to use the `REPLICATION-PartB.md` for this section!** Click to hide
>
> As part of the automated processing, the `REPLICATION.md` is split into two parts, `REPLICATION-PartA.md` and `REPLICATION-PartB.md`. Somebody else may be working on Part A at the same time as you are working on Part B. Please be sure to use the correct file for your work.

Signal that you are working on Part B

To signal that you are starting to run the code, you will now transition the JIRA subtask to "`Code is running`".



Principles

Keep a log of what you do, what you find, and what does not work, in the `REPLICATION-PartB.md`, under *Replication Steps*.

- Document if you manually downloaded data from anywhere other than the authors' deposit.
- Document any manual changes you made to the deposit structure (renamed file, created directory).
- If you run into error messages that are not captured by the "log file" (see below), then screenshot, or copy verbatim, into the report.

You should also run code so that it generates an actual "log file". How to do this depends on the software.

> **We have an entire section of the manual for this!**
>
> Check the software-specific guidelines in the manual's section on running code, debug the code, and set the code to run.

Note

Sometimes, code can run for a long time. Leave the Jira ticket in `Code is running` to signal that there is activity there. Remember to come back regularly to check on it, and remember to commit all changes immediately (before running code).

If you appear to be stuck…

then as soon as possible, call on

- your supervisors (Sofia, Lars)
- more senior RAs (pre-approvers) or graduate students
- bring it up in a Monday or Thursday meeting

Do not let yourself be blocked too long!

Once you have run the code, come back here to continue!

Details

- If code runs for a long time, record in the Jira issue (as a comment) when you started the code.
- Use the `Computing Environment` to record where (CCSS, BioHPC node, etc.) you started the code.
- It is not unusual for code to run for several days. You can start a new case in the meantime!

When you run into problems

Before you even write up your Findings, you may run into problems. In order to clearly document the problem to somebody else on the team who can reference it, state the problem clearly in the Comments of the Jira ticket. Ideally, you will also reference the filename, both by name and by reference to location within the repository, as well as a log file showing the problem.

Referencing Logfiles Click to hide

You should point to the exact line where the problem appears to show up, as follows:

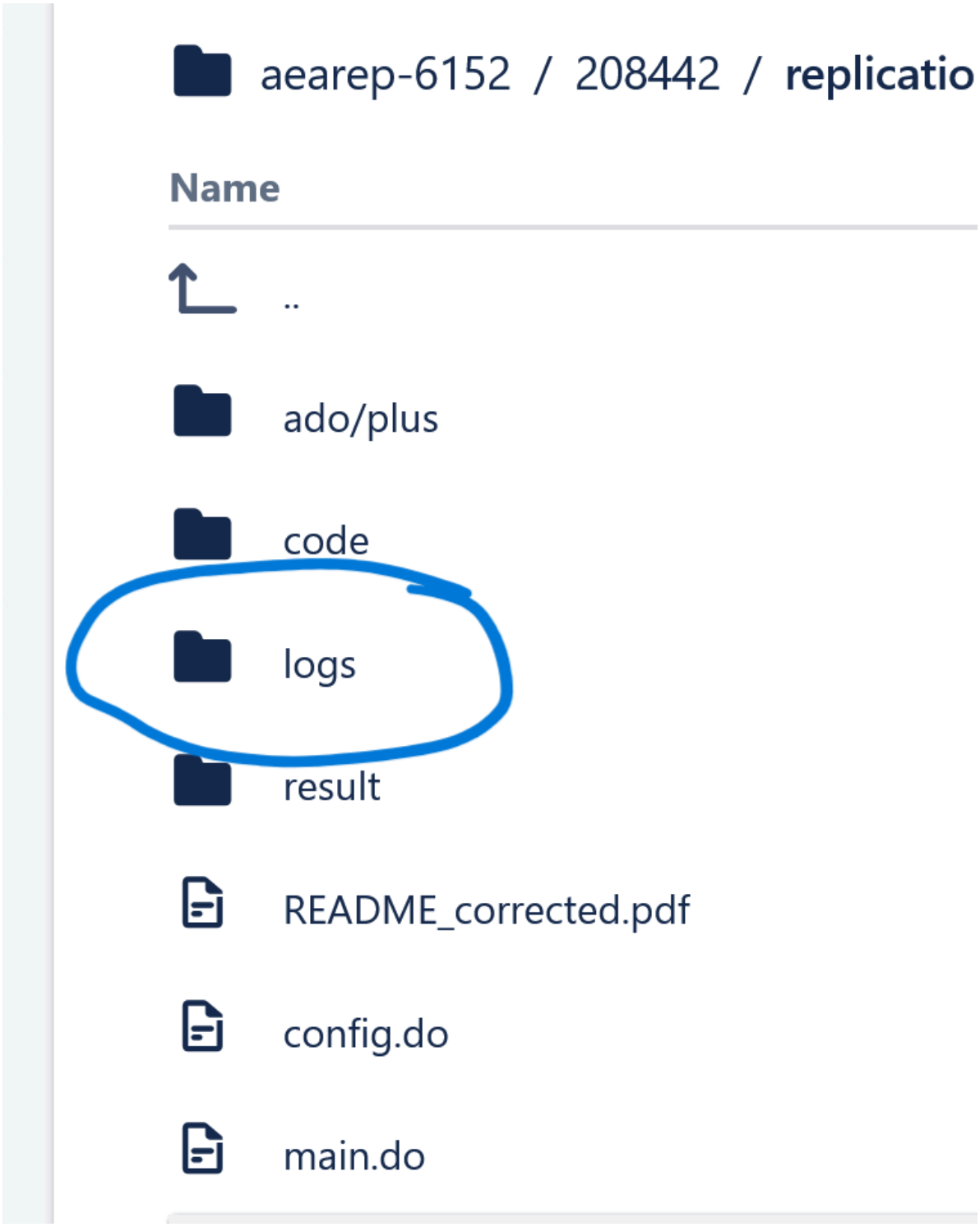
Step 0

Before doing anything,

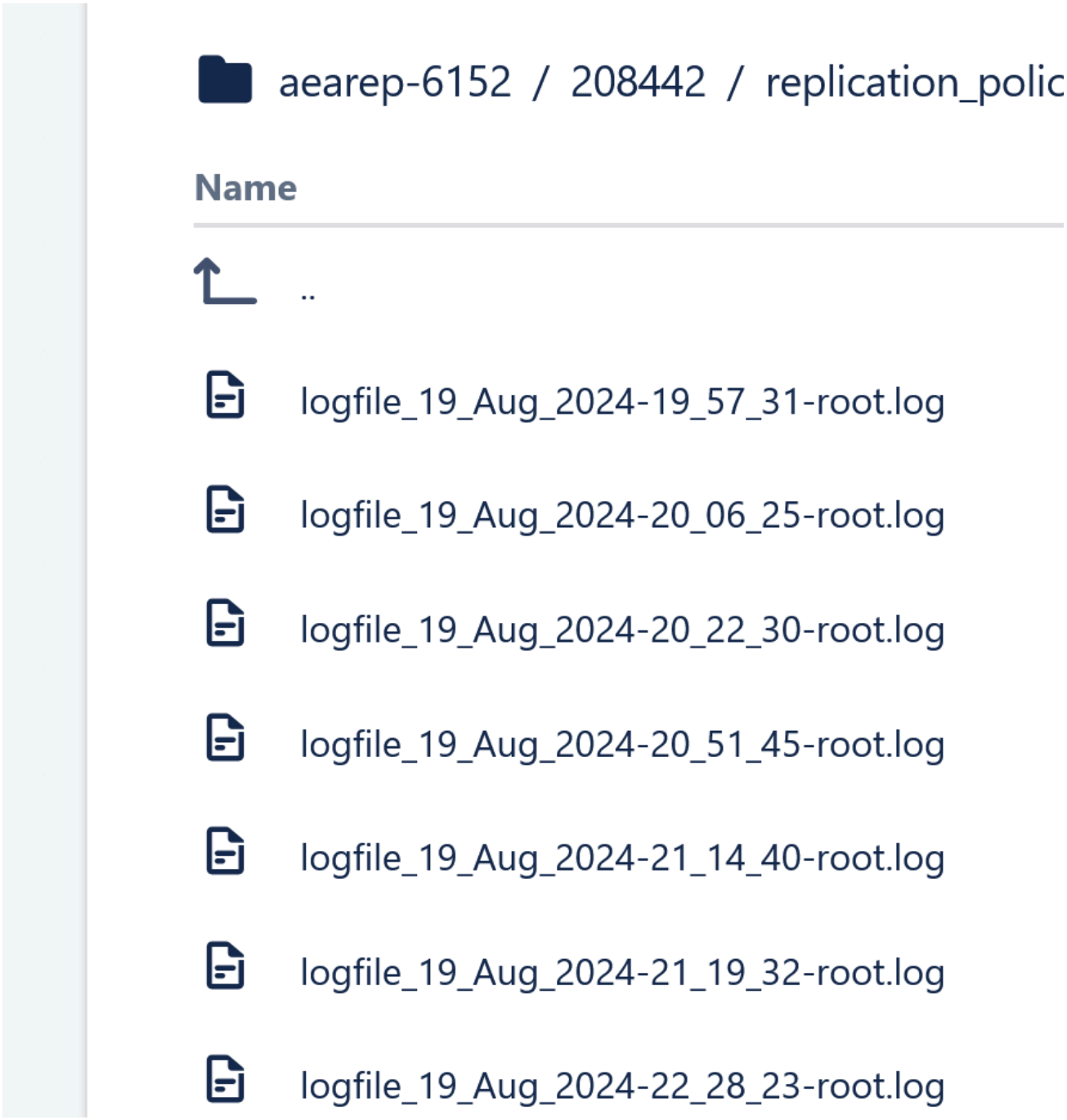
- `git add` any modified and any added files
- `git commit` with a commit message that identifies the problem
- `git push`

Step 1

Navigate to the Bitbucket repository of this case. If you are doing everything right, you should see the files you just committed and pushed, as well as the log files you are creating (if you are not creating log files, review Stata instructions, R instructions, or other relevant instructions). You should see a `logs` directory:



Navigate to the logfile you want to reference:



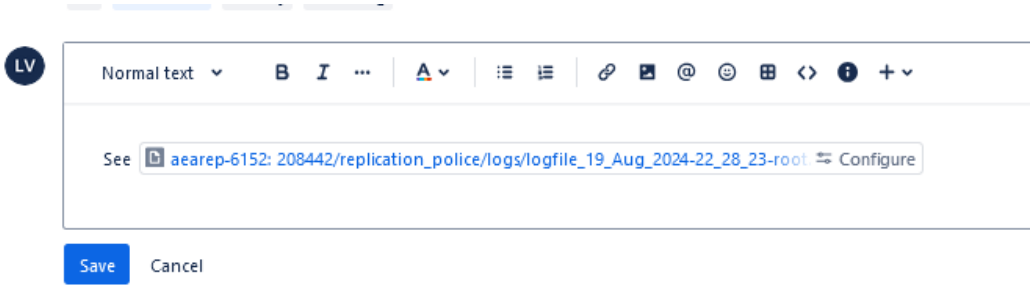
In general, the error will be in the last lines of the logfile:

```
184  > c.ado
185  *! version 2.4.0  13apr2018  Ben Jann
186  /opt/atlassian/pipelines/agent/build/208442/replication_police/ado/plus/
187  > t.ado
188  *! version 3.31  26apr2022  Ben Jann
189  /opt/atlassian/pipelines/agent/build/208442/replication_police/ado/plus/
190  > vtest.ado
191  Installing zscore
192  ssc install: "zscore" not found at SSC, type search zscore
193  (To find all packages at SSC that start with z, type ssc describe z)
194  r(601); t=0.09 22:28:23
195  .     }
196  r(601); t=0.09 22:28:23
197  r(601); t=0.00 22:28:23
198
199  end of do-file
200  r(601); t=0.00 22:28:23
201
```

Click on the line number (on the left margin). This should highlight a relevant line:

```
188  *! version 3.31  26apr2022  Ben Jann
189  /opt/atlassian/pipelines/agent/build/208442/replication_police/ado/plus/w/
190  > vtest.ado
191  Installing zscore
192  ssc install: "zscore" not found at SSC, type search zscore
193  (To find all packages at SSC that start with z, type ssc describe z)
194  r(601); t=0.09 22:28:23
195  .     }
196  r(601); t=0.09 22:28:23
197  r(601); t=0.00 22:28:23
198
199  end of do-file
```

Now copy the URL, and paste it into the Comment field of Jira:



This allows you point the reader/helper to a specific line immediately, making it very clear where you think the error occurs. In some cases, the error might actually happen earlier, and the logfile provides a lot of information, if not always enough, to help debug.

Wrapping up Part B

Once the code has run, **error-free and creating the intended output** (as far as you can tell), it is time to wrap up this sub-task.

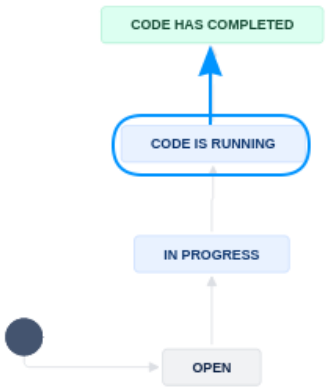
- [ ] **Commit all changes and outputs** to the repository. This includes the log files and any other output files that were generated.

```
git add (path to log files and output)
git commit -m "Successfully ran code all the way through"
git push
```

- [ ] Review the **Replication steps** section again, please include all steps that you undertook.

> **Note**
>
> You, or somebody else, will compare the output to the manuscript in the next step! You may, however, simply check that output exists, regardless of its specific content, before moving forward.

> **Commit this part of the report to the Bitbucket repository.**
>
> ```
> git add REPLICATION-PartB.md
> git commit -m "Completed Part B"
> git push
> ```

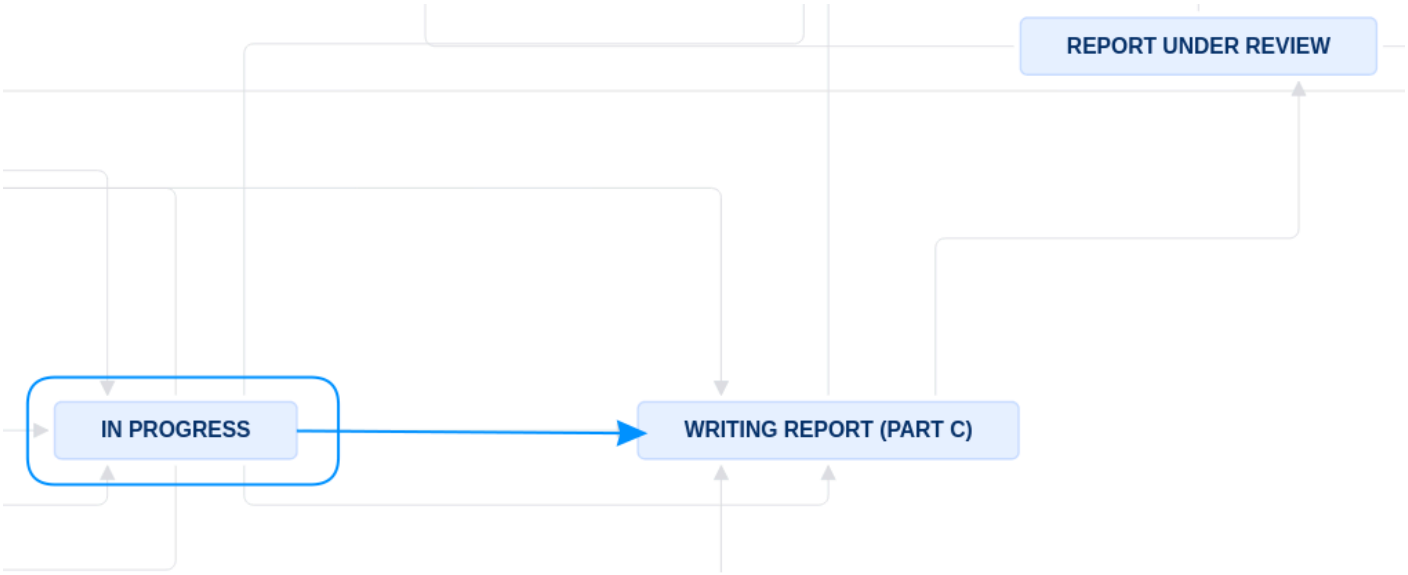


Writing Report

At this stage, you will write the final version of the report.

Signalling where you are

Move the Jira issue to "`Writing report`".



In order to do so:

- Check that both sub-tasks (part A and part B) are marked as "`Done`".
- If there is a sub-task about *restricted data*, it must either indicate that data were received, or that data were deleted. However, do **NOT** delete restricted data yourself.

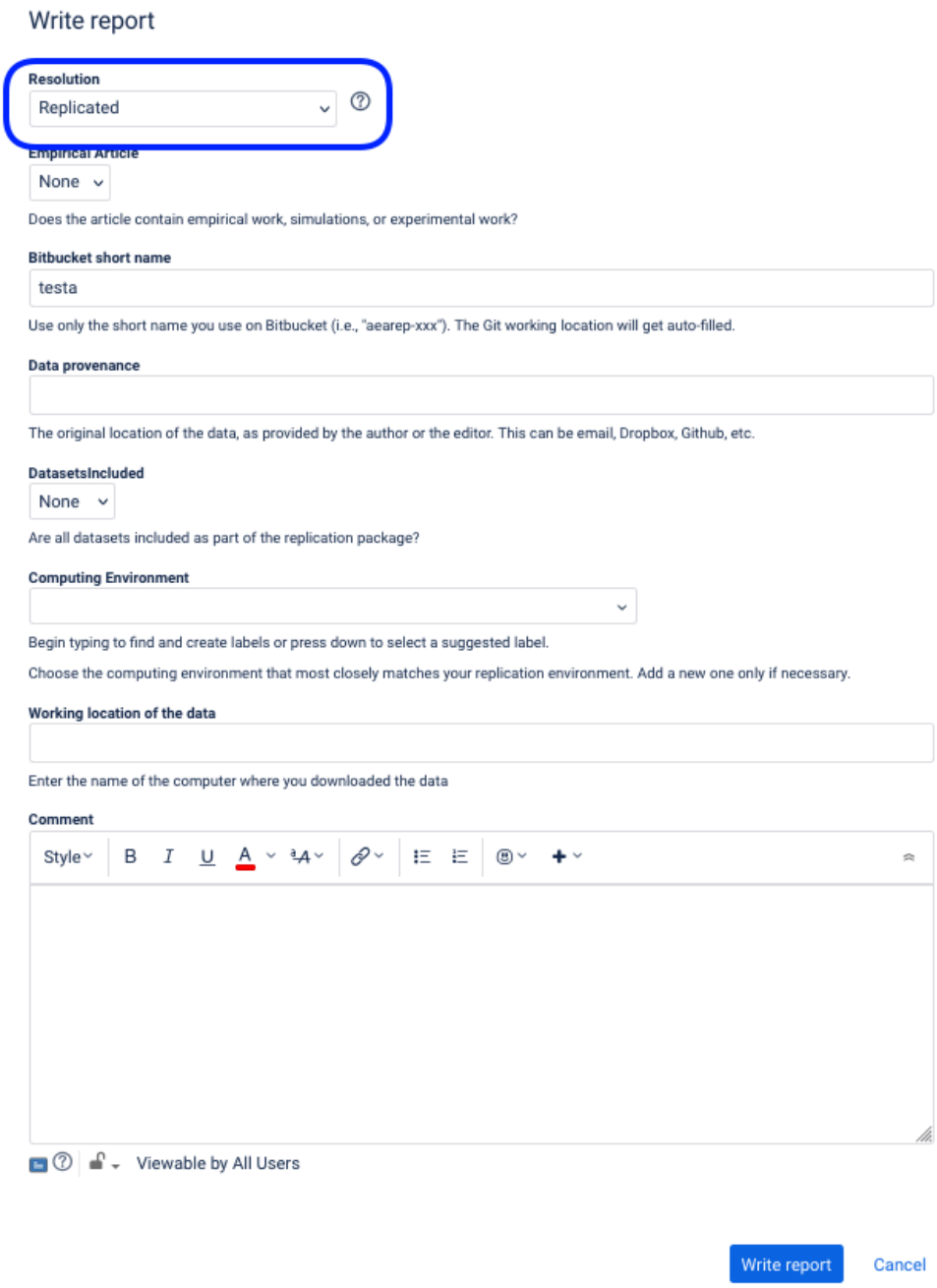
If you were NOT doing Part B Click to hide

now is the time to merge the other replicator's branch.

- If not already alerted via a comment, create a "pull request".
- Go to the `Git working location` (bottom right)
- Navigate to `Pull requests` (left panel).
- If you see a pull request, verify that it is there to merge `partb` into `master`.
- If you do not see a pull request, create one:
  - Click on `Create pull request`
  - Select `partb` as the source branch, and `master` as the destination branch.
  - Then merge it.

Everything else from here on is as it normally would be, except you have to rely on the other replicator's notes in the `REPLICATION-PartB.md` file, and any comments they have left in the repository, to write your report.

You should now be able to select "`Write report`" from the dropdown menu. A pop-up will appear, asking you to fill out the fields and confirm the transition.



Important!

You should choose here the "`Resolution`" that corresponds to the outcome of the replication. This is preliminary - you will be able to adjust it after completing Part C, but based on what you observed in Part B, you may already have a good idea of the outcome. If uncertain at this point, choose `x-------x`.

Run the Merge Pipeline

At this point, there will be two parts of the report: `REPLICATION-PartA.md` and `REPLICATION-PartB.md`. To facilitate further editing, we will merge these two parts into a single file, `REPLICATION.md`. This can be done manually, or using the pipeline.

Using the Pipeline **Manual Merge**

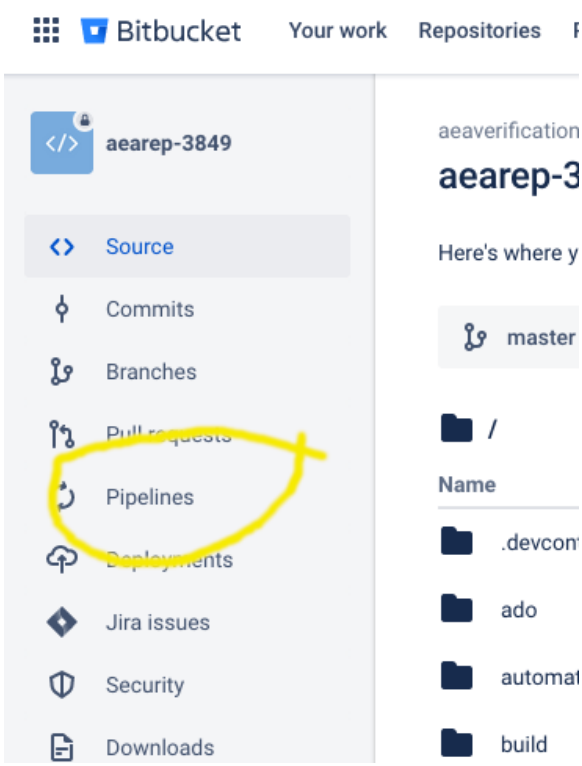
Run the pipeline to merge the two parts.

Important!

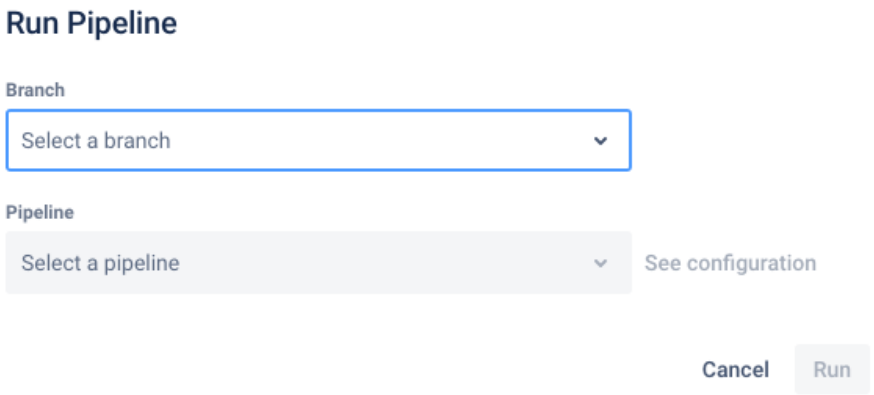
Be sure to have committed and pushed **all** changes to the repository before running the pipeline.

Finding the Pipeline menu Click to hide

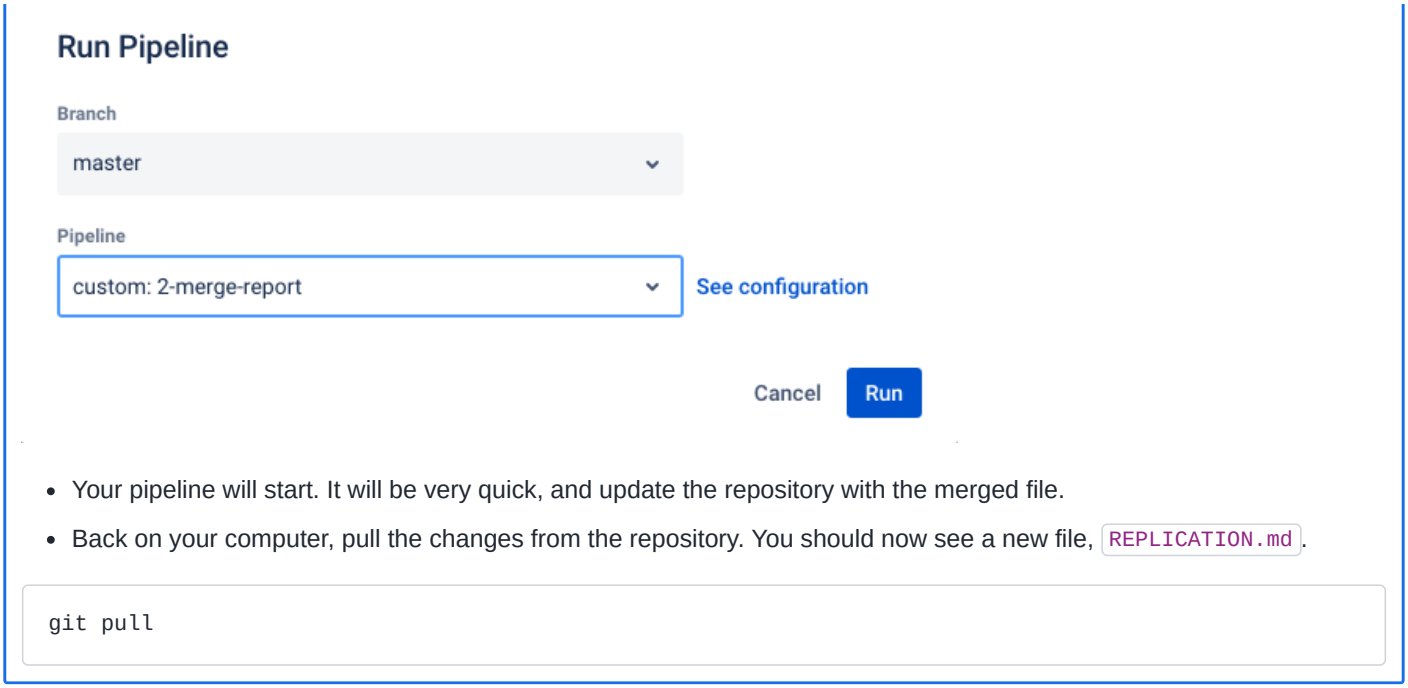
- First, in the Bitbucket repository for your issue (you can find the link under `Repl.info` -> `Git working location`) , navigate to the `Pipelines` tab



- You will now need to select a “pipeline” to run.



- Choose “`2-merge-report`” (might change in the future), and hit `Run`.



- Your pipeline will start. It will be very quick, and update the repository with the merged file.
- Back on your computer, pull the changes from the repository. You should now see a new file, `REPLICATION.md`.

```
git pull
```

Warning

From here on, be sure to use the consolidated `REPLICATION.md`!

Standard steps

- Compare images and tables
  - Some of this may involve “squinting” at images…
  - For tables that are output to Excel, you may be able to use some Excel tools (copy the old numbers, create a difference function between all cells). This is hard to provide examples for, as every table differs.
  - For tables that are output as LaTeX (files ending with `tex`) or as CSV files, *and* if the authors provided previous versions, you may be able to use Bitbucket to compare them globally, without looking at individual numbers (see **Comparing TeX files on Bitbucket**).

Warning

We want to learn about ALL differences that you see, even if minor.

- When there are differences: Include images of figures and screenshots of tables (both paper and as-reproduced) in the report
  - For paper images, use a screenshot tool to grab them from the `PDF_Proof.PDF` and save them as PNG.
  - For reproduced images, reference the location within the replication directory.
  - Using an annotation tool on your computer, circle or highlight where you see differences.

Tip

See detailed step-by-step guidance in the Appendix.

Example:

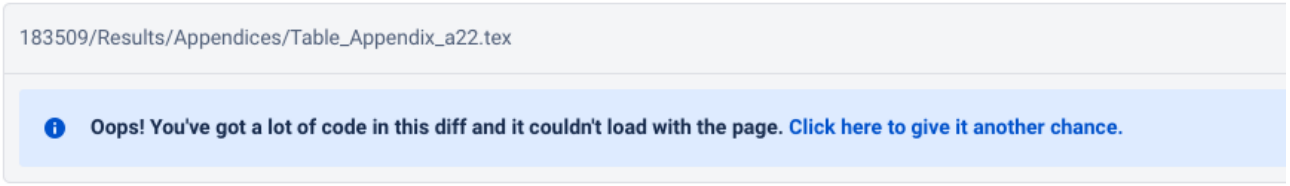
```
Figure 5 in the paper:

![Figure 5 in paper](figure5-paper.PNG)

Figure 5 as reproduced:

![Figure 5 reproduced](12345/results/figure5.png)
```

- Highlight differences:
  - if only a small number of table entries: mention them by table in the report
  - if a larger number: Highlight on the reproduced images (of figures, screenshots of tables) the differences you have observed
- If there are **too** many differences, let us know immediately, we may then simply send back to the authors with the raw output, and let them address it.
- There is sample language for commonly encountered problems at the sample-language-report.md link in the tall grey bar

Advanced: Comparing TeX files on Bitbucket Click to hide

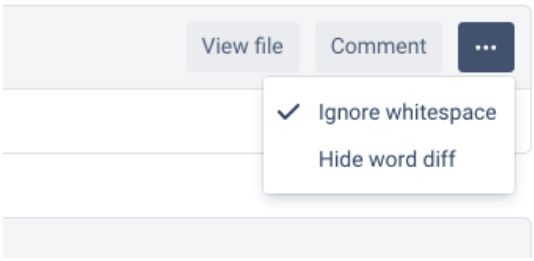
If the authors provided LaTeX files as part of the replication package, you may be able to leverage Bitbucket or Git to compare them.

- Note first that the typical table when output, if identical, will not register as “modified” with Git.
  - Check the date stamps - if the date is the date you ran the code, then the files were, in fact, replaced, but are not any different.
  - Alternatively, before you run the code, but after you have written the preliminary report, delete all the TeX files (do not git commit!), then run the code.
  - If the files are reproduced, but git shows no difference, they are identical.
- If for some reason there is a change in platform, git might see a difference, but it is solely the “whitespace” encoding, which differs between Mac, Linux, and Windows. In that case, proceed as follows.
- `git add *tex` then `git commit` and `push` as usual.
- Navigate to the Bitbucket repository, and select the tab “`Commits`”. Click on the commit you just made.
- You should see a list of files. Click on the first of the `tex` files you want to inspect.
- You might see the following information:



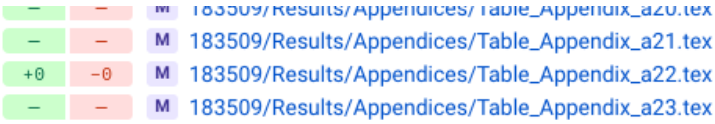
- Click on the “`give it another chance`”. You will now see (or were already seeing) a “git diff”, identifying first the old lines (in red), then the new lines (in green). This list can be long.

- Navigate to the right of the screen. Choose from the dotted menu the item “Ignore whitespace” (you only need to select the “whitespace” option for the first file).



- The page will recalculate the “diffs”. If you see the “`give it another chance`”, click it again. YOu should now, if the files are truly the same, see the following:

- Do this for all `tex` files (you only need to select the “whitespace” option for the first file). If you scroll up to the summary again, you should see a count next to the file. This shows the number of lines changed, taking into account the “whitespace” option.



Cleaning up

- Clean up the `REPLICATION.md` - it should be factual, objective, and not written in the first person.
- Copy-and-paste from the `code-check.xlsx`, including the column “Reproduced?” and any notes column, into the “Findings” part. Use the Excel-to-Markdown plugin for VSCode.
- Delete all of the instructional lines in `REPLICATION.md` before finishing the report.

⚠ **Commit this part of the report to the Bitbucket repository.**

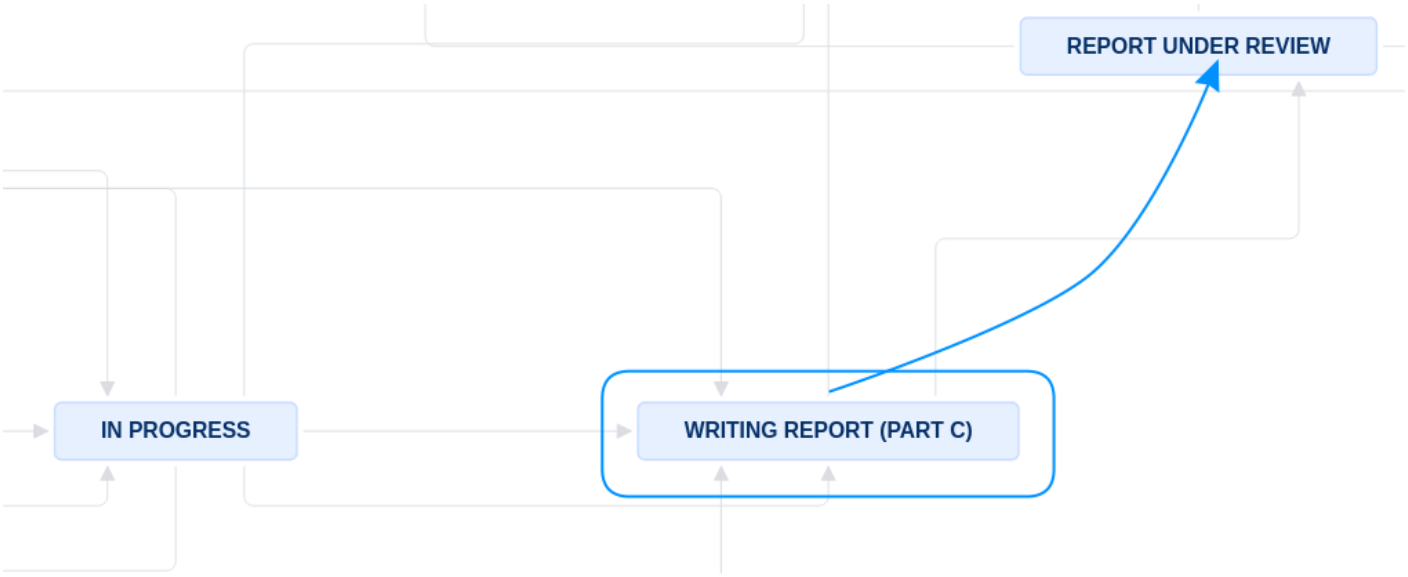
```
git add REPLICATION.md code-check.xlsx
git commit -m "Completed writing report"
git push
```

On Jira

- Check that the `DataCitationSummary` field is filled out indicating how many data citations are in order: all, some, or none.
- Check that the `Report Location` is filled out.
  - If it is, do **not** change it.

- If for some reason, it is missing (very rare!), enter the full URL from the repository, e.g. `https://bitbucket.org/aeaverification/aearep-123/src/master/REPLICATION.md`

You can now submit your report for review by changing the status to `Under Review`



Report completed

You have now completed the report. It will be reviewed by a **Pre-Approver**, who will check that the report is complete, and get in touch for any missing information. In some rare cases, it may mean that you need to go back, and run parts of the code, or review something you missed in the article or README. No worries, that's all fine!

Next step

Start the next case!

AEA: Revision reports after author resubmission

Some pre-publication reproducibility checks require revisions from the authors. We will try to assign these revisions to the replicator for the original submission whenever possible.

Generic Guidance

- Most revisions will not take you much time, so please try to process them quickly!
- Revisions do not require you to repeat all of the same steps as the original replication (see below).

> **Do not create a new Bitbucket repo.**
>
> You will overwrite the original repo. The original contents will still be available – this is why we use version control software like Bitbucket!

- The original REPLICATION.md is a contract; if the authors fix what we ask them to fix, then they have completed their part of the agreement.
  - If new issues turn up as a result of additional materials provided, these are okay to include as [REQUIRED] changes in the revised report.
  - If you are unsure about something, add it to the report. While reviewing your report, we can make a determination about whether or not it can be done after acceptance of the manuscript, be a suggested instead of required change, etc.
- In particular, **do NOT create "update" or "new" directories** The current state of the repository should always correspond to the author's structure. Overwrite files, delete files. The previous state is preserved in Git. This will also tell you what files have changed.
- Please be sure to have the committed "REPLICATION.md" be accurate when you commit it - do not let it contain holdover data from a previous replication attempt, as this can lead to confusion.

Revision Workflow

You should proceed through the workflow as you would for an original case with some exceptions:

- You **should not** create a new Bitbucket repository.
- You *may* not need to re-run any code.
- All sections are potentially subject to updates, so pay attention.

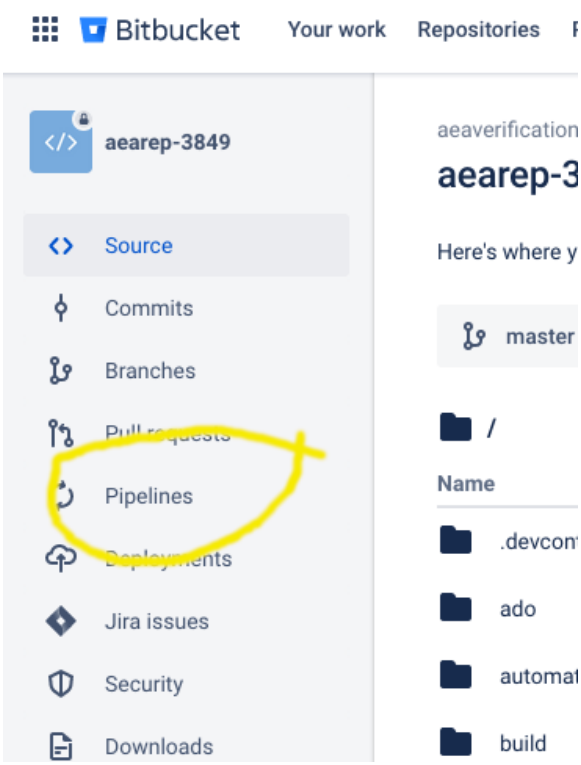
In Progress

First, advance the ticket from `Assigned` to `In Progress`.

- [ ] Identify previous issue.
  - You should see the previous issue listed under **Linked Issues** on JIRA.
  - Use the previous issue, to fill in the following JIRA fields (if not already filled):
    - `Replication package URL`
    - `Journal`
    - `Empirical Article`
    - `Bitbucket short name`. This should be the name of the original JIRA issue (e.g. `aearep-123)`.
    - `openICPSR project number`

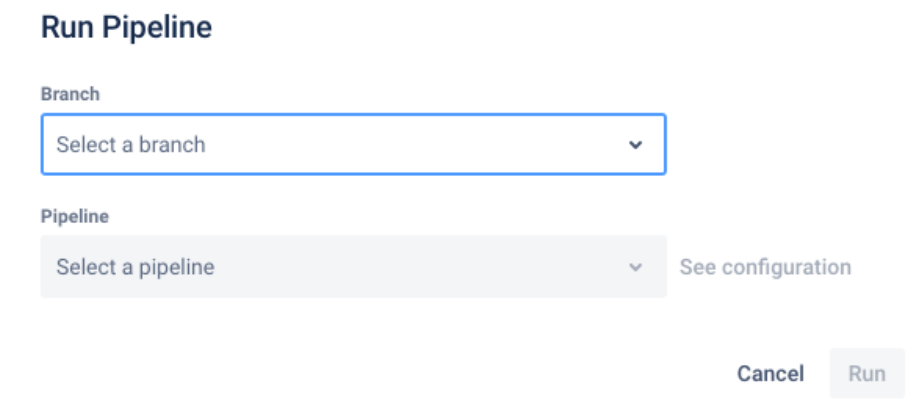
Updating pipeline code

First, you will want to update the tools in this repository. Navigate to the `Pipelines` tab

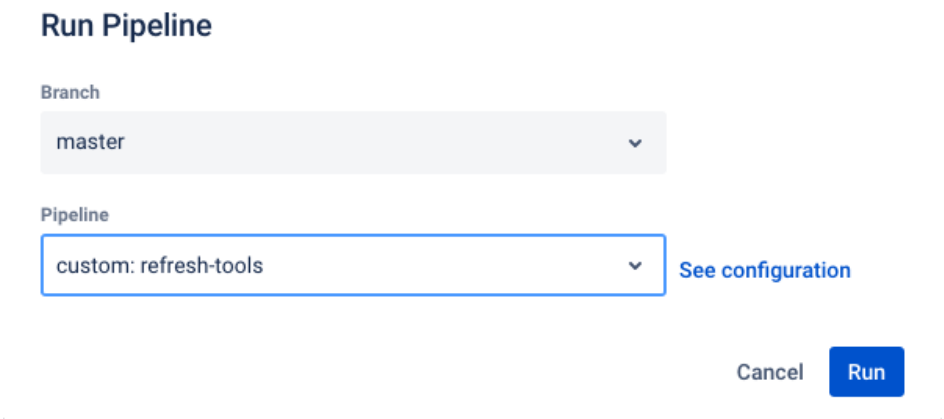


If you do not see pipelines run in the past, you will need to do this manually later.

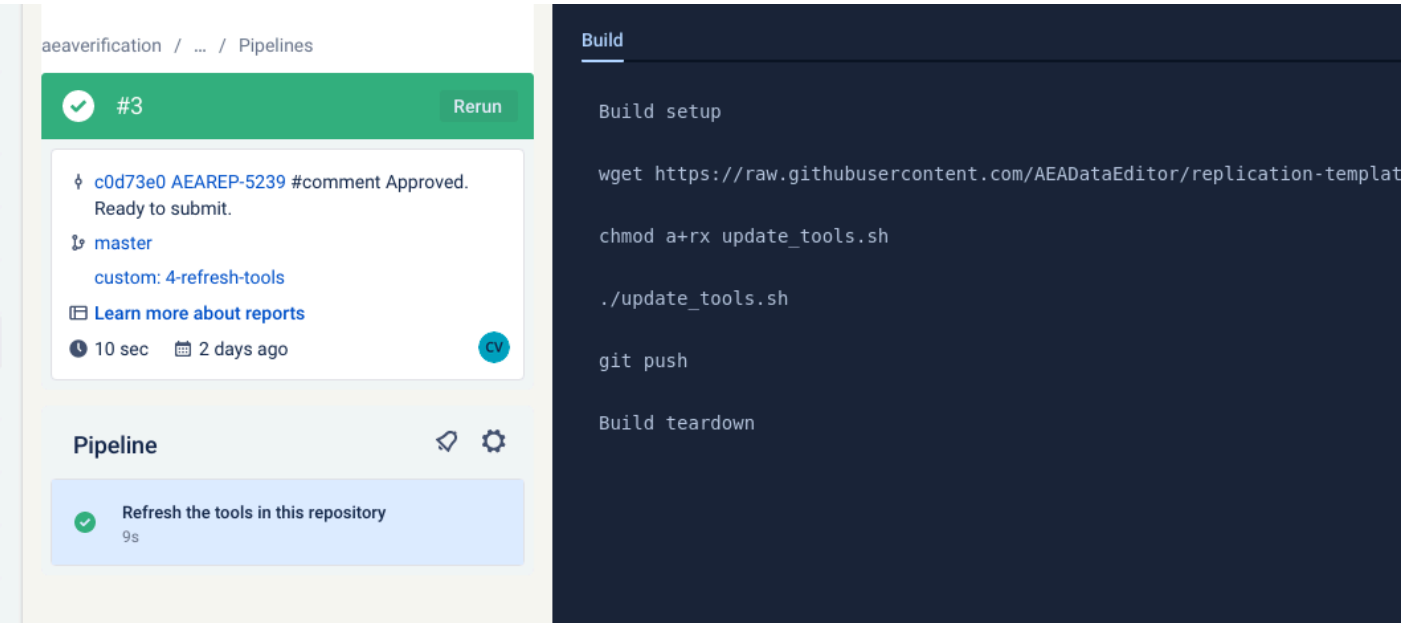
- You will now need to select a “pipeline” to run.



- Choose “`refresh-tools`”. Hit “Run”.



- Your pipeline will start, working through various steps, usually quite quickly.
- [ ] Once your pipeline is done, check that it is green.



Note

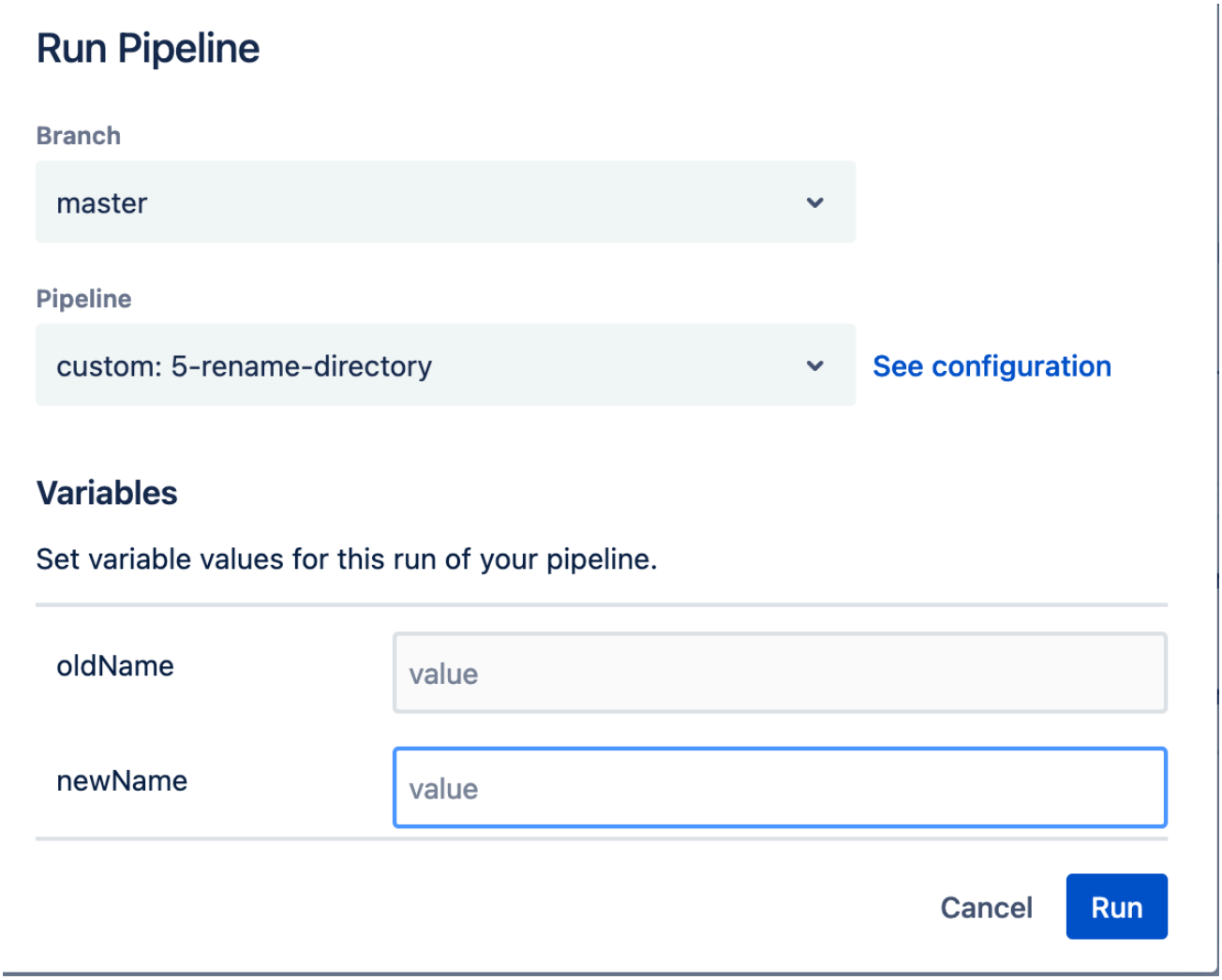
If for some reason, it fails, the logs are available for your supervisor to inspect, and to help you. You may then need to do this manually later.

It will update the code in place. You don’t have to do anything except wait! Once that is done, go to the next step.

Updating Bitbucket repository with latest code and documents

If the authors have submitted a DIFFERENT openICPSR deposit… Click to hide

… then you will need to rename the directory containing the author code in the repository. This is conveniently done using the `5-rename-directory` pipeline:



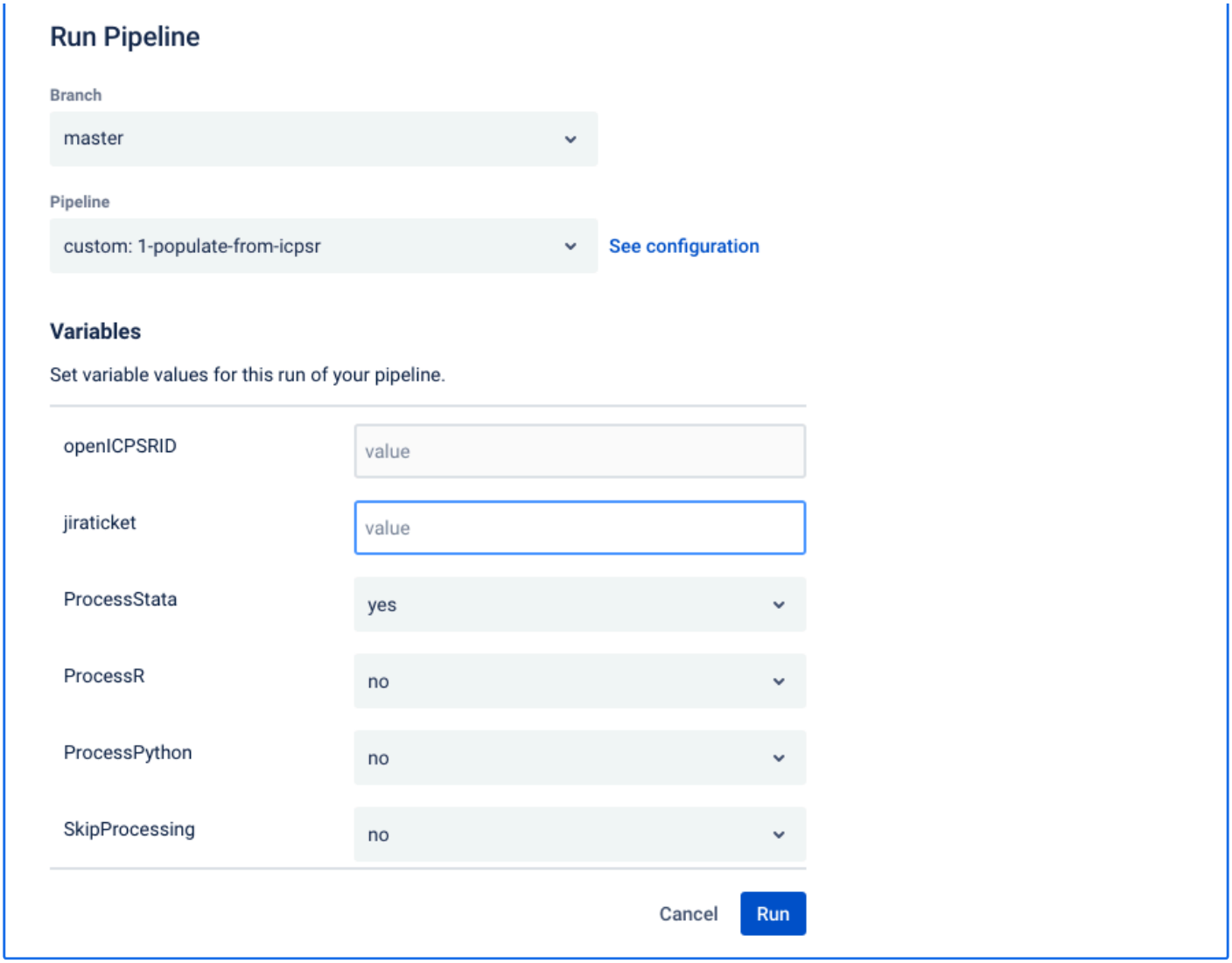
where the fields should be straightforward to fill in.

If for some reason you cannot run the pipeline, you will need to do this manually. `git mv` the directory containing the old deposit code to the new deposit code. E.g.

```
git mv 12345 67890
```

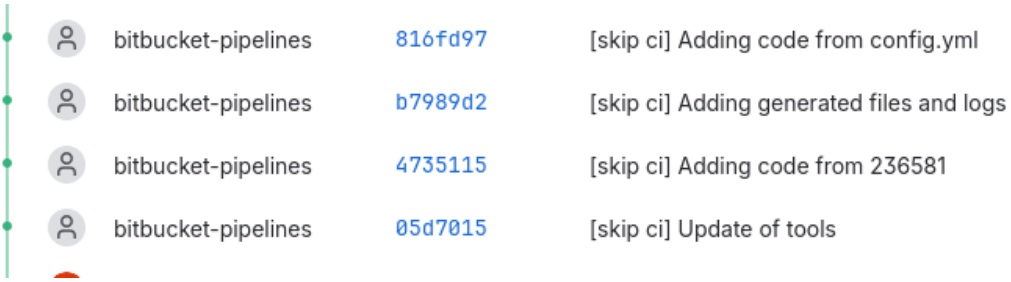
Bitbucket Pipeline | **Manually**

Next, run the next pipeline. This should be the same one that is described in Ingesting Author Materials.



Check that both pipelines have run successfully

The `Commit` tab of the Bitbucket repository should show at least two recent commits, one for updating the tools, and one for updating the openICPSR deposit, but typically more:



Prepare the working area

Similar to the original workflow, you will need to decide where you will run code and do other manipulations. If you are the original replicator, and this is YOUR revision, this will be the **same** computer where you originally did the verification. If not, refer to the general guidance on how to choose and set up your working area.

Please note:

If you are **returning** to the previous working area, be sure to run the same basic setup as initially

```
git pull
```

```
python tools/download_openicpsr-private.py 111234
```

```
unzip -n 111234 -d 111234
```

in the working area before doing anything else!

☠️ Danger Zone! ☠️ Click to hide

It sometimes happens that after the `git pull`, there are still data in the `111234` folder. If authors have updated data files with the same names, then the `unzip -n` will NOT replace them. If the three commands above are the FIRST thing you are doing after the running the Bitbucket pipelines, it should be **SAFE** to do the following:

```
rm -r 111234
```

```
unzip 111234 -d 111234
```

This will delete ALL files in the `111234` folder, so be sure you have committed any changes you made before doing this, and never do this if you do not know what you are doing.

You should **NOT do** this if you read in the README that there are lots of data to manually download from elsewhere!

Updating other materials

- [ ] Download the materials attached to the JIRA issue. This will typically include
  - an updated copy of the manuscript,
  - a response to the editor addressing the requested changes from the prior replication attempt.

> **Warning**
>
> Occassionally, the email received from Manuscript Central will strip out the manuscript because it is too big. You will notice this because
>
> - There will be no `PDF_Proof.pdf` attached!
> - There is text that "the application was unable to attach manuscript files to this email, because one or more of the files exceeded the allowable attachment size (6MB). **
>
> In this case, immediately contact the (assistant) Data Editor to obtain the correct manuscript! Do not use any other manuscript or working paper you might find online.

- [ ] Remove obsolete files. In the root, this should be obvious (old manuscript).
- [ ] Add these to the root of the repository locally, and then `git add`, `git commit`, and `git push` them to the Bitbucket repository (e.g., `git add PDF_Proof.pdf readme.pdf reply_to_editor.pdf`)
  - The root of the repository should contain only our files (i.e., REPLICATION.md, etc.) and the manuscript files (main manuscript, any online appendices and README files provided through the JIRA ticket). Example:

```
code-check.xlsx
config.do
PDF_Proof.PDF
PII_stata_scan.do
readme.pdf
REPLICATION.md
reply_to_editor.pdf
```

- [ ] Open `REPLICATION.md` to review the requested changes from the prior replication attempt. Read the reply to the editor, if provided, to get a sense of what changes the authors made. Make a determination if the revision requires re-running code.
  - If there were [REQUIRED] changes in the Code Description, Replication Steps, or Findings sections of the original report, you will likely have to re-run code.
  - If the only [REQUIRED] changes were data citations or changes to the README, you should not need to re-run code.

Data

- [ ] Update the Data Description section of the report.
- [ ] Update the Data Checks section of the report, if new data have been provided.
  - Check the `generated` directory for any update summary, most of the mechanical checks will be automatically added to the appendix.
- [ ] Fill out the required JIRA fields:
  - `Working location of the data` or
  - `Reason for non-accessibility`
  - and `Data Provenance`

Code

Bitbucket Pipeline | **Manually updating code**

If you already ran the Bitbucket pipeline, your code is already updated.

- [ ] Update the Code Description section of the report.

Code Review or Verification

Re-running code

- You **do not** need to re-verify tables/figures that were successfully replicated the first time *unless* the new code also affects them.
- If parts of the code take a long time to run and the [REQUIRED] changes do not involve those parts, you do not need to re-run that part of the code.
- **Be careful!** Sometimes the code will produce intermediate outputs along the way that may be needed later on. So if you skip parts of the code and cannot replicate the result(s), go back and check to see if you actually need to run all of the code.
- [ ] Run relevant parts of code, including the config.do generating system.
- [ ] Update the Replication Steps section of the report, accordingly.
- [ ] Update the `code-check.xlsx` file.
- [ ] Update the Findings section of the report.
- [ ] `git add`, `git commit`, and `git push` any new results to the Bitbucket repository.

> Change the status from `In Progress` to `Writing Report`.

Writing Report

In this stage, you will finalize the revision report.

- [ ] Ensure all `[REQUIRED]` and `[SUGGESTED]` items from the original report have been changed to `[We REQUESTED]` and `[We SUGGESTED]` in the body of the report.
  - The resolution (or lack thereof) should be included as a bullet point directly below the request.
  - If using the scripts, you can use the `aearevision` to change all the tags
  - If the resolution is incomplete (`Not done` or `Partially done`), then repeat the usual `> [REQUIRED]` tag, as this will become an action item.
- [ ] Ensure any new issues found are tagged with the appropriate `> [REQUIRED]` and `> [SUGGESTED]` tags.
- [ ] Delete/modify any comments of the report template that are no longer true:
  - e.g. if the previous report stated “Data not cited” and the author has now added citations, then that part should state “Data is cited” or “Data is now cited”.
- [ ] Delete any parts of the report template that are no longer relevant:

    - e.g. if no code changes were `[REQUIRED]`, then delete the sections involving code, replication steps, and findings.
    - careful however here, too: authors should only be making changes to parts that we requested changes for, but if they change things elsewhere, then you should, in fact, retain that section, and accurately describe it again. New errors *can* be introduced!
- ☐ Update the Classification.
    - If you do not need to re-run code for the revision, keep the original classification.
    - If you do need to re-run code for the revision, update the classification appropriately.
- ☐ Incorporate all old/new requested changes and resolutions in the SUMMARY section of the report:
    - Create a new sub-section `### Previously` and collect the **Unresolved**, and **Resolved** issues.
        - Leave the existing `### Action Items (manuscript)` and `### Action Items (openICPSR)` sections!
    - Collect all the `> [We REQUESTED]` and `> [We SUGGESTED]` items from the rest of the report (this should correspond to the previous - [REQUIRED]`and`- [SUGGESTED]` items)
        - These should go under the `### Previously` section, appropriately classified into `Unresolved` and `Resolved`.
        - Do not create **separate** `Previously` sections for Manuscript and openICPSR sections!
        - Below each item, include the same resolution you listed in the body of the report.
    - New and any unresolved issues should be collected with their `- [REQUIRED]` and `- [SUGGESTED]` tags under the appropriate `### Action Items` sections.
        - If using the scripts, you can use the `aea-parse-tags` as usual.

Some notes

Please be clear when writing the revision report. The report should make sense without having to refer to the previous report.

- The body of the report should reflect the current status of the deposit.
    - Example: if authors were missing a setup program before and now provide it, the `Code Description` section of the REPLICATION.md should be updated to reflect the inclusion of this program.
- Replace all [REQUIRED] and [SUGGESTED] items with [We REQUESTED] and [We SUGGESTED], respectively.
    - Note: in the summary, these tags are using bullets (`- [REQUIRED]`) - those should be changed to “quotes”: `> [We REQUESTED]`
- Below each such tag, add a bullet point. Start the paragraph with “Done” if the issue was resolved, or “Not done” if not. Then explain what was done.

An example:

```
> [REQUIRED] Please add a setup program that installs all packages as noted above.
  Please specify all necessary commands. An example of a setup file can be found at
  https://github.com/gslab-econ/template/blob/master/config/config_stata.do
```

becomes

```
> [We REQUESTED] Please add a setup program that installs all packages as noted above.
Please specify all necessary commands. An example of a setup file can be found at
https://github.com/gslab-econ/template/blob/master/config/config_stata.do

- Done. A setup program has been added to the deposit, which installs all necessary
packages.
```

An example

In the example below, the revision found bugs in the code that were not previously present (a **new action item**), and identified the continued lack of data citations (an **unresolved** action item). A setup program that was requested was provided by the authors, and is thus **resolved**.

```
SUMMARY
-------

### Action Items (openICPSR)

- [REQUIRED] Please provide debugged code, addressing the issues identified in
  this report.
- [REQUIRED] Please add data citations to the article.
  Guidance on how to cite data is provided in the
  [AEA Sample References](https://www.aeaweb.org/journals/policies/sample-references)
  and in [additional guidance](https://social-science-data-editors.github.io/guidance/addtl-data-citatior

### Previously

#### Unresolved

> [We REQUESTED] Please add data citations to the article.
  Guidance on how to cite data is provided in the
  [AEA Sample References](https://www.aeaweb.org/journals/policies/sample-references)
  and in [additional guidance](https://social-science-data-editors.github.io/guidance/addtl-data-citatior

- Not done. Please add data citations to the manuscript for the
  following data sources: data source 1, data source 2.

#### Resolved

> [We REQUESTED] Please add a setup program that installs all
  packages as noted above. Please specify all necessary commands.
  An example of a setup file can be found at
  [https://github.com/gslab-econ/template/blob/master/config/config_stata.do](https://github.com/gslab-ec

- Done. A setup program has been added to the deposit, which
  installs all necessary packages.
```

Finally, don't forget to `git add`, `git commit`, and `git push` the new report. Then, change the status in JIRA from `Writing Report` to `Report Under Review`.

Publication

Once all review rounds have been completed, the last revision will lead to a recommendation of "Accepted". The Data Editor's staff prepares the openICPSR deposit for final publication. In general, this means that a note is added to the "Project Communications Log" on openICPSR, denoting the acceptance of the deposit. The AEA publication staff can subsequently move this issue forward to "Published" when the supplement has been published on openICPSR.

- The field `openICPSRDOI` is pre-filled, but should be checked by the AEA publication staff.

See Preparing for publication for details.

AEA-related meetings

The team meets twice a week, on Mondays and Thursdays.

> **Note**
>
> All times are Eastern time zone.

Monday meetings

> **Regular Monday meetings are "all-hands-on-deck" meetings.**
>
> **5:30 PM - 6:30 PM** on Zoom.

Team members will have received a calendar invite with the Zoom link.

All team members are expected to attend, any absence should be motivated by email, with a status report on cases being worked on.

Thursday meetings

> **Thursday meetings are "check-in" meetings.**
>
> Meetings happen in two time blocks:
>
> - 8:00 AM - 8:30 AM (Green)
> - 11:00 AM - 11:30 AM (Blue)

> **Note**
>
> Participation is by "call-in" (video not required) using Zoom. Team members give a very brief update, signal any issues, and sign off. The typical status report is less than 1 minute.

When there are too many people in a particular time slot, we may be more precise about the time:

Split Thursday meetings

Meetings happen in four time blocks:

- 8:00 AM - 8:10 AM (Green)
- 8:10 AM - 8:20 AM (Turquoise)
- 8:20 AM - 8:30 AM (Purple)
- 11:00 AM - 11:30 AM (Blue)

Checking Unassigned Jira Tickets

Setup

Permissions… Click to hide

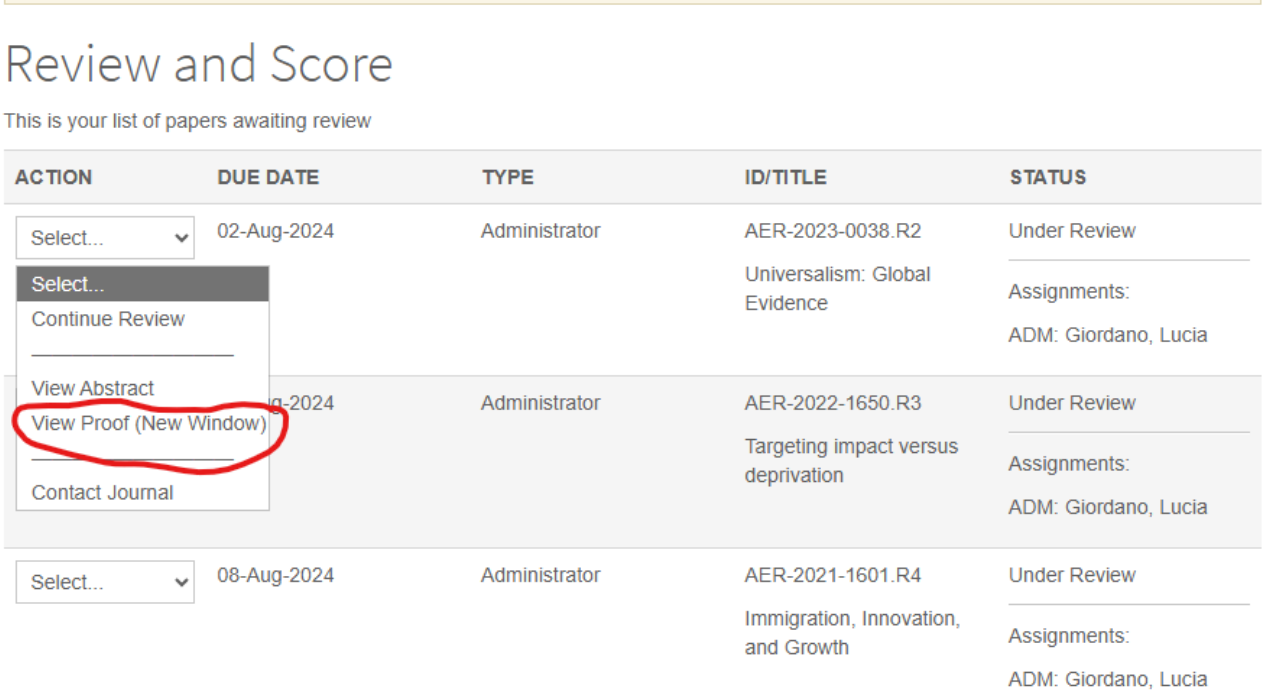
- ☐ Assigner needs password to ScholarOne (**Data Editor** shares via LastPass)
  - Then you'll need to install the web browser extension for whichever browser you use (Chrome, Firefox, etc).
- ☐ Assigner needs permissions on openICPSR (Data Editor or **Assistant** requests from openICPSR)

Basic Instructions

- Go to the unassigned filter in Jira
  - Easy way: click on this link
  - Longer way: `Dashboards` > `Admin Dashboard` > `AEA tasks by Assignee` section > scroll down to "Unassigned".
- Click on the `Key` for a case to start the process.
- The Jira issue should have two documents attached:
  - The manuscript ("PDF_Proof.pdf")f
  - The DCAF (should never be missing…)

If the manuscript is missing… Click to hide

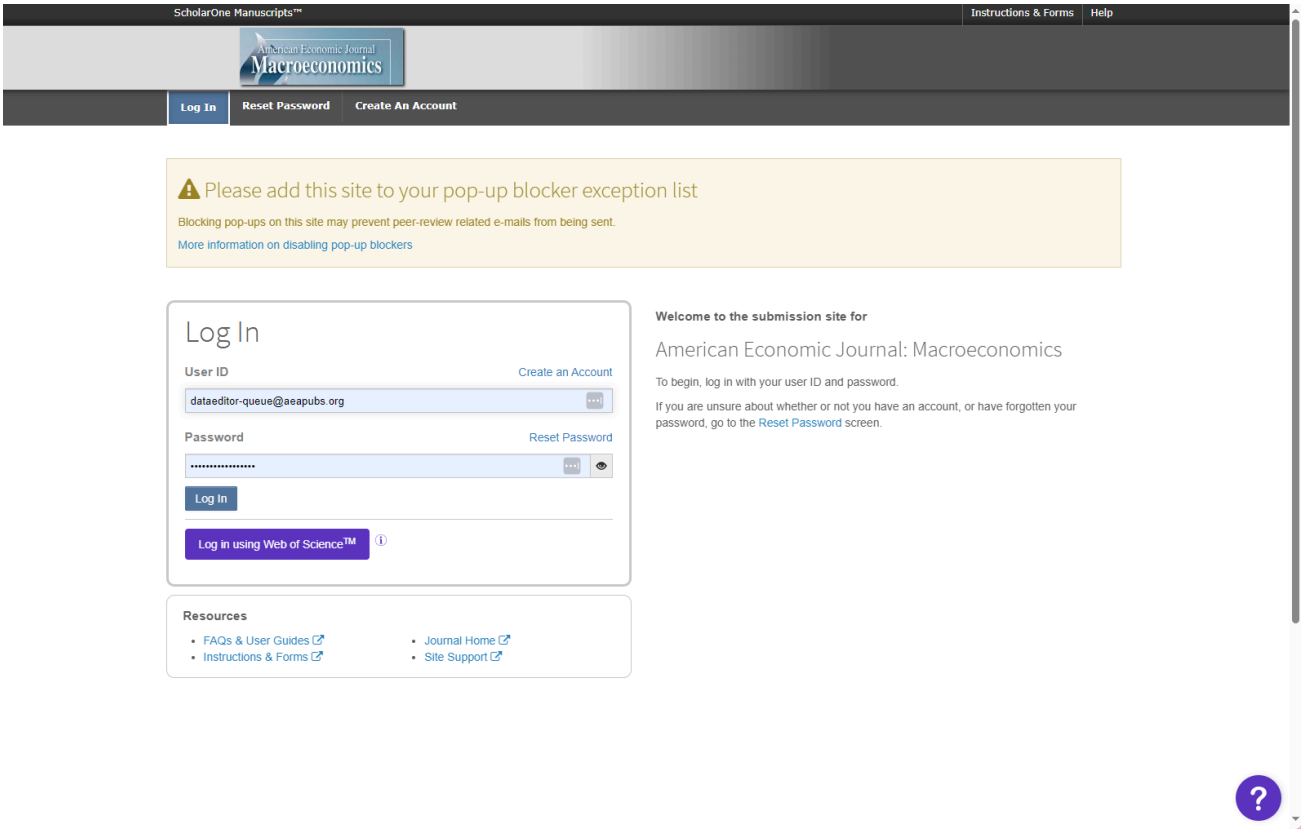
- Make a note of the `Manuscript Central Identifier` (copy it to your clipboard).
- Click on the `Other links` tab
- Click on the `MCEntryURL` link
- Enter the ScholarOne/Manuscript Central login details provided to you.
- Click “Review” and search for the Manuscript Number (Ctrl+F/Command-F).
- Once you find the manuscript, go to the Action section, “View Proof”, and download the PDF of the manuscript.



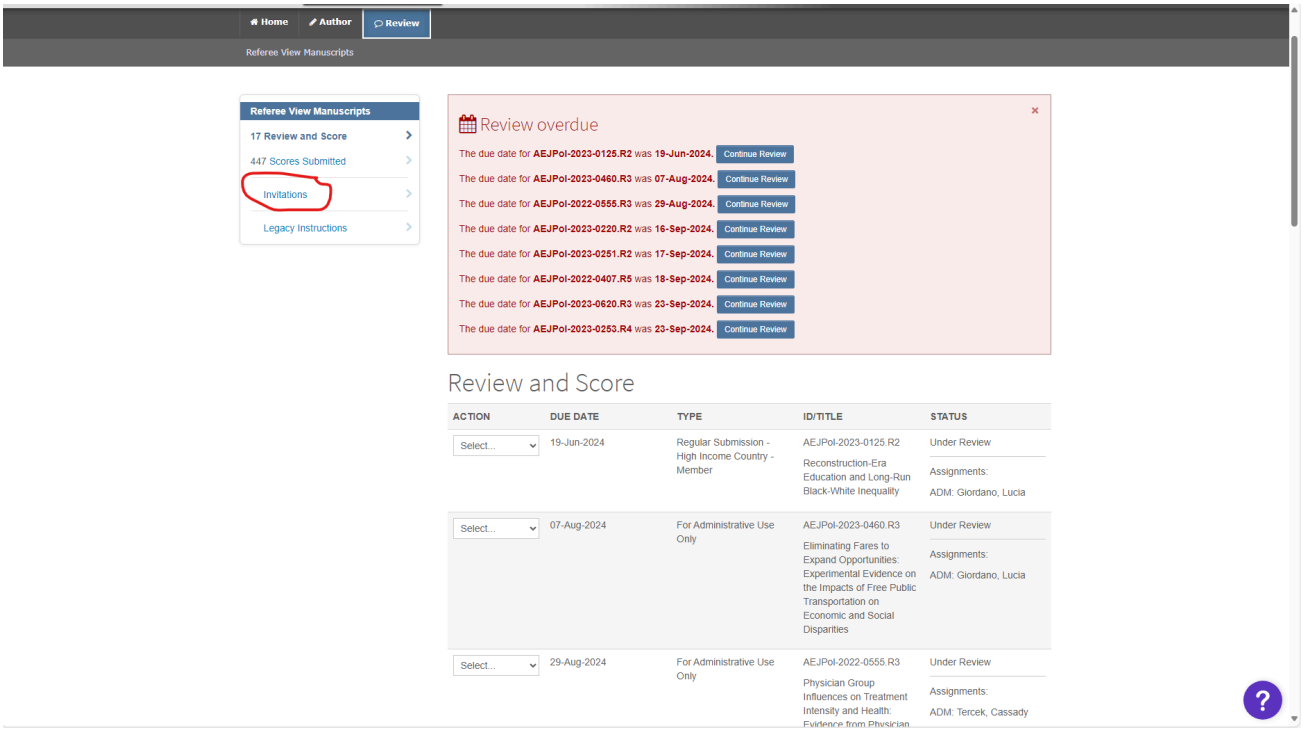
- Attach the PDF to the Jira ticket.

If the ‘MC Status’ field is RR (“Invitation to Review”)… Click to hide

- Go to the MCEntryURL field in the “Other Links” section
- Log in to ScholarOne and click Review.



- Navigate to the link that says “Invitations”



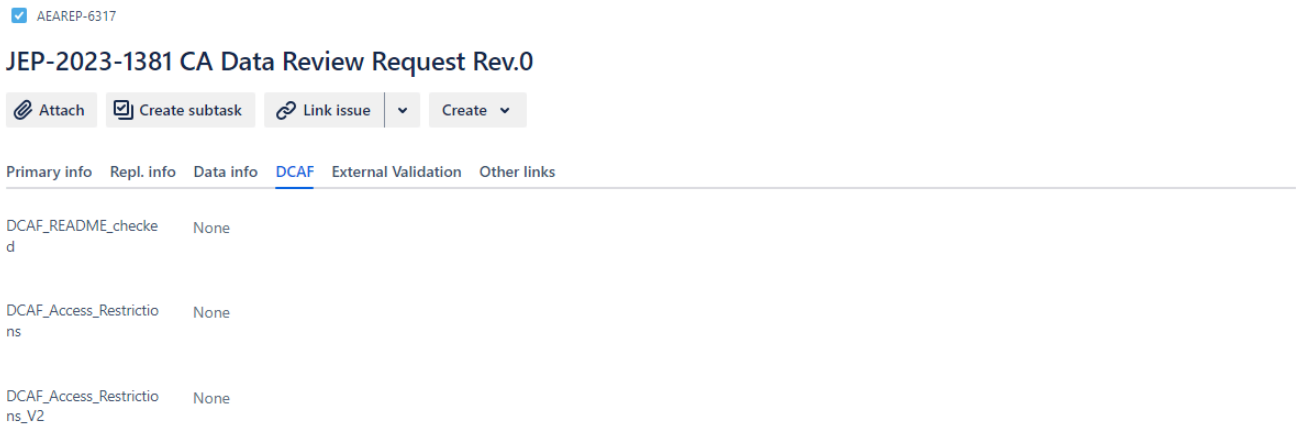
- Search for the manuscript number (Ctrl+F/Command-F).

> - Click “Agreed and begin review”
> - Then close out the tab.

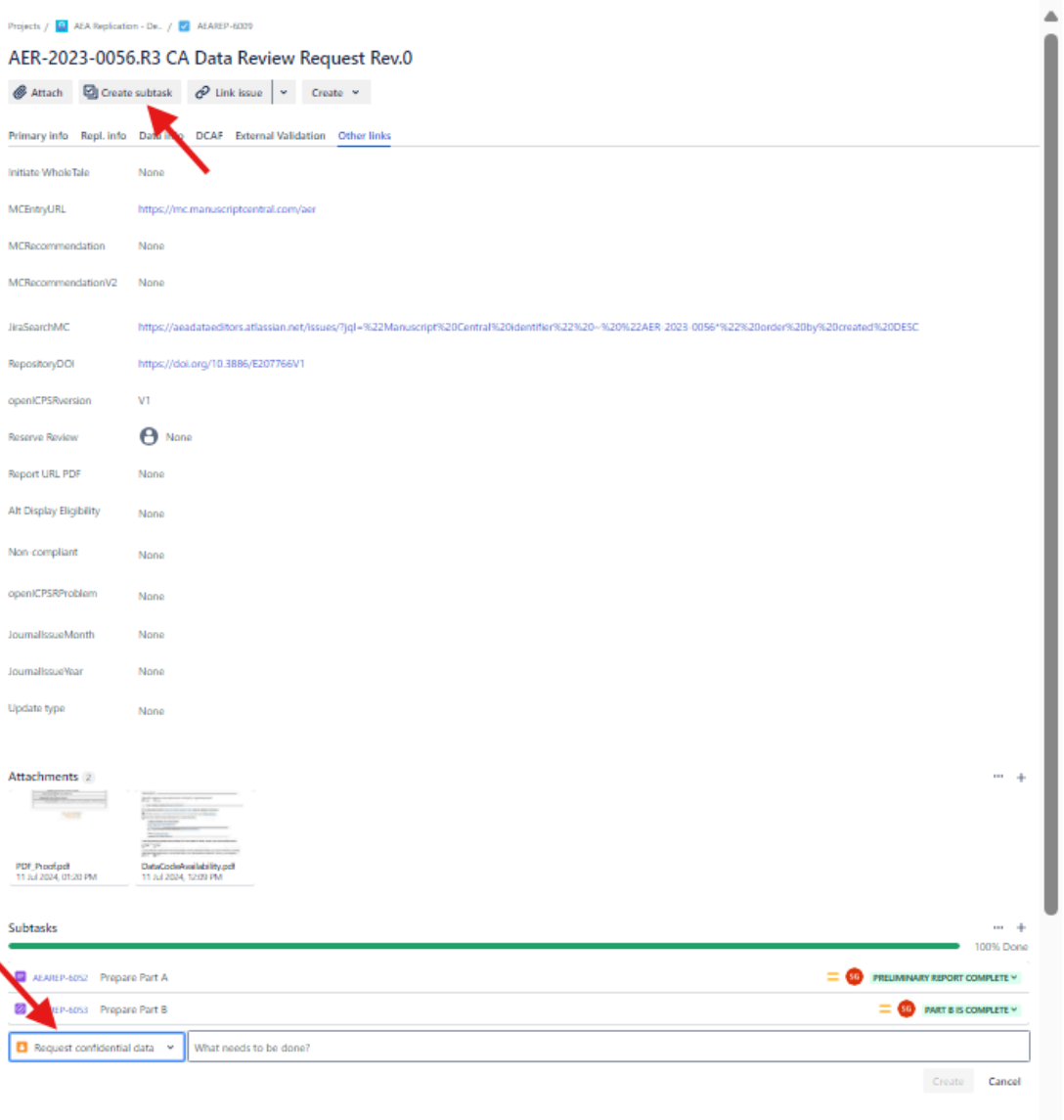
- Open up the DCAF (you can do this within Jira, without downloading).

In the most recent cases, fill out the following info from the DCAF attached:

- Fill out the `openICPSR Project Number` - this should be JUST the pure number part, not `openicpsr-123456`!
- If a different DOI is there (does not include `10.3886`), copy the full DOI into the field `Replication package URL`!
- Review the part with “Is any of the data used in this manuscript….”. This will be encoded in the field `DCAF_Access_Restrictions_V2` (which is on the `DCAF` tab in Jira)



- if `Yes, data can be made available privately` is checked:
  - Create a subtask of type `Request additional data`, with subject `Request Restricted Access Data for (AEAREP-NUMBER)`



- Tag the Assistant Data Editor in the newly created subtask comment.
- The Assistant Data Editor will do additional steps

Now you need to open the draft replication deposit (typically, but not always on openICPSR)

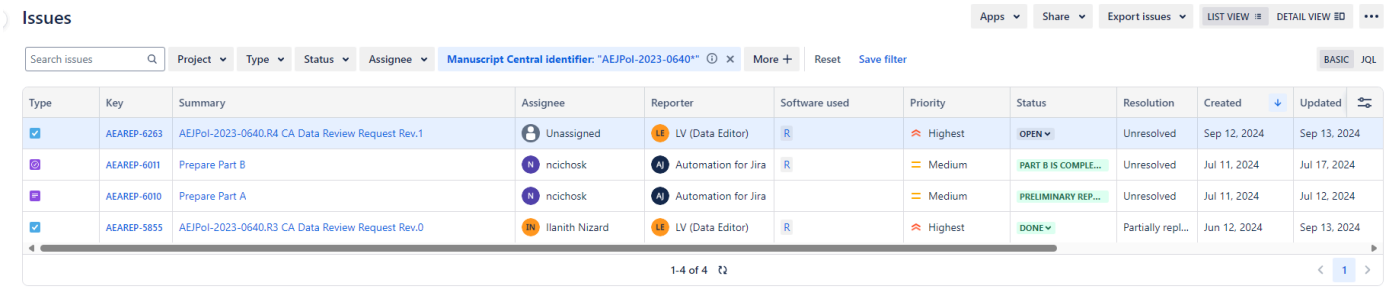
- Click on the `Replication Package URL`, which will open up a separate tab.
- Find the README in the deposit.
- Click on the README (you can preview, probably do not need to download) and scan through to identify the software used
- Go to the tab `Repl info` and add the software identified to the `Software Used` field.

If there is no README… Click to hide

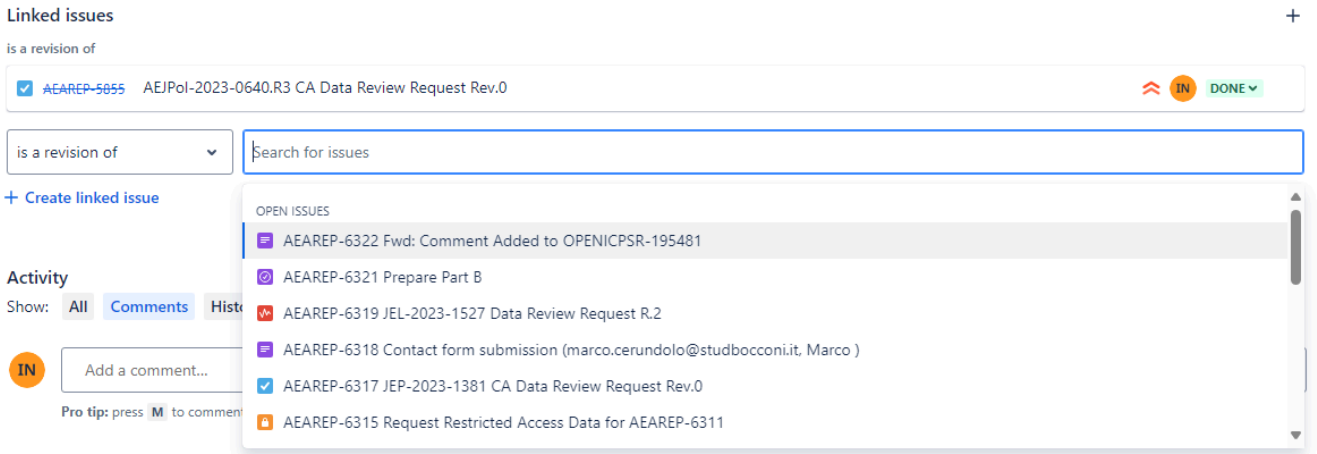
If there is no README, alert the Assistant Data Editor, who will create a short report and send this case back to the authors.

Check if the case is a revision:

Check if the case is a Revision by going to the Other Links tab and clicking the JiraSearchMC. If there are other Jira issues (not subtasks or FYIs), then it is a Revision.



- Link the Revision: Go back to the main Task you were editing, click “Link issue” > “Is a Revision of” and then input the AEAREP number of the older task.



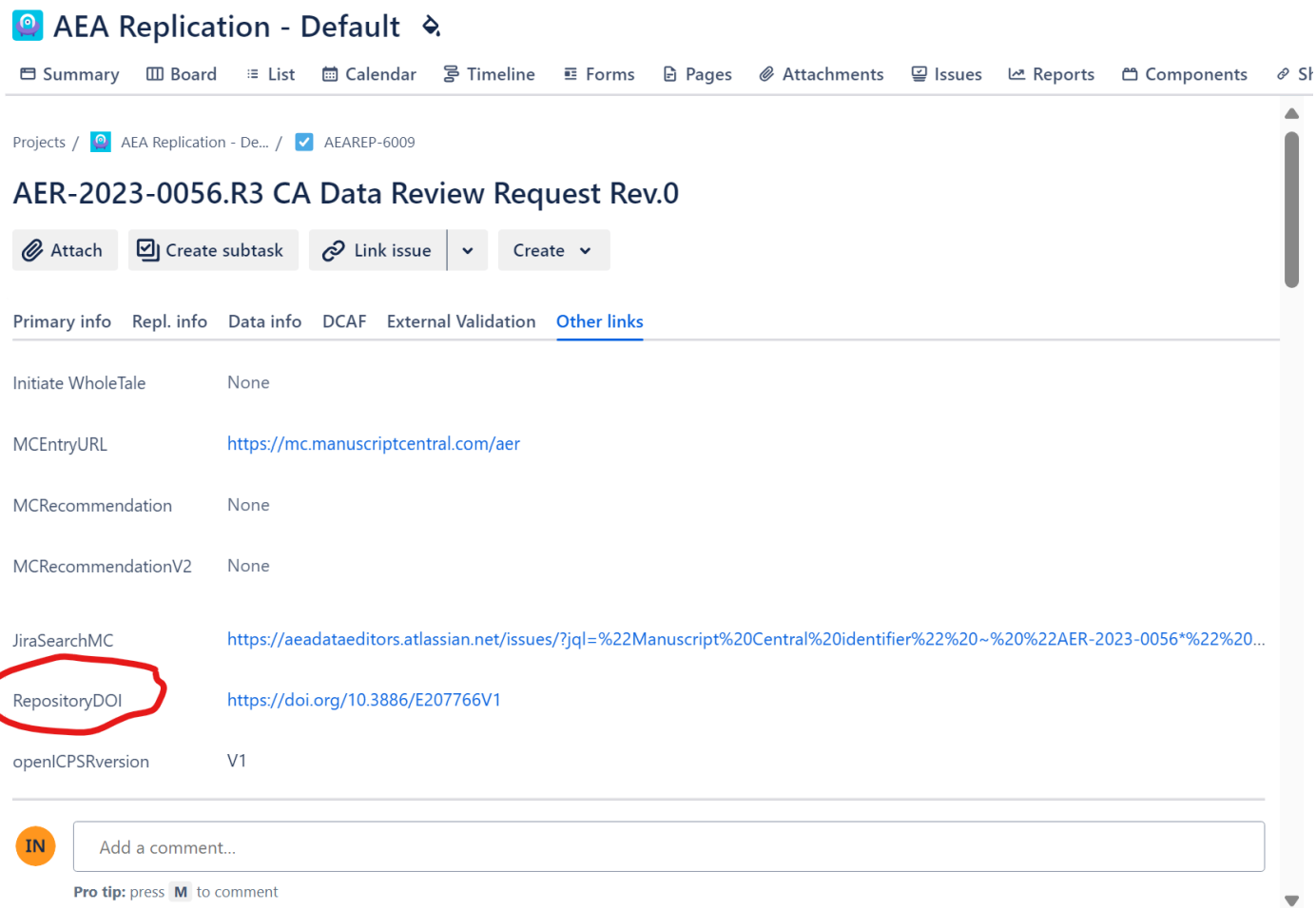
- Update the MCStatus field to include ‘Revision’ if it doesn’t already have it.
- Update the bitbucket short name with the repository name of the previous cases.
- Check if the older Jira ticket had restricted access data (i.e. Working location of restricted data was filled out). If yes:
  - Link Issue, select type “relates to” and add the aearep-xxx for the subtask “Request Restricted Access Data for AEAREP-nnn”
  - Fill out Working location of restricted data with the same L drive path
  - Fill out Agreement Signed to match the older Jira ticket

Non-standard unassigned cases

When you get a Jira ticket titled `"Contact form submission (researcher@university.edu, FirstName LastName)"`, often these are in reference to openICPSR deposit that have already been published. The message is from someone reviewing the deposit and found something concerning, like a missing code file or potential PII.

- Review the message in the `Primary Info Section`.
- Using the DOI or article titled provided, search for the Manuscript Identifier.
  - You can find the article titled on the manuscript attached.

- You can find the DOI under the “Other Links” tab.



- Fill out the `Journal` field and the `Manuscript Central Identifer` field.
- Mark the `MCStatus` as Update and fill out the `Update Type` (under Other Links) as the appropriate label, depending on who is contacting us.
- Tag the Assistant Data Editor in the ticket comments.
- The Assistant Data Editor will contact the authors.

AEA: Reviewing and (Pre-) Approving Reports

“Approvers” and “Pre-approvers” will review the reports, and finalize the Summary.

Generic Guidance

- Pre-approvals usually can be completed rather quickly and should be prioritized.
- The pre-approver should not be running any of the code or downloading any of the data unless asked to do so.
- The language for the `[REQUIRED]` tag comments is drawn from the Sample Language for Reports. To ensure uniformity across reports, pre-approvers are encouraged to use the phrases verbatim.

Preliminary Steps for Pre-approving Reports

> **The "Review for pre-approval" transition in Jira requires specific permissions.**
>
> If you have not been granted this permission, please reach out to your supervisor.

1. Clone the Bitbucket repository of the report that requires pre-approval to the local machine or CISER machine. **Do not download the openICPSR deposit!** The pre-approver checks and finalizes the report but does not (usually) redo the whole replication.
   - You can also use "Github Codespaces" for this process.
2. Skim through the report and note if it is an original replication or a revision. Follow the steps below correspondingly.

Original Replication Report

If this is a `RR` case, check for overlapping `CA` cases! Click to hide

In some cases, we will have worked on a `RR` case (title says "Invitation to Review"), but the editor of the journal may have already issued a `CA`.

- Usually, these cases should be linked - check if there is a `Is Revised By` link.
- If not, double-check by going to the `Other Links` tab in JIRA, and clicking on the `JiraSearchMC` field (is always pre-filled). This will show all cases that are tagged with the same `Manuscript Central Identifier`. Check if one of them is an incomplete or unassigned `CA` case.

If there is an overlapping `CA` case, the next steps depend on what updates the authors may have made:

> Check the openICPSR deposit to see if there have been any updates since the `RR` was originally submitted.

- Often there are no updates. In that case, treat this report as if it were a `CA` report, and then contact the Data Editor.
- If there are no updates, this gets complicated. Contact the Data Editor immediately.

1. Verify that the replicator has deleted all of the `INSTRUCTIONS` comments of the report; if not, please do so.
2. Verify if the replication requires **IRB approval/RCT registration**. These are commonly found in papers that utilize **randomized experiments, lab experiments, or surveys** run by the authors, involving "human participants." Ensure that the manuscript contains the necessary approval information (on the title footnote). If absent, insert relevant `[REQUIRED]` or `[SUGGESTED]` tag comments wherever necessary:
   - Sample tags for IRB issues
   - Sample tags for RCT issues
   - For surveys and experiments, also check if relevant "code" to run the survey or experiment is included. Sample tags for survey/experiment issues
3. Read through the **Data Sources section** of the report and check for completeness.

    - Reports sometimes might be missing data sources. Cross-check (scan) this with the manuscript and the README. This is usually the most time-intensive part of the pre-approval.
    - You are not expected to redo this section, but you should spot-check to see if it is complete and accurate.
    - When replicators mention “URL does not work” but provide no further information, verify the URL (if less than 5 URLs to check)
    - The report should accurately reflect whether the data source has been cited and whether its location/access modality has been provided.
    - Verify/Insert relevant `[REQUIRED]` or `[SUGGESTED]` tag comments.
4. Check the **Data checks section** for completion.
    - Check if the PII scan output is present in the repository (should have been done automatically by the Pipelines)
    - Check if the Package scan output is present in the repository (should have been done automatically by the Pipelines)
    - If these sections are not present in the report, retrieve them from the `/generated` folder. If not present there, remove the placeholder tags.
    - Verify/Insert relevant `[REQUIRED]` or `[SUGGESTED]` tag comments.
5. Check the **Code checks section** for completion.
    - Check if the completed Code check spreadsheet is present in the repository.
    - Verify/ Insert relevant `[REQUIRED]` or `[SUGGESTED]` tag comments.
6. Check the **Stated Requirements** and **Missing requirements** section for completion and insert relevant `[REQUIRED]` or `[SUGGESTED]` tag comments.
7. Carefully read through the **Replication steps section.**
    - Ensure that this report section is coherent in its description of the steps the replicator took to perform the entire replication. Can you follow the narrative/list of steps, and from it, do you think you understand what the replicator did?
    - If the replicator noted that the code contains bugs, the error message/error code should be explicitly listed.
    - **The pre-approver should not be rerunning any of the code!**
    - Communication between the pre-approver and replicator is encouraged so that any confusing/unclear language used in this section can be clarified. Use the Jira comments, and tag the Data Editor as necessary.
    - Verify/Insert relevant `[REQUIRED]` or `[SUGGESTED]` tag comments.
8. Check the **Computing Environment of the Replicator section** for completeness. (It should not include the description of the replicator’s laptop unless it is clear that the replicator actually ran code on the laptop)
9. Verify: Format the results in the **Findings section** into tables by utilizing the the Excel-to-Markdown Extension in Visual Studio.
    - If any of the steps in the **Replication Steps** section involved bugs that were fixed, ensure that the `[REQUIRED]` tag about debugged code is added (often forgotten)
    - Check if the output/log files have been pushed to the Bitbucket repository.
    - If there are discrepancies between the replicated tables and figures and those of the manuscript, the report should contain images/screenshots to highlight the differences.
    - (rare) If there are numerous (too many) tables and figures that exhibit discrepancies, these should be put into a .zip file. This action should be noted in the **Summary section** as well as mentioned in a JIRA comment.
10. Ensure the classification of the replication fits the degree of reproducibility.
11. Fill out the **Action Items** of the report.
    - These are separated into “Manuscript” and “openICPSR” (these sections are pre-existent in the template)
    - Use a script or manually copy-paste all of the `[REQUIRED]` tags to this list.

        - `aea-parse-tags` extracts the `[REQUIRED]` tags and places them at the top of the REPLICATION.md
    - Split the action items according to where they can be addressed. Some may be addressable in both sections (for instance, correcting figures and tables, and/or data citations). Others belong only into the "Manuscript" part (IRB, RCT), others only into the "openICPSR" (anything related to code)
    - Order them by importance: `[REQUIRED]` items first, and within these, the most important at the top (correcting bugs, providing missing files).
    - `[SUGGESTED]` items can be at the end of each section.
        - `aea-parse-tags` extracts the `[SUGGESTED]` tags only if provided with the additional argument `sug`
12. Fill out the **Summary section** of the report.
    - Start with a thank you, and a positive note: Highlight the merits of the replication i.e. how many figures/tables were replicated, if data citations were properly completed, etc.
    - Add a template line with the proposed resolution. These are at the top of the sample-language-report.md.
13. Commit the report.
    - Use standard `git commands`: `git commit -m "Pre-approval of replication report" -a` followed by `git push`
    - Alternatively, use a script: `aeaready [JIRAISSUE] pre nopdf` will create the right commit message, and push as soon as you hit enter. If you are on a computer that can create the PDF (some Linux and some MacOS systems), dropping the `nopdf` part will also create the PDF.
14. Select a recommendation on the `Other links` tab of the JIRA ticket (see Choosing a Recommendation)
    - If this is a `RR`, choose a recommendation in the `MCRecommendation` field.
    - If this is a `CA`, choose a recommendation in the `MCRecommendationV2` field.
15. Advance the JIRA ticket to "Pre-approved." The pre-approval is now **complete!**

External Reproducibility Reports

In cases where an external reproducibility check was completed (e.g., CASCAD, World Bank), the following steps must be taken during pre-approval to ensure proper documentation and integration.

1. **Verify the "External Validation" checkbox is selected** in the JIRA ticket.
2. **Ensure that the external report is included in the repository.** It must be part of the final deposit and not just attached to JIRA or email.
3. **Ensure no section titles from our `REPLICATION.md` template report are deleted.** The internal report must maintain its structure even when referencing external work.
4. In any section of the internal report where the external replicator's work applies, include a pointer such as:

    > "Please refer to the external report below."

5. **Review the external report.** Identify any bugs or discrepancies and ensure they are recorded in the **Summary** and **Findings** sections of the internal `REPLICATION.md`.
6. **Add appropriate `[REQUIRED]` or `[SUGGESTED]` Action Item tags** corresponding to the issues found in the external report.
7. **Generate a PDF of the internal `REPLICATION.md` report** and name it: `REPLICATION_internal.md`.
8. **Append the external report PDF** (as received) to the internal report.

9. **Save the combined file as:** `REPLICATION.md` and convert it to a PDF.
10. **Commit both files to the Git repository:**

- `REPLICATION_internal.md`
- `REPLICATION.pdf`

Important Notes

- **Do not delete the external report.** The combined `REPLICATION.pdf` serves as the official record.
- The external report must be **included at the end of** the final `REPLICATION.pdf`.
- This procedure applies to all external reproducibility checks.

> **Note**
>
> The pre-approver should reach out to the original replicator for clarifications should there be any confusions during the course of pre-approving the report. If bugs in the code seem trivial i.e. missing packages, missing `Results` directory, replicator cannot find output etc., the pre-approver should reach out to the original replicator for further clarification.

> **Warning**
>
> For the majority of the `[REQUIRED]` comment tags, it is best to use them verbatim to preserve uniformity across our reports. However, there are times where a more accurate description can help the author pinpoint what exactly is required of them.
>
> For example, say we have a scenario where the author did not include code for Table A.6 in the code deposit but everything else has been provided and replicates perfectly.
>
> Instead of writing:
>
> > [REQUIRED] Please provide complete code, including for construction of the analysis data from raw data, and for appendix tables and figures, and identify source for inline numbers.
>
> It is clearer to the authors if we write:
>
> > [REQUIRED] Please provide complete code for appendix tables and figures, in particular Table A.6.

Revision Report

The checklist of items to review are roughly the same as above. In addition:

- The pre-approver should check that `[We REQUESTED]` tags are used in place of `[REQUIRED]` tags. These tags should be preceded by a “>” to create a comment rather than “-,” which creates a bullet point.

    - The script `aeareview` changes all the `[REQUIRED]` to `[We REQUESTED]`.
- The report should reflect the **current** state of the replication.
- In the **Summary Section:**
    - Create a new section "`### Previously`" that covers all the action items from the previous round.
    - Create a list of persisting issues that require attention. These should be added as action items to the usual section, in addition to any new items.
    - There is no need to distinguish new from persistent action items: the action item list should simply contain a complete and exhaustive (check)list of items that need to be done. As in an original replication report, all persisting and new tags within the action item list should be either `[REQUIRED]` or `[SUGGESTED]`, not `[WE REQUESTED]` and `[WE SUGGESTED]` (as seen below).

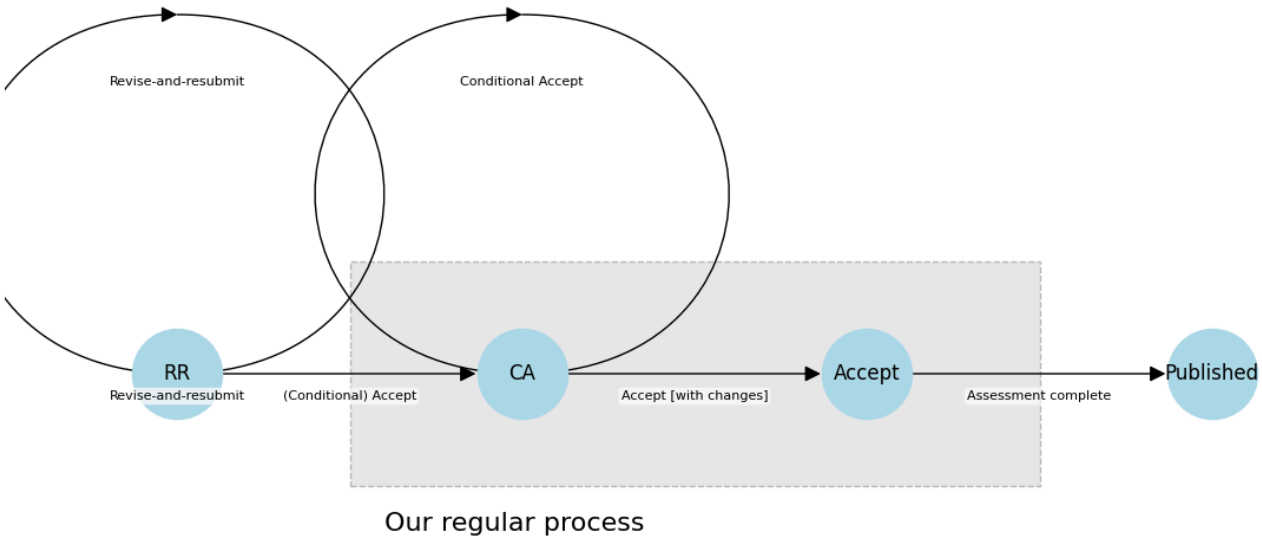
Example:

```
### Action Items (Manuscript)

- [REQUIRED] The data collection reported in this article had IRB approval. Please provide full IRB appro

### Action Items (openICPSR)

...

### Previously

> [We REQUESTED] The data collection reported in this article had IRB approval. Please provide full IRB a

Not done. The manuscript still does not mention the IRB information in the titlepage footnote.
```

Choosing a Recommendation

Graphical overview of flow

▶ **Show code cell source**



Our regular process

Recommendations for `CA` reports

Approvers must select/confirm one of the recommendations (field `MCRecommendationV2`):

- **Accepted** - the manuscript moves forward in the publishing workflow on Manuscript Central, the Data Editor does not see the manuscript again.
- **Accepted with changes** - same, but some conditions may be imposed. However, the Data Editor does not need to see the manuscript again.
- **Revisions requested - manuscript ready** - Some revisions need to be made, and the Data Editor needs to see the authors' response. However, the manuscript can move forward in the publishing workflow. This is rarely used, but opens up the possibility that the managing editors can pull out a manuscript from this category to move forward, depending on the backlog for publication.
- **Conditional Acceptance** - the Data Editor expects to see a response from the authors to the report.
- **Revise and resubmit** - the Data Editor has detected a serious problem which needs to go back to the "Revise and resubmit" phase of the publishing workflow. This is only invoked if there are significant concerns as to the validity of the manuscript's conclusions based on the reproduction attempt. **Rarely used, and only for the most serious cases.**

Recommendations for `RR` reports

Approvers must select/confirm one of the recommendations (field `MCRecommendation`):

- **Accept** - the manuscript moves forward in the publishing workflow on Manuscript Central, the Data Editor does not need to see the manuscript again **during the `RR` phase** of the publishing workflow (we will see it again as a `CA`).
- **Revise and resubmit** - the Data Editor expects to see a response from the authors to the report, and wants to see it again before the Editor can issue a `CA` decision.
- **No recommendation** may be chosen in specific cases. Rare.

- **Reject** - the Data Editor has detected a serious problem. This is only invoked if there are significant concerns as to the validity of the manuscript's conclusions based on the reproduction attempt. **Rarely used, and only for the most serious cases.**
- **By email** kept for technical reasons, should not be chosen.

Publication

Once all review rounds have been completed, the last revision will lead to a recommendation of "Accepted". The Data Editor's staff prepares the openICPSR deposit for final publication. See Preparing for publication for details.

AEA: Submitting information back to the Editorial Office and to authors

> **Note**
>
> The AEA uses ScholarOne (also known as Manuscript Central) as its Journal Management System (JMS), but the guidelines below will work for any email-based system. A more integrated approach is probably preferred.

Basic setup

In the JMS, the Data Editor is set up as a reviewer who can be assigned during the conditional accept stage. Assignment is via email to a pre-configured email address. Once the Data Editor has finalized the replication report, a manual upload is required to submit the report. If the report requires a revision, a note may need to be posted on the openICPSR deposit as well. If the report requires no revisions, a note is made on the openICPSR deposit to confirm that everything is in order.

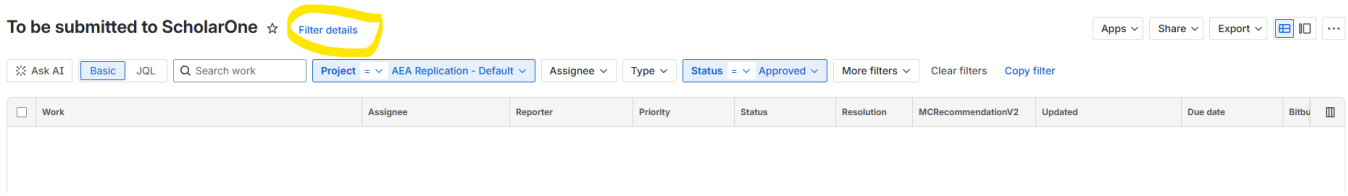
> **Note**
>
> **Permissions**
>
> - ☐ submitters needs password to ScholarOne (**Data Editor** shares via LastPass)
>   - Then you'll need to install the web browser extension for whichever browser you use (Chrome, Firefox, etc).
> - ☐ submitters need permissions on openICPSR (Data Editor or **Assistant** requests from openICPSR)
> - ☐ submitters need `Publisher` permission on Jira (**Assistant** can set in Project -> Settings -> People in Jira)

Daily ScholarOne Subscription Reminder

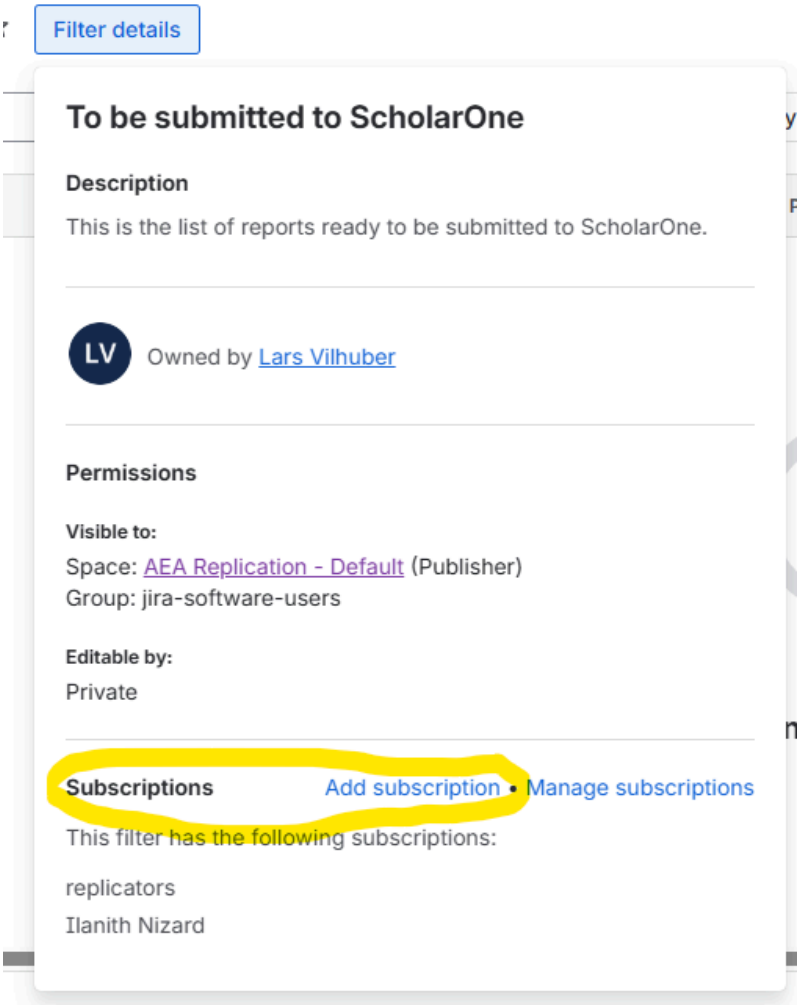
Submitters should receive a daily email reminder titled **"Subscription: To be submitted to ScholarOne"** to ensure reports are submitted on time.

1. Go to the Jira filter for reports **To be submitted to ScholarOne**: https://aeadataeditors.atlassian.net/issues/?filter=10030

2. Click **Filter details**.



3. Scroll down to **Subscriptions**.



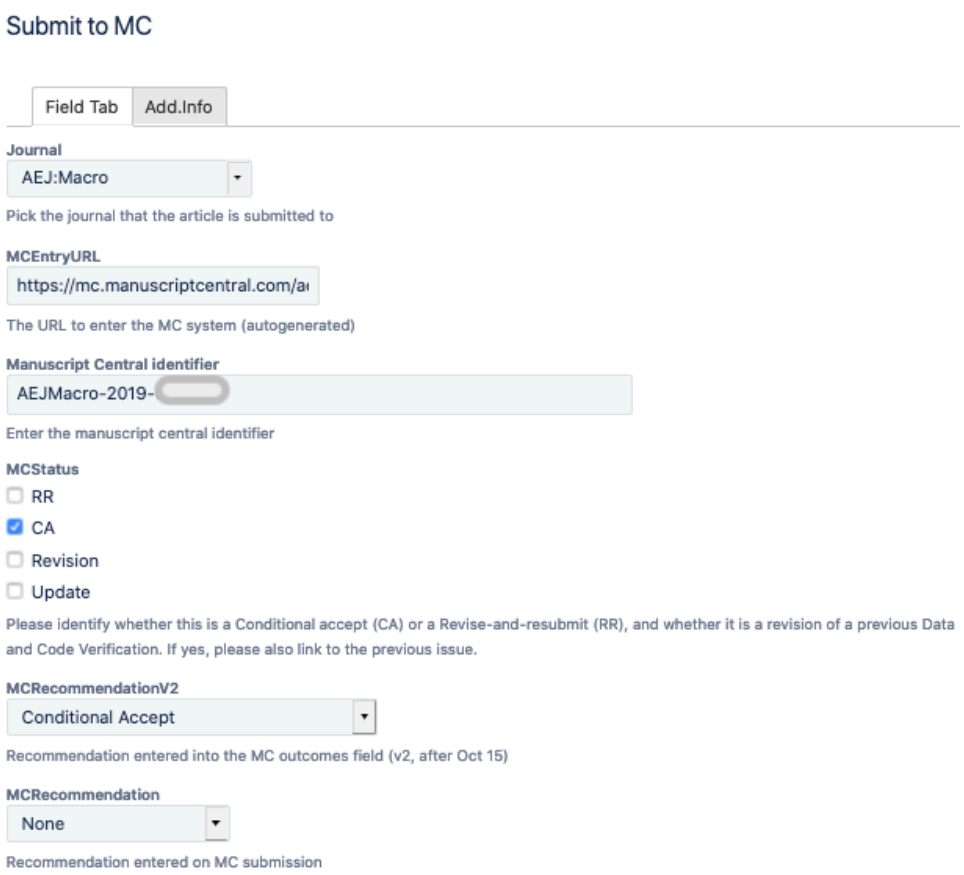
4. Click **Add subscription**.
5. Set up a **Personal subscription**:
    - Choose your own email address.
    - Set it to run **every day**.
6. Save the subscription. You should now receive daily reminders for reports that need to be submitted to ScholarOne.

Submitting to the JMS ScholarOne

> **Tip**
>
> Reminder: for the AEA, the JMS is ScholarOne.

Submitting to JMS

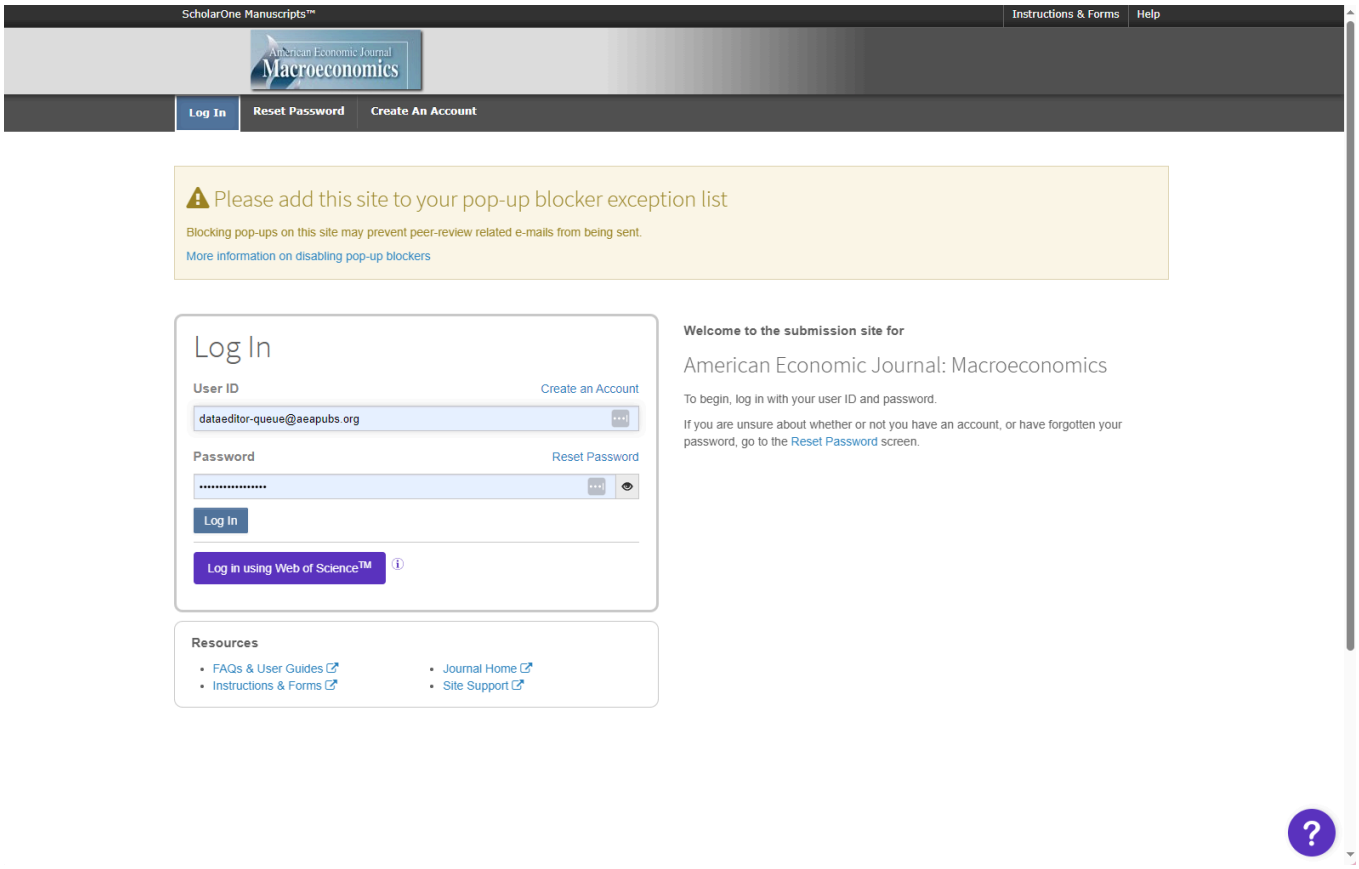
1. Open the issue on JIRA. It must be `Approved`.
2. Click on the `Submit to MC` transition. A pop-up will be shown.



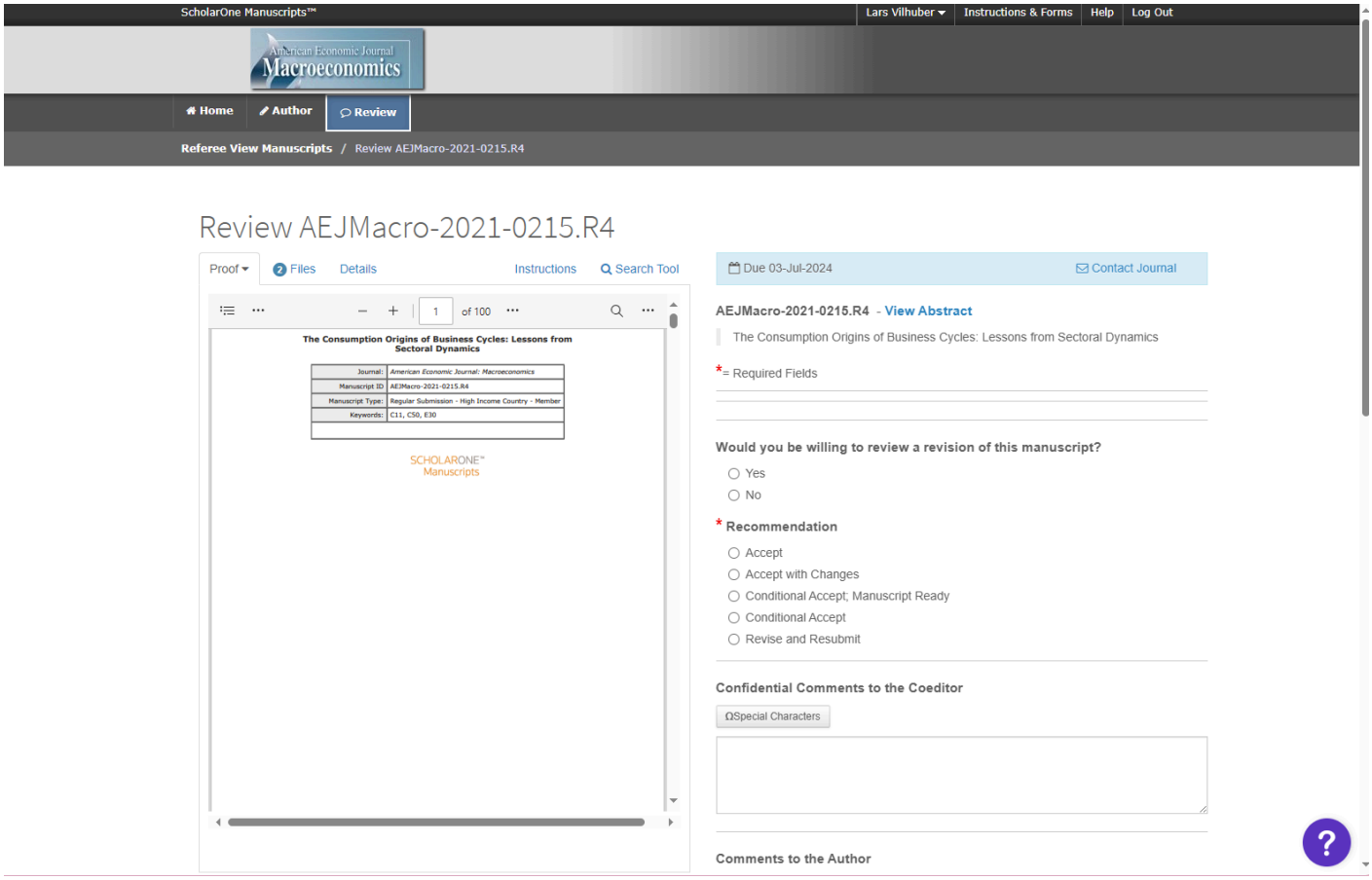
3. In the pop-up, you should have all the necessary information
    - Note: links in the pop-up window are not clickable: double-click, then use right-click to `Open in New Tab`
    - `MCEntryURL` has the link to Manuscript Central (MC) in order to submit
    - `Manuscript Central Identifier` to find the manuscript
    - Field [`MCRecommendationV2`] has the information about how the editorial office should proceed, to be selected in the JMS
        - In cases where `MCStatus` contains `RR`, the information will be in field [`MCRecommendation`]
    - Field [`Git working location`] has the information to clone the repository (and thus be able to access the report)
4. If necessary, clone the Bitbucket repository associated with the issue. You should find a `REPLICATION.pdf` there.

> **If there is no `REPLICATION.pdf`...**
>
> … then the Approver probably forgot to create it. Contact the Approver to avoid any misunderstandings.

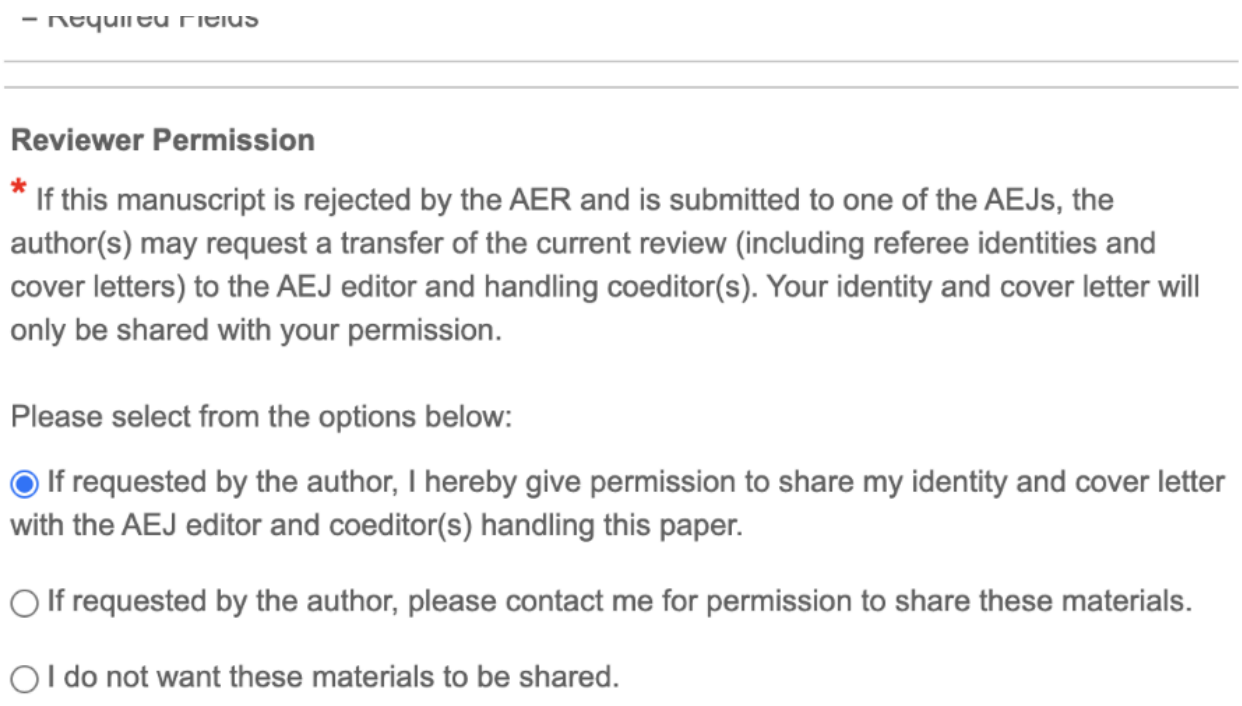
6. Open the Manuscript Central link `MCEntryURL` (double-click, right-click, open in new tab)
    1. Enter your credentials to access Manuscript Central (let LastPass fill the information)



2. Click on the review tab and identify the manuscript number (`Manuscript Central Identifier`) of the paper
3. Select `Continue Review`



4. Choose the first item in `Reviewer permission`



5. Click `Yes` when asked: Would you be willing to review a revision of this manuscript?
6. Select the recommendation as noted in the Issue
    - If `MCStatus` contains `CA`: Look at the field [`MCRecommendationV2`]
    - If `MCStatus` contains `RR`: Look at the field [`MCRecommendation`]
7. Copy-paste the “Summary” part from `REPLICATION.md` into the field `Comments to the Author`.
8. Select and upload the `REPLICATION.pdf`, click on `For author and  editor`.
9. In some cases, the Data Editor will have put a note in the issue with a “for Editor only” file. The contents of that file should be copied and pasted into the field `Confidential Comments to the Coeditor`.
10. Re-verify all information
11. Click on `Submit`
12. Back in the pop-up,
    1. Click on `Submit to MC`

Deleting restricted data

Decision point?

Note

If

- [`Agreement signed`] is empty, or it has some value, and
- [`Was data deleted?`] is `Yes`

then you can proceed to Submitting deposit-related information via openICPSR.

Otherwise, you will need to go through the “Restricted Data Deletion” process outlined on this page.

Warning

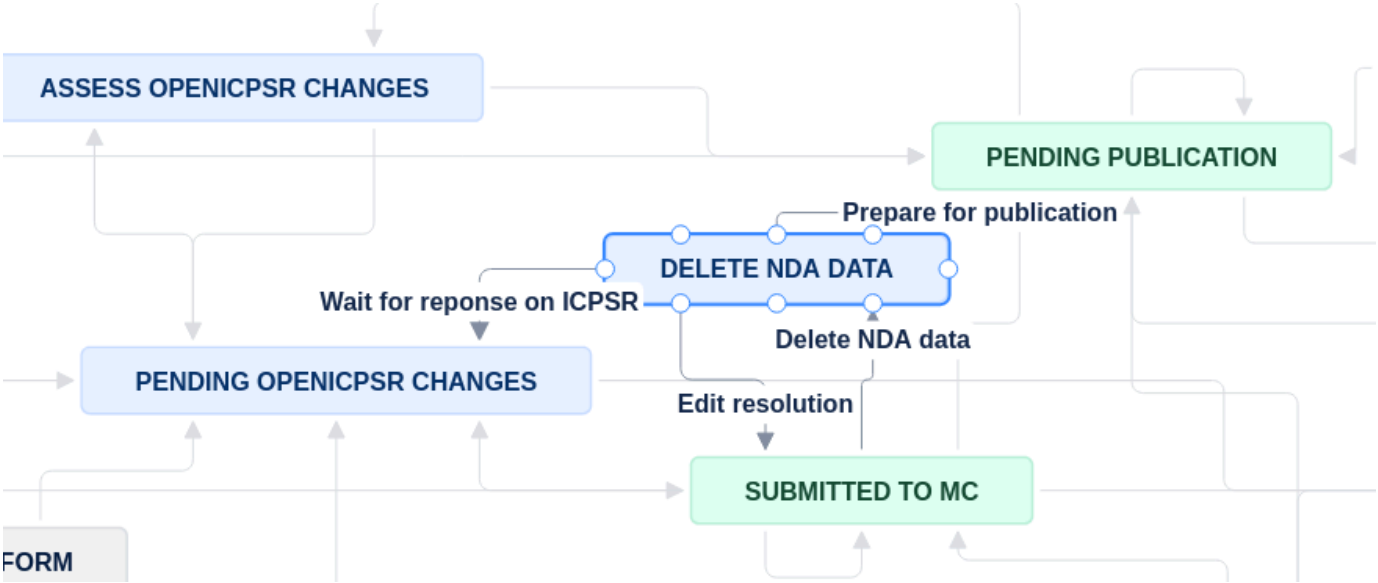
This does not apply to `MCStatus` = `RR` or `MCRecommendationV2` = `Conditional Accept` cases. For both of these scenarios, we will see the replication package again, and will need the restricted data.

Note

The process of deleting data should be performed by the Research Aide.

Moving the Jira issue

You should move the Jira issue to the `Delete NDA Data` status by choosing `Deleting NDA Data`.



Delete the data

- [ ] CCSS-Cloud: Delete the data from the L-Drive.
- [ ] If some processing occurred on BioHPC: Confirm deletion with the RA, or delete the entire `aearep-xxxx` folder.
- [ ] Box: Follow instructions at LDI Research Aide guide

Move the issue forward

Once the box for [`Was data deleted?`] is marked `Yes`, you can proceed to the next steps. Select `Pending openICPSR Changes`.

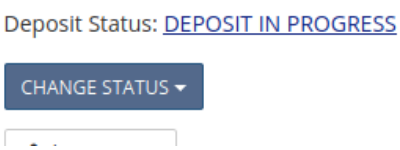
Submitting deposit-related information via openICPSR

Decision point?

> **Note**
>
> If
>
> - [`MCStatus`] contains `CA` and [`MCRecommendationV2`] = `Conditional Accept`, or
> - [`MCStatus`] contains `RR`
>
> then proceed to Request revisions.
>
> Otherwise, if [`MCStatus`] contains `CA` and [`MCRecommendationV2`] = `Accept` or `Accept with changes` then proceed to Preparing Deposit for Publication.

Request revisions

In principle, once the report is uploaded, the author will get the report with the requested revisions via ScholarOne. However, under the post-August 2020 workflow on openICPSR, the project may need to be unlocked for the author to make changes. To do so, proceed as follows:

1. Open the issue on Jira.
2. If the issue is still in `Submitted to MC`, then choose `Post message to openICPSR` transition, which will pop up with fields displayed. Wait here.
3. Right-click on the [`Replication package URL`] field to open the openICPSR deposit.
4. On openICPSR,
   - verify what the openICPSR `Deposit Status` is (top right corner) 

   - if `Deposit Status` = *`Deposit in Progress`*, you are done on openICPSR. Go back to the Jira issue
   - if `Deposit Status` = *`Submitted`*, then

    - click on `Change Status`, choose `Request revisions`

- in the pop-up, paste and submit the following lines:

> **We call this**
>
> the **“Request-revisions-RR” message**.

Use this template if the paper is in [`RR`] :

```
Revisions requested. Details in the full report, which the manuscript's corresponding author will receive
or from the editor in charge of the paper.

> [NOTE] Replication packages may be published as soon as all requested changes to the deposit have been
```

Use this template if the paper is in [`CA`] :

```
Revisions requested. Details in the full report, which the manuscript's corresponding author will receive
in a few business days.

> [NOTE] We may publish replication packages as soon as all requested changes
to the deposit have been made. Please process any requested changes as soon
as possible.
```

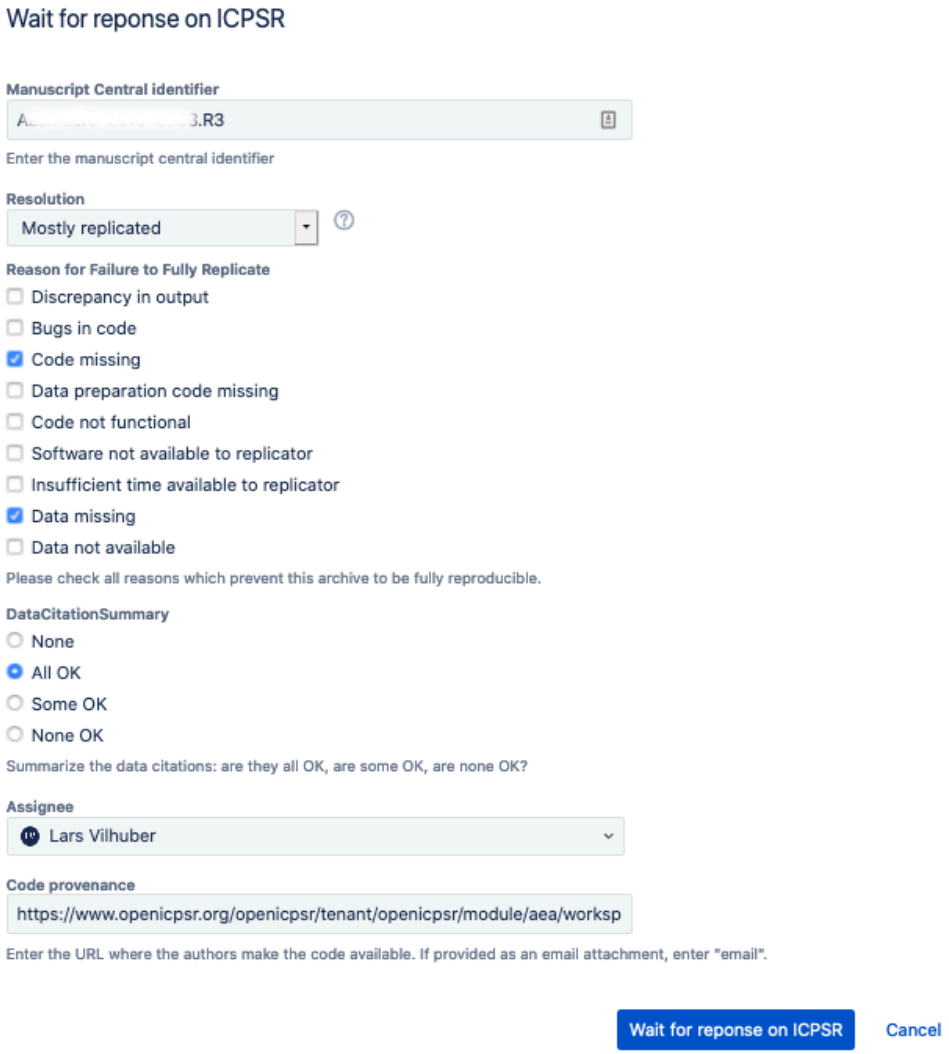
4. Back in the Jira issue, continue to transition the issue to `Done`.

Preparing Deposit for Publication

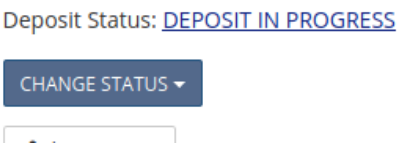
FOR ACCEPT WITH CHANGES

1. Open the issue on Jira.
2. Click on the `Wait for response on openICPSR` transition to `Pending openICPSR changes`.

- In the pop-up, you should have all the necessary information.
    - Note: links in the pop-up window are not clickable: double-click, then use right-click to “Open in New Tab”.
    - If not already done: `Replication package URL` should point to openICPSR. If not, go to the final step.
    - Make a note of the issue number (in the URL) and the `Manuscript Central identifier` again.



3. On openICPSR,

- verify what the openICPSR `Deposit Status` is (top right corner) 

.

4. On openICPSR, if `Deposit Status` = *`Deposit in Progress`*
    - start a message in the Communication log:
        - with subject line: `Please make the following changes (AEAREP-xxx)` (replace with appropriate numbers)
        - Message content:
            - the contents of the portion of the report after "`Action Items (openICPSR)`", but ONLY the action items.
            - then the following lines

> **We call this**
>
> the **"Request-revisions-CA" message when** `Deposit in Progress` is shown.

```
Details in the full report, which the manuscript's corresponding author will receive via ScholarOne in a
few business days. Please provide your response to the items listed above
via the openICPSR Project Communication log, specifying AEAREP-xxx.
Other items in the report may need to be addressed via ScholarOne.

Once all changes have been made, please change the status of your deposit to "Submit to AEA".


> [NOTE] Replication packages may be published as soon as all
requested changes to the deposit have been made. Please process
any requested changes as soon as possible.
```

(replace xxx with the issue number)

6. On openICPSR, if `Deposit Status` = *`Submitted`*:

- click on `Change Status`, choose `Request revisions`
- in the pop-up,
  - paste the contents of the repository-specific portion of the report after "`Action Items (openICPSR)`", but ONLY the action items.
  - then the following lines:

> **We call this**
>
> the **"Request-revisions-CA" message when `Submitted` is shown**.

```
Details in the full report, which the manuscript's corresponding author will receive via ScholarOne in a
few business days. Please provide your response to the items listed above
via the openICPSR Project Communication log, specifying AEAREP-xxx.
Other items in the report may need to be addressed via ScholarOne.

Once all changes have been made, please change the status of your deposit
to "Submit to AEA".


> [NOTE] We may publish replication packages as soon as all requested changes
to the deposit have been made. Please process any requested changes as soon
as possible.
```

(replace xxx with the issue number)

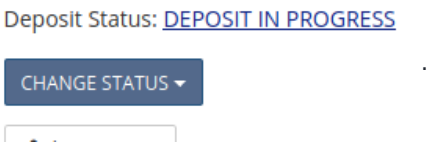
FOR ACCEPT:

1. Open the issue on Jira
2. Click on the `Prepare for publication` transition

If the transition is not available Click to hide

There are several conditions that need to be met in order to move it forward. Check that all apply if the transition is not available

- If there was an NDA, then `Agreement signed` will be filled, but the field `Was data deleted?` must be set to `Yes`. If not, check with supervisor.
- Field `openICPSRversion` has to have a value. This is usually automatic.
- `MCStatus` cannot be `RR`
- The field `Non-compliant` cannot be 'Yes'.
- The field `MCRecommendationV2` cannot be `Conditional Accept`

3. In the pop-up, you should have all the necessary information.
    - Note: links in the pop-up window are not clickable: double-click, then use right-click to "Open in New Tab".
    - If not already done: `Replication package URL` should point to openICPSR. If not, go to the final step.
    - Make a note of the issue number (in the URL) and the `Manuscript Central identifier` again.
4. On openICPSR,
    - remove any RAs from the Share list (leave anybody else who is on there!)
    - (new!) verify what the openICPSR `Deposit Status` is (top right corner) 

.
5. On openICPSR, if `Deposit Status` = *`Deposit in Progress`*
    - start a message in the Communication log:
        - with subject line: `Data and Code Deposit accepted for MCNumberXXX AEAREP-xxx` (replace with appropriate numbers)
        - with body

We call this

the **"Sign-off" message when author needs to `Submit`**.

```
This data and code deposit is accepted.

Action items:

 - Author: Please change the status of your deposit to "Submit to AEA"
 - AEA Staff: update DOI, Vol, Iss, Year of related publication, then publish.
```

6. On openICPSR, if `Deposit Status` = *`Submitted`*:

- start a message in the Communication log:
    - with subject line: `Data and Code Deposit accepted for MCNumberXXX AEAREP-xxx` (replace with appropriate numbers)
    - with body

We call this

the **“Final Sign-off” message**.

```
This data and code deposit is accepted.

Action items:

 - Author: no further action required
 - AEA Staff: update DOI, Vol, Iss, Year of related publication, then publish.
```

7. Back in the Jira popup, finalize by clicking `OK`. The issue will be moved to `Pending Publication`.
8. You are **not quite done** yet!
    - You will receive an email from openICPSR.
    - forward the email to *dataeditor-queue@aeapubs.org* , but **don’t press send yet!**
    - manually add the **issue number** (AEAREP-xxx) into the subject line
    - delete anything in the body of the email before the “From:”
    - Now send the email.
    - This will add the message to the Jira ticket.

> ICPSR does not always successfully send out a notification email for the posting of the comment. If you don’t receive the email, as a last resort, simply copy and paste your ICPSR comment into the Jira ticket so that we have a record.

AEA: Interfacing with repositories

Note

The AEA uses openICPSR as its primary data and code repository. However, other acceptable repositories include Zenodo, Codeocean, and others.

In addition to the various tasks already described on openICPSR (here, here, and here), some other tasks may sometimes need to be done on the platform. This section describes those tasks.

AEA: Monitoring Pending openICPSR Changes

Pre-requisites (JIRA)

One-time administrative setup (permissions) Click to hide

- [ ] assessors need to be in the `Assessor` group (**Assistant** can set in Project -> Settings -> People in Jira)
- [ ] assessors need permissions on openICPSR (Data Editor or **Assistant** requests from openICPSR)

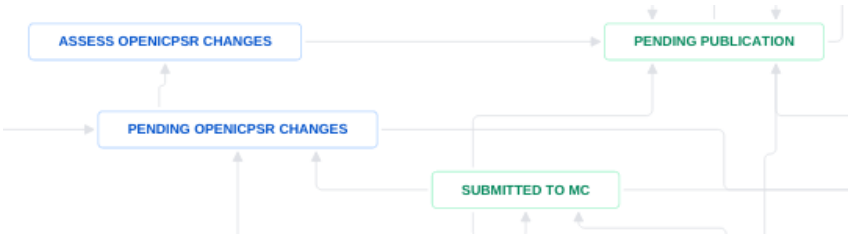
Background

Many cases which the Lab reviews receive a recommendation ([`MCRecommendationV2`] on Jira) of `Accept - with Changes.` What this means is that the changes which are requested do not constitute a complete revision from the authors. Instead of re-submitting a complete revision for review by the Lab, the authors will make any necessary changes to the deposit directly on openICPSR. Separately, any changes to the manuscript/appendix will be made at the copyediting stage by the editorial office, without further interaction with the Lab.

Note

In some cases, the deposit may have been made at a different repository, such as Zenodo or Dataverse. Wherever you read “openICPSR” here, treat it as “wherever the deposit may be”. Other repositories have other methods of inspecting files, or downloading the entire package.

It is important to understand how the submission process works once the final report has been approved. The RAs responsible for this process are following the instructions in Submitting information back to the Editorial Office and to authors. If you are not yourself involved in that process, please review those instructions. Briefly, cases that are designated as `Accept - with Changes` will have the reproducibility report submitted to ScholarOne (aka Manuscript Central (MC)), the Jira ticket will be moved into status `Pending openICPSR changes`, and the ICPSR deposit will have been unlocked so that the authors can make changes, with comments to that extent in the “Project Communication Log.”



The openICPSR deposit is subsequently regularly monitored. Authors may contact the Data Editor as well. When it is clear that edits have been made to the deposit, the Jira issue is moved to `Assess openICPSR changes`. The Data Editor and team then need to verify that all of the `[REQUIRED]` tags in the last submitted report have been completed. The RA conducting that check can review the required changes in the full report on Bitbucket, but may also need to consult additional communications in the Project Communication Log in openICPSR and on Jira.

The evaluation process usually does **NOT** require running any code. In most cases, these are minor changes, such as adding software dependencies or data access instructions to the README. Some cases will involve minor debugging issues, for which the Lab only checks to see that edits to the code have been made as identified in the report. Unless specifically instructed to do so by the Data Editor, do not run code.

Process

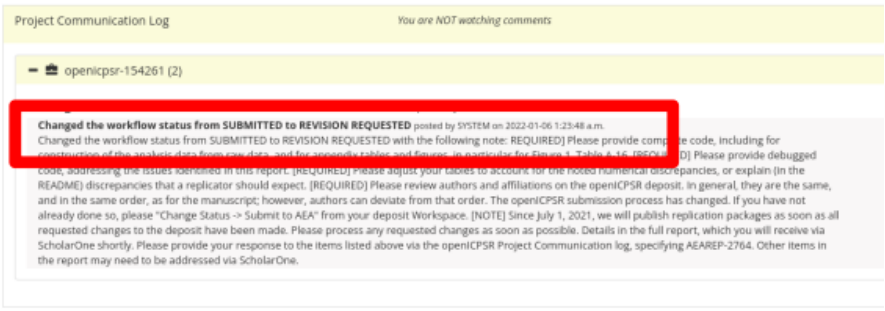
If you are the RA tasked with this, these are your instructions.

Verifying if changes have been made

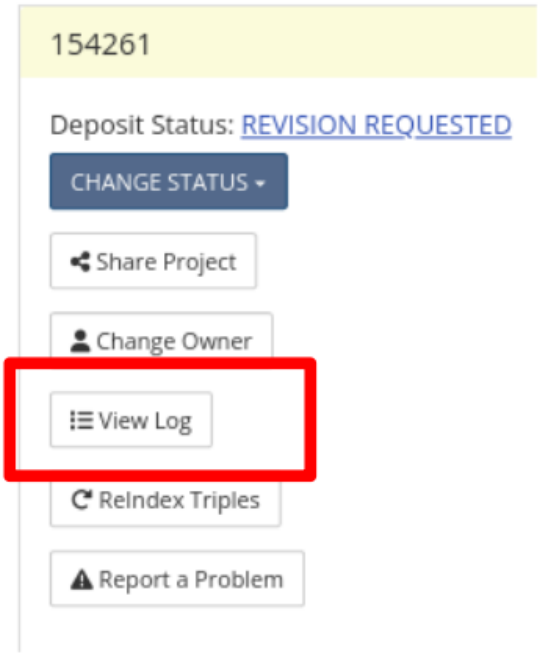
- A good place to start in this process is to go to the openICPSR deposit (fields [`Replication package URL`] or [`openICPSR alternate URL`])
- First, verify the **date** of the last request made to the author, in the "Project Communication Log" area. The Project Communication log always contains our request to the authors, and may contain subsequent responses from the author.

While checking the Communication Log, look for any uploaded response letters from the authors. If a response letter is attached there, download it and attach it to the corresponding Jira ticket, so we have a complete record in one place.

Now open the project log: click "View Log", just below "Share Project" and "Change Owner."



- Now open the project log: click "`View Log`", just below "Share Project" and "Change Owner."



- This log will tell you all the changes that have been made to the deposit and when. From here we can tell whether or not the authors have made any changes since we originally requested the revisions.
- Additionally, this is a great resource for checking which program files the authors have made changes to.

No Changes Have Been Made

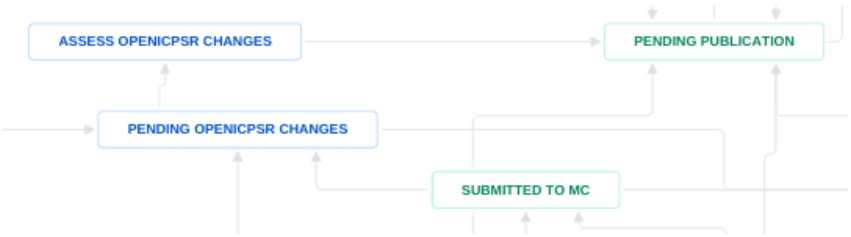
- If no changes have been made to the deposit **four weeks** after requesting revisions, start a message in the Communication log:
  - with subject line: `AEAREP-xxx Data and Code Deposit Revisions Reminder` (replace with appropriate numbers)
  - with body

```
Authors,

Please make the revisions requested to the openICPSR deposit so that we may
move forward with publication of the deposit.

See our previous comment above and our full report for details.
Feel free to contact us directly at dataeditor@aeapubs.org with any questions.


> [NOTE] Starting July 1, 2021, we will start to publish replication
packages as soon as all requested changes to the deposit have been made.
Please process any requested changes as soon as possible.

Thanks!
```

- Make a note in the Jira ticket that such a message has been posted.
- If after an additional **four weeks** still no changes have been made, make a note in the Jira ticket, tagging/ alerting the Data Editor.

If changes have been made

Once you have ascertained that changes have been made, and if this hasn't already been done, move the Jira ticket to `Assess openICPSR changes`.



Update the Bitbucket repository

We want to capture the latest code, and generate a listing of all the files in the openICPSR deposit. To this extent, re-run the Ingesting Author Materials steps.

> **Note**
>
> If for some reason, this fails (see screenshot), you may need to update the Pipeline tools in the repository, then try again.

```
if [ -f tools/download_openicpsr-private.py ]; then python3 tools/download_openicpsr-private.py $openICPSRID; f
```

```
1  + if [ -f tools/download_openicpsr-private.py ]; then python3 tools/download_openicpsr-private.py $openICPSRID; fi
2  No debug:None
3  Traceback (most recent call last):
4    File "tools/download_openicpsr-private.py", line 152, in <module>
5      oauth_redir_url = data_req.headers.get("Refresh").split("URL=")[-1]
6  AttributeError: 'NoneType' object has no attribute 'split'
7
```

Verify Requested Changes

You should now open the Bitbucket report, and identify the changes that the authors were expected to make. How to do this will vary. In some cases, you may be able to simply inspect the deposit, in others, you may need to download the deposit again, and verify the changes made, as you might do for a full revision (see Revision Reports).

> **Warning**
>
> Except in extremely rare cases, do **not** re-run code.

- Open the report. You can do this by checking out the Bitbucket repo, or by clicking on the [`Report URL`] field.
  - Check the "Summary" (at the top), as well as the "Reason for incomplete reproducibility" (at the bottom, and in the Jira issue)
- Copy all the items in `Action Items (open ICPSR)` (disregard the manuscript section, this is checked by the editorial office)
- Paste the Action Items into the Comments section of Jira.
- Go through each item and verify the changes by marking 'Done' or 'Incomplete' next to each item. Refer to the summary or the action item for more detail as to why this item was considered incomplete. Add any relevant details. For example:

[REQUIRED] As specified in the Policy and the DCAF, the README shall follow the schema provided by the Social Science Data Editors'template README. ***Incomplete****- They added computational requirements but are missing Statement About Rights*

- For each change, you should make a note of what problem was addressed. For instance, if the "Reason for incomplete reproducibility" notes that the code contained fixable bugs, and the authors have made the changes noted in the report, then you will want to uncheck the box "Bugs in code".
- You can find the "Reasons for incomplete reproducibility" in the "Repl. info" tab in the Jira ticket, or in the popup when moving from `Assess openICPSR changes` to `Pending publication` at the end of this process.
- Check that the deposit doesn't have any extraneous files (Manuscript PDFs, Response to the Editor PDF, files with tildes (`~_readme.doc`, …) etc). If there are these files, note this on the Jira comment, tagging the senior members of the lab. Authors must remove these files. If in doubt, check with senior members.
- **Check that the deposit is "Submitted" status**.

How do you know that the changes are sufficient?

For instance, the last report might have said:

```
- [REQUIRED] Please provide debugged code, addressing the issues identified in this report.
```

and the report will have contained some mention of the bugs found. Without re-running the code, how can we assess this?

Inspect the code, in particular the changes made to the code (see `Commits` tab on Bitbucket and navigate to the file that was a problem), and see if it is **plausible** that the changes fix the bug. In general, we trust that the authors, provided with a list where bugs occur, are able to fix these types of bugs at this stage.

Additional exceptions

Some tags may not have a clear resolution just by scrutinizing the deposit, because they may have been handled in the copy-editing process and adjustments to the manuscript, which we (the Data Editor team) do not observe. All tags that appear in the "Action Items (manuscript)" section are communicated by the copy-editing team to the authors, so we **assume** that they will be handled there. Here are a few tags that fall into that category:

- "[REQUIRED] Please adjust your tables to account for the noted numerical discrepancies, or explain (in the README) discrepancies that a replicator should expect".

If the README makes no mention of this, and there is no mention in the Project Communication Log, then you will need to assume that this was handled by adjusting the manuscript or (online) appendix. Add the following note:

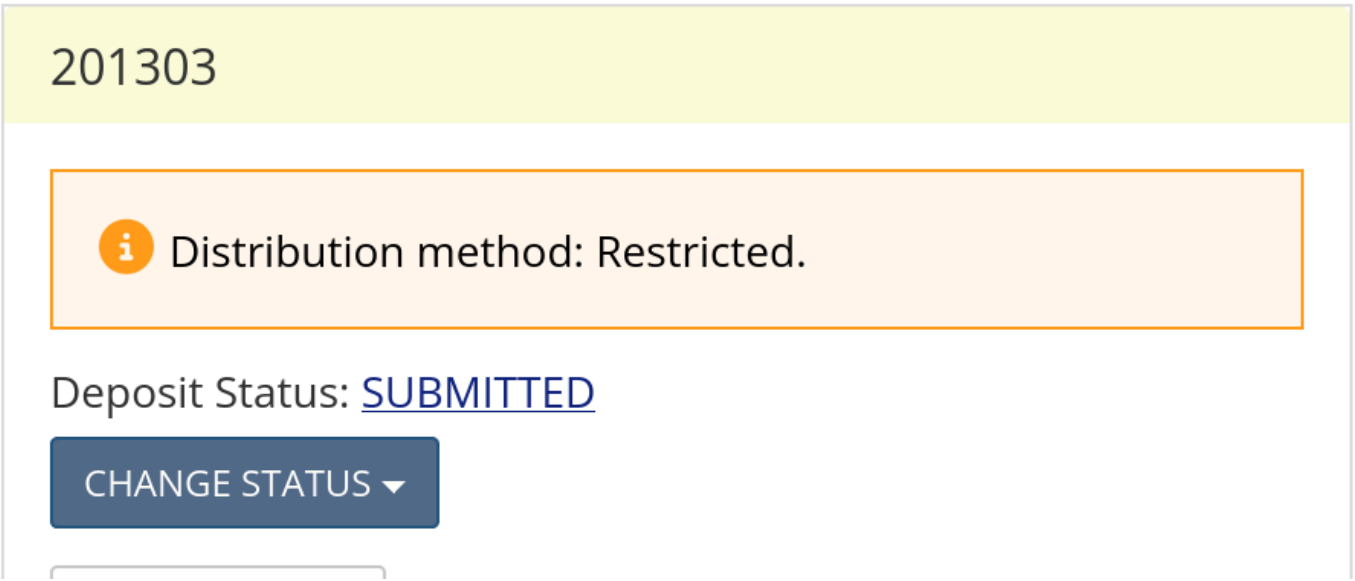
> **Not verifiable by us**. Not addressed in the README or code, we assume that the online appendix and manuscript will be/has been updated.

- **If you have verified that all the required changes have been made**, continue with the "For Accept" section of Submitting deposit-related information via openICPSR.
  - You have another opportunity to uncheck any boxes here that have been addressed.

- Check under “Other links” if “Non-compliant” = yes, if so, **do not proceed** until you find clarification
- - See if the reason for non-compliance (usually mentioned in the report, and at least in the comments) is resolved, **consult with the Data Editor**, then uncheck that box.
- If the deposit is **not** in “Submitted” status, choose the appropriate variant of the **Signoff** message.

Verifying if Deposit was marked Restricted

In some cases, authors will have checked a few boxes (see Guidance) that lead to the deposit being marked as **Restricted**. In almost all cases, this is in error.



You must immediately notify the Data Editor’s assistant, who will contact the authors to clarify.

> **Warning**
>
> Do not sign off on a deposit when it is marked as **restricted** unless instructed to do so.

Reporting Insufficient Changes to the Authors

- If, in your review, you find that not all changes have been made, or it is unclear whether or not certain changes are acceptable/sufficient, please reach out to the assistant to the Data Editor with a comment on the Jira ticket outlining your question.
- These cases are not always cut and dry, please err on the side of caution and ask questions before posting a “final acceptance” message on the deposit, or before requesting additional changes from the authors.
- Do not include any `[SUGGESTED]` items in the response. We only suggest **once**, in the original report. If the only items not completed are `[SUGGESTED]` items, the deposit is accepted.
- **If you are certain that items are incomplete** and theres 3 or less action items remaining:
  - Go back into openICPSR and “Change status” -> “Request Revisions”. Then input the message:

```
Thank your for your revisions. The remaining action items need to be addressed:
```

    - Then paste only the incomplete `[REQUIRED]` action items (along with the reason we consider this incomplete), ignore any `[SUGGESTED]` items.
- **If you are certain that items are incomplete** and theres more than 3 action items remaining:
    - Go back into openICPSR and “Change status” -> “Request Revisions”. Then input the message:

```
Thank your for your revisions. The remaining action items are detailed in the Communication Log
```

    - Paste the action items in the Communication Log with the Title “Revisions Requested (AEAREP-xxx)” so that it’s easier for the author to read the list.
- Then move the Jira ticket back to `Pending openICPSR changes`

Summary of Steps

1. Open `REPLICATION.md` on bitbucket
2. Copy all the items in “Action Items (openICPSR)” (disregard the manuscript section since we can’t check that at this point)
3. Paste the Action Items into the Comments section of Jira.
4. Go through each item and mark ‘Done’ or ‘Incomplete’ next to each item. With any relevant details afterwards.
5. If all changes have been made, continue with the “For Accept” section of the “For Accept” section of Submitting deposit-related information via openICPSR.
6. If some items remain incomplete, tag Data Editor’s assistant in Jira, who will then contact the authors.

Notes

- **A note on [SUGGESTED] items**. We, of course, attempt to get authors to make their deposit as reproducible as possible. Which means suggesting improvements such as creating a `master.do` or including code to automatically export results. However, they are only suggestions. In other words, they do not impede reproducibility and thus we do not require that the authors make those changes. If the only changes not made to the deposit were [SUGGESTED], move forward with acceptance.
- **A note on deposit status**. When an openICPSR deposit has a status of “Submitted” it is locked. This means that the authors will not be able to make any changes. If the deposit status is “Deposit in Progress” or “Revisions Requested” the deposit unlocked and changes may be made. Review the submission instructions above for information on how to unlock a deposit.

Note

When authors ask if they need to re-submit the updated manuscript to ScholarOne/Manuscript Central. Paste the following within the acceptance (or reminder) post on openICPSR:

```
At this stage, any changes to the manuscript are handled directly with editorial office and another submission to Scholar One is not necessary. If you are not already in contact with the editorial office, please reach out to the managing editors via aejaccept@aeapubs.org or aeraccept@aeapubs.org.
```

AEA: Publishing a deposit

Responsibilities

The complete general sequence is as follows:

1. Publish replication package deposit (openICPSR, others)
2. Publish AEA article
3. Update the “related publication” link using the “Import from DOI functionality.”

In general, publication of the deposit is done by the AEA Editorial Office, in synchronization with the publication of a the manuscript.

In certain circumstances, this may also be done by the AEA Data Editor’s office. This page describes how to proceed.

Please note:

Always check with your supervisor before attempting to publish a deposit.

Pre-requisites (JIRA)

Note

Fields are in `Publication info` tab, unless otherwise noted.

- ☐ The openICPSR project number must be filled out, if the paper has an openICPSR deposit.
- ☐ The correct `openICPSR version` number needs to be selected.
- ☐ A `replication package DOI` needs to be present (if necessary, run a script).
- ☐ The field `Non-compliant` is not checked (not `Yes`) (in `Other links` tab)

Warning

Some manuscript may have deposits elsewhere. Handling may differ in those cases. Examples include:

- World Bank Reproducible Research Repository
- Codeocean Open Science Library
- Zenodo American Economic Association community
- Dataverses, for instance at Harvard, Yale, and Canada

Instructions for those may not yet exist.

One-time administrative setup Click to hide

- [ ] (Administrator for Project) needs to add user to `Publisher` group.

JIRA: Move to “Processing Publication”

Select the transition “`Publish`” to move it to “`Processing Publication`”. In the popup, verify that all information seems complete.

Deposit: Sequences

On openICPSR, the sequence is `Add DOI` -> `Publish`. For other platforms, it often is `Amend publication with DOI`.

In a separate tab, open the deposit.

Deposit: Add DOI

openICPSR | **Zenodo** | **Zenodo bulk updates** | **Codeocean** | **Others**

Verify that the usual “ready for publication” communication entry has been added. Click to hide

- if not, investigate why

Scroll down to the “Related Publications” section.

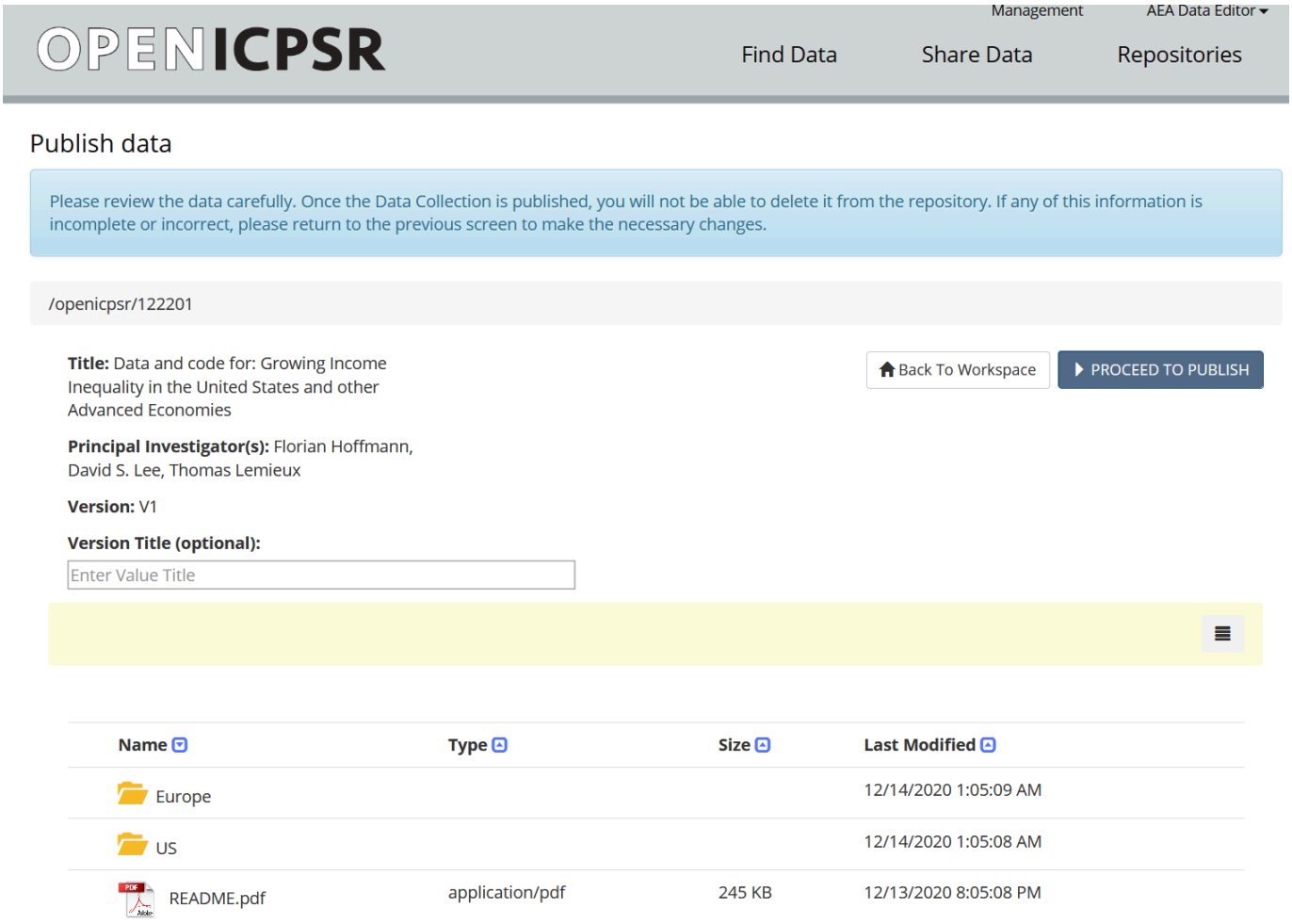
- Go through the process to add the DOI.
- Remove any related publications that simply say “n.a.” in publication info
- If there are any “supplemental” data publications (e.g., on Zenodo, see **Zenodo** tab), link them here as well. You may need to first “accept” or “publish” the supplemental deposits.

Deposit: Publish

openICPSR

Move to the next step:

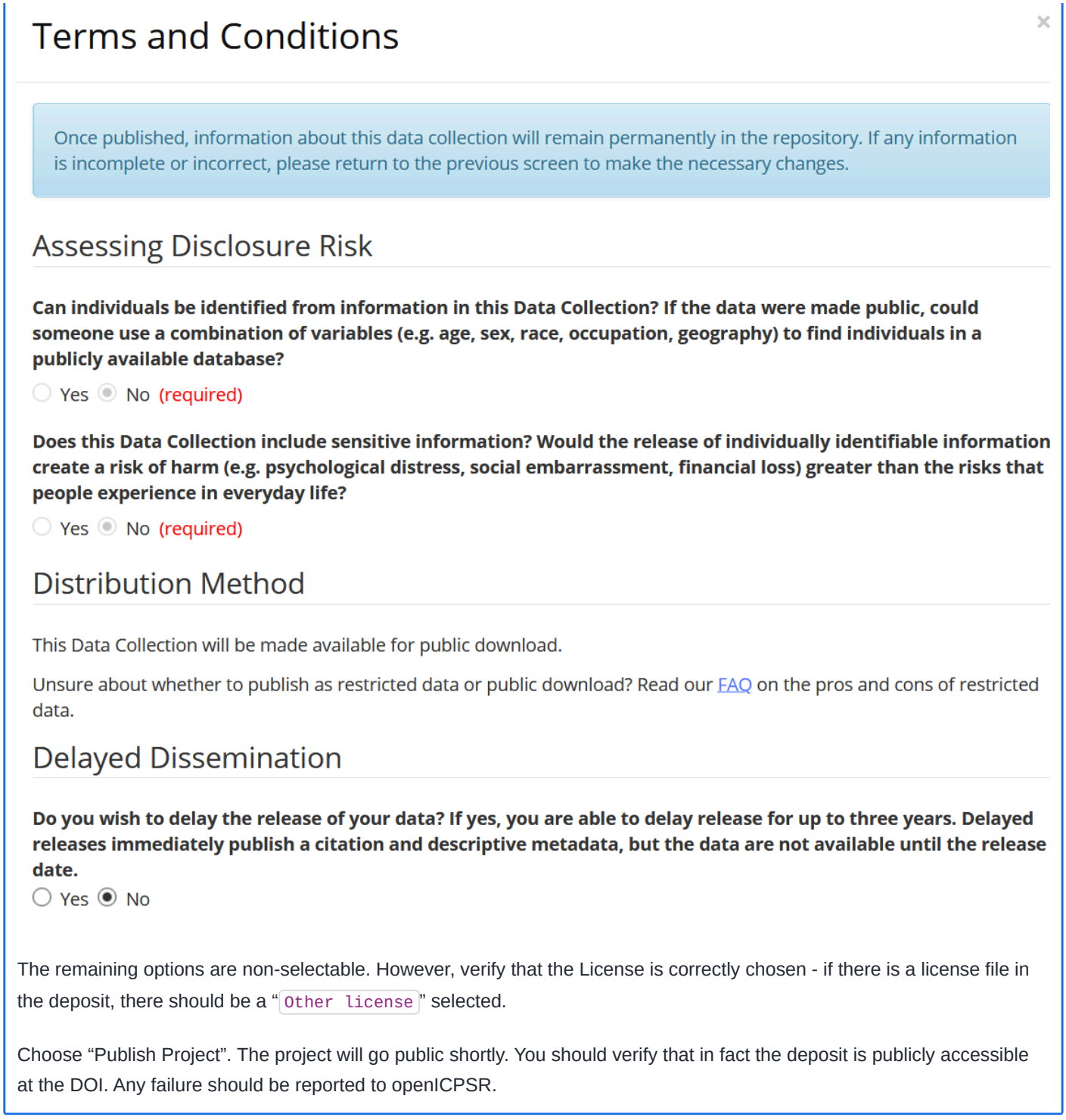
> Change Status -> Publish



Verify the information again, then “Proceed to publish”.

Most of the values on the next screen should have been consented to by the authors. However,

- Verify that the data collection is scheduled for “public download.” Only in rare cases will this be for restricted-access.
- Verify that “`Delayed dissemination`” = “`no`”. We never use that.



The remaining options are non-selectable. However, verify that the License is correctly chosen - if there is a license file in the deposit, there should be a "`Other license`" selected.

Choose "Publish Project". The project will go public shortly. You should verify that in fact the deposit is publicly accessible at the DOI. Any failure should be reported to openICPSR.

JIRA: Move to "Published"

Finalize the process on JIRA.

Adding the article DOI to the openICPSR deposit metadata

- Choose the space "related publications"

[screenshot here]

- (optional) remove the existing manually added temporary link (usually with date “n.d.”)
- obtain the article’s DOI
- use the “Import from DOI” functionality
- Verify that all information seems correct.
- Save.

It is not necessary to publish the deposit again, this metadata takes effect immediately.

> **Note**
>
> Authors can also do this themselves. In the case of PSID or related repositories, this action **must** be performed by authors themselves.

Adding the article DOI to the Dataverse deposit metadata

When the replication package is hosted at a Dataverse repository, the article DOI must be added to the metadata by the authors or the Dataverse curators. The following steps describe how to convey this to the authors or curators

Go to the deposit

The deposit DOI should be available in the field `RepositoryDOI`. Click on it.

Choose `Contact Owner`

There usually is a button called `Contact Owner` top right:

Fill out the `Email Dataset Contact` form

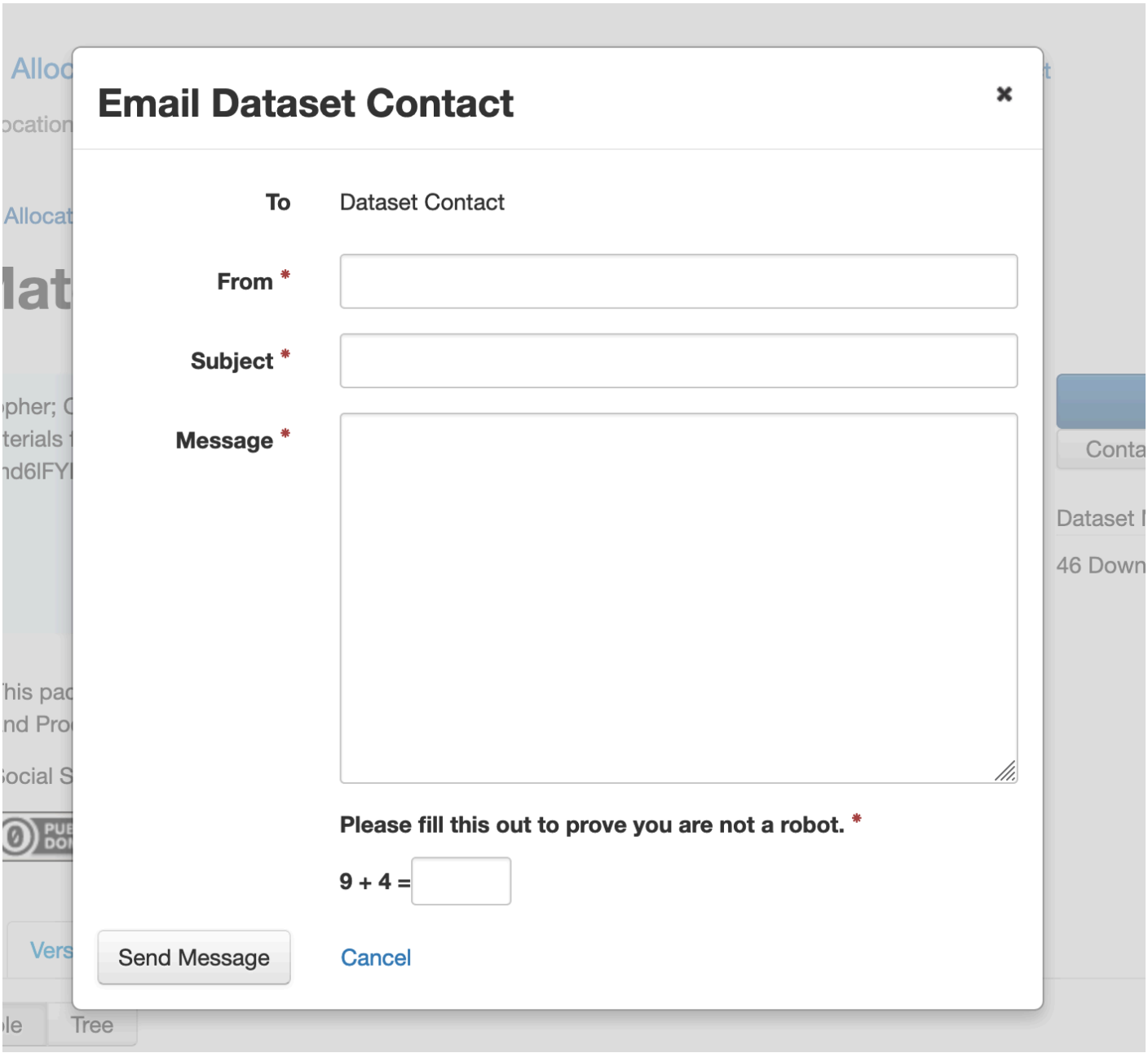
You should use the following information:

- **From**: `dataeditor@aeapubs.org`
- **Subject**: `Please add article DOI to deposit metadata`
- **Message**:

```
Dear [AUTHOR], others,

please add the article DOI ([DOI HERE]) to the "Related Publication" field. We would appreciate it.

Sincerely,

Lars Vilhuber and the Data Editor team
```

where you should replace `[AUTHOR]` with the author's name, and `[DOI HERE]` with the article DOI.

Then fill out the math problem, and hit `Send Message`.



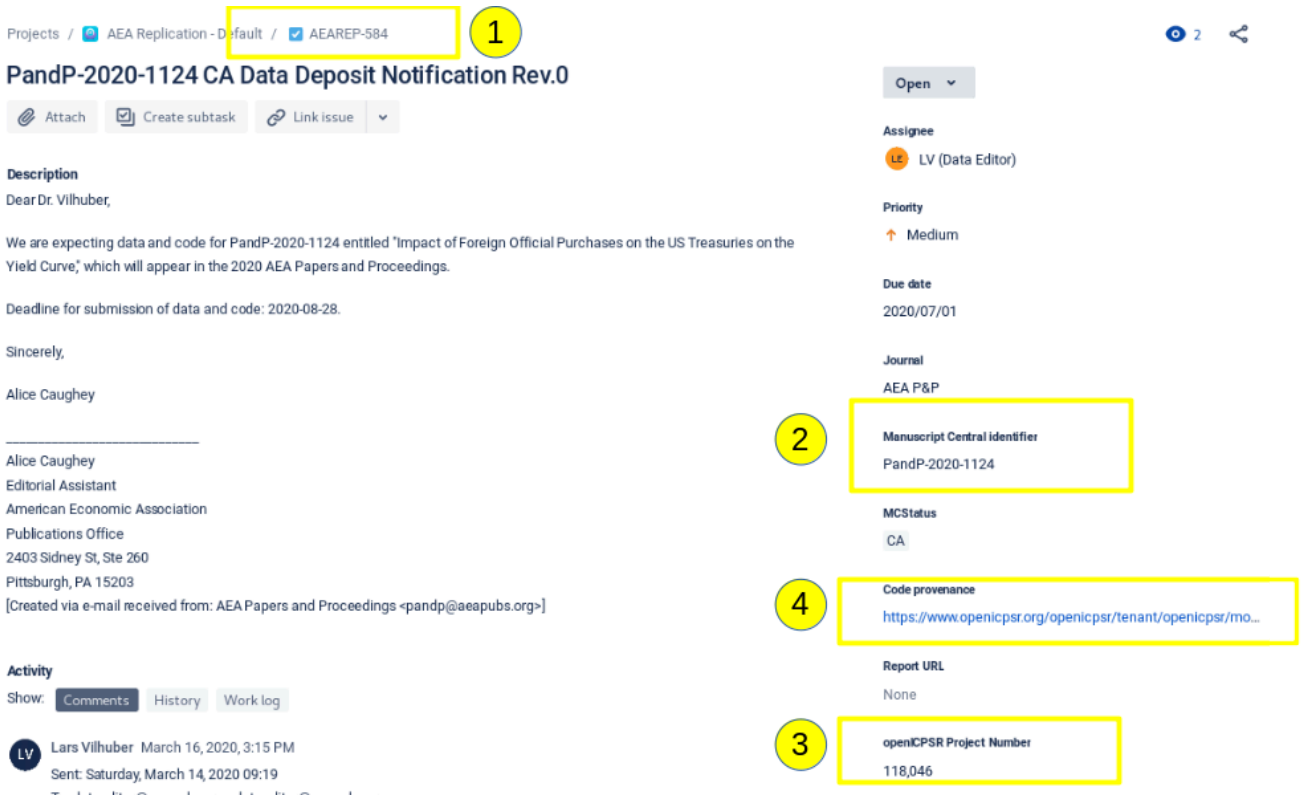
Make a note in Jira

Add a comment in the Jira ticket:

```
Left note on Dataverse contact form, requesting that authors  add article DOI to deposit metadata.
```

AEA: Instructions for Papers and Proceedings checks

You will be assigned a Jira issue. It should look something like this:



Take note of the 4 highlighted fields:

- (1) the “AEAREP” number (identifier for the issue on Jira)
- (2) the “Manuscript Central Number” (identifier on the AEA’s editorial system)
- (3) the “openICPSR Project Number” (identifier on the ICPSR system)
- (4) the link for “Replication package URL”, which should direct you directly to openICPSR

Additionally, certain steps below (i.e. transitioning to “Pending Publication”) require that you have “Publisher” permission in Jira. If it is unclear whether or not you have that permission, please reach out to a supervisor.

> **Note**
>
> Here’s the guidance document.

Step 1

You should now do the following:

- Open the link in (4) on openICPSR, (more rarely) Zenodo, or Dataverse
  - If you need to find the openICPSR repository, use this link
- Open the 2026 form: link here
- Assess the openICPSR repository, filling out the form as you go along

Requirements	open ICPSR	Dataverse	Zenodo
open ICPSR number	Applicable	N/A: Use doi number to distinguish	N/A: Use doi number to distinguish
Manuscript Number	Applicable	Applicable	Applicable
Jira Ticket (Internal Reference)	Applicable	Applicable	Applicable
Title	Applicable	Applicable	Applicable
Principal Investigators	Applicable	Applicable: Go to metadata, look for author	Applicable: Name is right below title
Summary/Description	Applicable	Applicable: Go to metadata, look for description	Applicable: Look at description below Author
JEL Classification	Applicable	N/A	N/A
Manuscript ID	Applicable	Applicable	Applicable
No ZIP Files	Applicable	Applicable	Applicable
README Format	Applicable	Applicable	Applicable
Subject Terms	Applicable	N/A	N/A
Geographic Coverage	Applicable	N/A	N/A
Time Period(s)	Applicable	N/A	N/A
Funding Sources	Applicable	N/A	N/A
Collection Dates	Applicable	N/A	N/A
Data Types	Applicable	N/A	N/A
Data Sources	Applicable	N/A	N/A
Unit of Observation	Applicable	N/A	N/A

- It is important that you enter the AEAREP and Manuscript numbers accurately. We suggest copying-and-pasting.
- Submit the form.
- Close the form after each issue, and open it again for the next one.
- Navigate to the “PandP Docs” Google Drive folder here, locate the PDF with the Jira ticket of the case you are working on (PDF format: `Report-PandP-AEAREP-xxxx.pdf`), and download the file. This is the PDF version of the checklist form using the information you entered.

In **Jira**:

- Move the issue forward by choosing “**Process PandP**” to status “*Submitted PandP Form*”

    - This indicates that you have completed the form.
- Attach the downloaded PDF to the Jira ticket.

Step 2

Once the form PDF has been added to the issue, you should first return to the openICPSR repository, and then come back to Jira. Depending on what's on the form, proceed to do one of the following:

If the form has all "required" elements checked, you should "sign off"

On openICPSR | **On Dataverse** | **On Zenodo**

- Write a "Project communication log" entry. After clicking on "+ add entry", copy the following to the relevant cell. Update AEAREP number accordingly.
    - *Subject line:*

```
AEAREP-xxx Data and Code Deposit for P&P submission accepted
```

    - *Content*:

```
Thank you for uploading your code and data. This
deposit is accepted. No computational verification was
conducted, only compliance with required metadata was checked.
```

In Jira

- Move it through the workflow
    - Choose "**Prepare for publication**"
    - Resolution: *Evaluation only*
    - MCRecommendationV2: *Accept*

If the form does not have all "required" elements checked, make the following entry

On openICPSR | **On Dataverse** | **On Zenodo**

1. If `Deposit Status` = `Deposit in Progress`

- Write a "Project communication log" entry
    - *Subject line:*

```
AEAREP-XXX Modifications to make to your P&P deposit
```

    - *Content:*

```
Thank you for uploading your code and data. We have checked the deposit for compliance with th
elements. Please see the attached form for modifications to make. Once all changes have been m
```

- Attach the PDF from the issue to the communication log.

2. If `Deposit Status` = `Submitted`,

- Change the status of the deposit to “Request Revisions” with the note:
  - Please see the attached form for modifications to make.
- Write a “Project communication log” entry
  - *Subject line:*

```
AEAREP-XXX Modifications to make to your P&P deposit
```

  - *Content:*

```
Thank you for uploading your code and data. We have checked the deposit for compliance with th
elements. Please see the attached form for modifications to make. Once all changes have been m
```

  - Attach the PDF from the issue to the communication log.

Back in Jira,

- Choose “**Wait for openICPSR response (PandP)**”
  - Resolution: “*Evaluation only*”
  - MCRecommendationV2 = “Accept with changes”
- Later, once the issues have been resolved (typically days or weeks later) follow the process under “required elements checked”

Guidelines for running code

In this section, we will provide you with a few guidelines on how to run code in the most robust and reproducible way! Solutions may differ for each software, but the principles are often the same. Come back here often, and let us know if something is wrong, or could be more complete.

Note

Some tips can also be found on our Wiki:

- Stata tips
- Matlab tips
- Python tips
- R tips
- and more.

Stata-related procedures

In this section, we will show you a few things related specifically to running code reproducibly with Stata. For some specific debugging tips, see Debugging Stata. For more general debugging tips for Stata and other computer languages, see our wiki.

> **The following instructions are for Windows.**
>
> See Instructions for BioHPC for running Stata code generally on Linux, and specifically on BioHPC.

Using config.do in STATA

When running code, we ask you to keep a log of what you do. Moreover, authors often use packages that are not part of STATA. How can you keep track of what is needed, and what was used by the code? We provide `template-config.do` in the template repository you clone which addresses these problems.

Why do we need log files?

- Log files record each step of the analysis and its results as a text. It also records error messages if you encounter any error upon running the code.
- There are other purposes to have log files, but for us, it is to communicate with other team members.
  - When a replicator submits the report, a preapprover (and an approver) needs to verify how the code ran. It is to ensure that any discrepancies we find are not due to mistakes on our end.
  - A log file is crucial for this verification. Otherwise, preapprovers and approvers have to run the code again to verify which is not an ideal use of time, nor an efficient way to process the case.

Why do we have to install programs?

- STATA, or many other statistical software, does not provide all the packages (or "libraries", or "modules") that enable or facilitate the analysis. Therefore, many user-written programs or extensions are publicly available for downloads. For STATA, this is often at the SSC (`ssc install package`), but sometimes at the STATA Journal archives (`net install sjt0444, using(http://www.stata-journal.com/software/sj16-3)`), and sometimes in authors' webpages (`net install package, using(https://author.com/packages)`)
- We differ in installation process from many others in the sense that, we want to install programs in a specified directory that is NOT a system directory.
  - This is to ensure that the set of packages used by replication package is complete. A complete replication package should be stand-alone, regardless of packages installed elsewhere in the machine that program is run on.

Explaining config.do

> **Note**
>
> In the following text, the line numbers mentioned are approximate. We regularly update the `template-config.do` to improve it, which may shift specific lines.

Setting directory structure information

The directory structure of the replication package and of the `main.do` within determines the location of `config.do`. It also determins a few other parameters within the `config.do`.

If the replication package contains a `profile.do`... Click to hide

In some cases, replication packages might use a non-Stata master file to run Stata code. Sometimes (not always) they will specify a location for a `profile.do` file. When you see this, for all the considerations as to where to locate the `config.do`, replace `main.do` with `profile.do`.

[ACTION] Adjust the `config.do`

- If the main .do file is directly placed in the root directory, set the parameter `scenario` to be `B` and save.
- If the main .do file is inside a folder, open `config.do` and set the parameter `scenario` to `A` and save. (This is the default, so really no action is necessary.)
- If the replication package includes a folder with STATA packages, add the line `adopath ++` followed by the path of the location of that folder and save. See "Using Config.do" for details.
- Add packages that need to be installed to `config.do`. See "Using Config.do" for details.

We will refer to the `directory` that contains the package as `$rootdir`. The scenarios helps us to define that programmatically.

- line 80, `global rootdir : pwd` sets the current working directory as a root directory, a.k.a. `rootdir`. It does not explicitly write the directory path, but see later on how to run STATA so that it picks that up **automatically**.

Warning

Many authors of replication package tell you to "write the path where the package is". DO NOT DO THIS. Use `$rootdir` instead.

Note

Any other paths defined or used within the replication package should be relative to `$rootdir`. This is to ensure that the replication package is self-contained, and can be moved to another machine without breaking.

Scenario A

A simplified directory structure that correspond to scenario "A" look like this:

```
directory/
             code/
                main.do
                01_dosomething.do
             data/
                data.dta
                otherdata.dta
```

Example

- A main .do file is inside a folder and you have placed `config.do` in that same folder.

```
* Template config.do */

local scenario "A"
* *** Add required packages from SSC to this list ***
local ssc_packages "estout ivreg2"
    // Example:
    // local ssc_packages "estout boottest"
    // If you need to "net install" packages, go to the very end of this program, and add them th
```

Scenario B

A simplified directory structure that correspond to scenario “B” looks like this:

```
directory/
       main.do
       scripts/
            01_dosomething.do
       data/
            data.dta
            otherdata.dta
```

Scenario C

Sometimes, we encounter Scenario C, which adds an additional level. A simplified directory structure that correspond to scenario “C” looks like this:

```
directory/
           step1/
           scripts/
               main.do
               01_dosomething.do
       step2/
               scripts/
                   othermain.do
                       01_analysis.do
       data/
           data.dta
           otherdata.dta
```

Note: you would place the `config.do` in **all** directories that have some sort of `main.do`.

Example

- A main .do file is in the main directory, and you have placed `config.do` in the main directory. The package `estout` and `ivreg2` need to be installed:

```
/* Template config.do */

local scenario "B"  // around line 48
*** Add required packages from SSC to this list ***
local ssc_packages "estout ivreg2"
    // Example:
    // local ssc_packages "estout boottest"
```

Directory paths for log files.

`config.do` creates a subdirectory and saves log files in the subdirectory. Let's say the current working directory path is the following, since Jira issue number is `AEAREP-9999` and openICPSR case number is `111111`

```
L:/Workspace/aearep-9999/111111
```

- line 97, `global logdir "${rootdir}/logs"` sets the following directory as a directory for log files: `U:/Workspace/aearep-9999/111111/logs`. Note that it will automatically use the `$rootdir` created earlier based on your choice of scenario.

```
global logdir "${rootdir}/logs"
cap mkdir "$logdir"
```

- The first time you run the code, no such directory exists. Therefore, the do file creates a new directory in line 98.
  - `mkdir` is a command to create a directory

Opening a log file with current date and time

Since we usually run the program several times until the code is completely debugged (!), we would like to record all the runs. Therefore, we record the initial time we start running the code and use it in the name of the log file.

- Lines 102-110 create a log file with the current date and time, and an identifier of who is running the code:

```
/* check if the author creates a log file. If not, adjust the following code fragment */

local c_date = c(current_date)
local cdate = subinstr("`c_date'", " ", "_", .)
local c_time = c(current_time)
local ctime = subinstr("`c_time'", ":", "_", .)
local ldilog = "$logdir/logfile_`cdate'-`ctime'-`c(username)'.log"
local systeminfo = "$logdir/system_`cdate'-`ctime'-`c(username)'.log"

/* global logfile */
log using "`ldilog'", name(ldi) replace text
```

- the last line starts the log file, with an internal name `ldi` which prevents collision with any log files opened by authors.

System information

We require system information as part of the replication package. This is because some commands are sensitive to the OS, STATA version, machine type, etc. Lines 113 calls a separate logfile so as not to clutter up the main logfile:

```
/* used only for system info */
log using "`systeminfo'", name(system) replace text
```

Package installation

As explained above, we often need to install packages. Even when the packages were installed in previous cases, it should be irrelevant to your current case, since we install those packages within our deposit directory so that we can verify the completeness of the replication packages.

```
* *** Add required packages from SSC to this list ***
local ssc_packages ""
    // Example:
    // local ssc_packages "estout boottest"
    //
    // If you have packages that need to be unconditionally installed (the name of the package differs fr
    // examples are morematа, egennmore, blindschemes, etc.
local ssc_unconditional ""

    // If you need to "net install" packages, go to the very end of this program, and add them there.
```

- Add list of packages in the quotation marks in **line 50**
    - line 52 provides an example.

```
local ssc_packages "estout"
```

- If there are packages that do not contain a command of the same name (examples are `egenmore` and `morematа`), add these packages to **line 56**.

```
local ssc_unconditional "egenmore"
```

- normally, no additional changes are needed!

```
/* install any packages locally */
di "=== Redirecting where Stata searches for ado files ==="
capture mkdir "$rootdir/ado"
adopath - PERSONAL
adopath - OLDPLACE
adopath - SITE
sysdir set PLUS     "$rootdir/ado/plus"
sysdir set PERSONAL "$rootdir/ado"       // may be needed for some packages
sysdir
```

- The `adopath` and `sysdir` commands (lines 120-124) redirect STATA to search for, and install ado files in the directories referenced.

```
/* add packages to the macro */

    display in red "============ Installing packages/commands from SSC ============="
    display in red "== Packages: `ssc_packages'"
    if !missing("`ssc_packages'") {
        foreach pkg in `ssc_packages' {
            capture which `pkg'
            if _rc == 111 {
               dis "Installing `pkg'"
                ssc install `pkg', replace
            }
            which `pkg'
        }
    }
```

- Lines 159-170 installs each package if there are packages listed and these packages do not already exist.

```
/* add unconditionally installed packages */
    display in red "================ Unconditionally installed packages from SSC ==============="
    display in red "== Packages: `ssc_unconditional'"
    if !missing("`ssc_unconditional'") {
        foreach pkg in `ssc_unconditional' {
            dis "Installing `pkg'"
            ssc install `pkg', replace
        }
    }
```

- Lines 172-180 do the same for packages that do not have a command with the same name.

```
/*============================================================================================*/
/* If you need to "net install" packages, add lines to this section                           */
    * Install packages using net
    * net install grc1leg, from("http://www.stata.com/users/vwiggins/")
```

- Some packages are not hosted on SSC. In that case, you have to use "`net install..`" instead of "`ssc install`". Write such `net install` commands after line 185, an example is given in line 185. **This is one of the RARE instances where you have to edit something in the latter part of `config.do`!!**
- In case where the authors provided ado files, we may need to specify this extra path, if the authors have not done so themselves (this varies across packages). Line 68

```
local author_adopath ""
```

will allow you to set that path at the top of the file. Lines 132-134 will implement this:

```
if "`author_adopath'" != "" {             // The author adopath variable is filled out
    adopath ++ "$rootdir/`author_adopath'"
}
```

How to deploy config.do

If the replication package contains a `profile.do`… Click to hide

Reminder: wherever it says `main.do`, read: `profile.do`.

Copy the config file to the right location

[ACTION] Copy the `template-config.do` to the right location.

You should copy it (using Windows actions, or command line) and paste it into the folder where the `main.do` file is located. Change the name from `template-config.do` to `config.do`

The template is called `template-config.do`. In order to use it, copy it into the openICPSR folder (e.g. , `111111`) next to the author's main file, and rename it to `config.do`. If there is no `main.do` or similar, create one. See the next section for more details.

Include config.do

- If there is a main dofile, you should put the following line at the beginning of the `main.do`:

```
include config.do
```

Note

Do NOT include it in the individual code files that are being called from the `main.do`. This will lead to errors.

Notes

- If there is no main dofile, you should try to create one (see how in the next section), starting with the lines above, and ending with the `log close` command.
- If creating a main dofile is not possible, **and only then**, add the lines at the beginning and the end, respectively, of **each** code file.
- There will be cases where authors **create their own log files**. Do NOT comment out the log file creation here, as the named logfile will not conflict with any author-generated files.

Running Code in Stata

Although, there are plenty of ways to run code in Stata, our goal with these instructions is to show the easiest way to do it, by minimizing both the manual steps replicators have to go through and the chance of making a mistake that prevents a successful run.

In essence, these instructions show how to deal with the three most common actions that replicators have to undertake when running Stata code:

1. Making sure that paths (i.e., something like "`Mycomputer/Documents/Workspace/`") in the .do files (Stata scripts) reflect the appropriate location of code, data, and output in the computer where the code is run.
2. Installing user-written functions, programs, or packages that are necessary to do computations and produce tables/figures.
3. Creating .log files (files that record, in this case, Stata output) of the replication attempts.

Step 1: check for a "main" .do file

> **[ACTION] Check the README or the repository and determine if a main .do file was provided.**
>
> A main .do file is a Stata script that will call, in the correct sequence, all the programs necessary to construct analysis datasets, do all computations, and produce figures and tables. If a main do file exists, it should be mentioned in the README. In most cases, running a single main do file is sufficient to complete the reproduction. In general, a main script does not need to be a .do file. However, we will focus on cases where all work done in Stata is reduced to executing a single .do file.

When a main .do file is provided

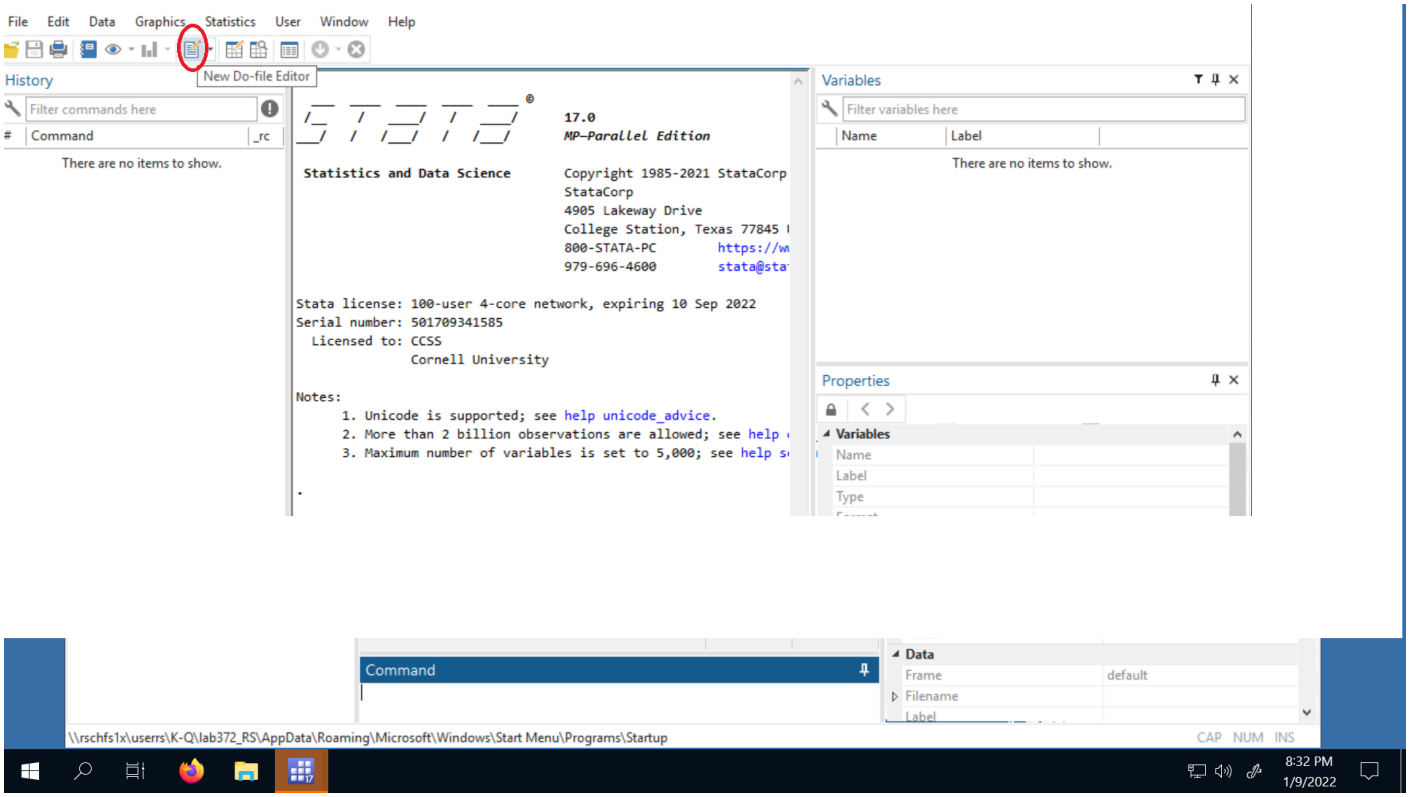
If there is a main do file, continue with Step 2.

When a main .do file is NOT provided

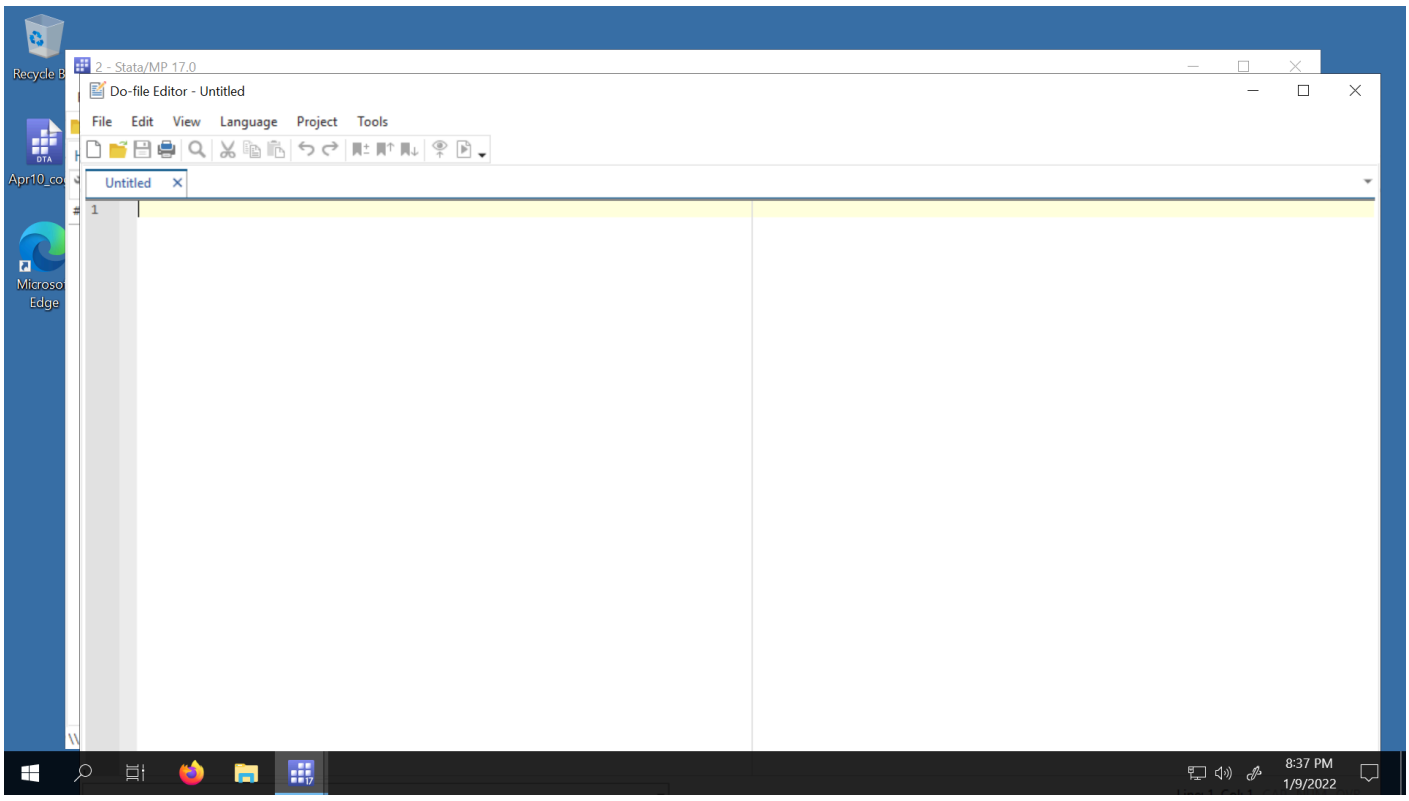
> **[ACTION] If a main .do file is not provided, you should create a one.**
>
> Identify all the programs (do files) that need to be run, and create a main file. This is easy!

To create a do file follow the following steps:

1. Check that the README for specific instructions about the order in which each program is supposed to be run. If there are no such instructions, or they are not obvious by the name of the programs, is probably best to not create a main do file.
2. Assuming that the sequence of programs is clear to you, open stata and click on the "New do file editor" (you can also work on Visual Studio Code):



To open the do file editor:



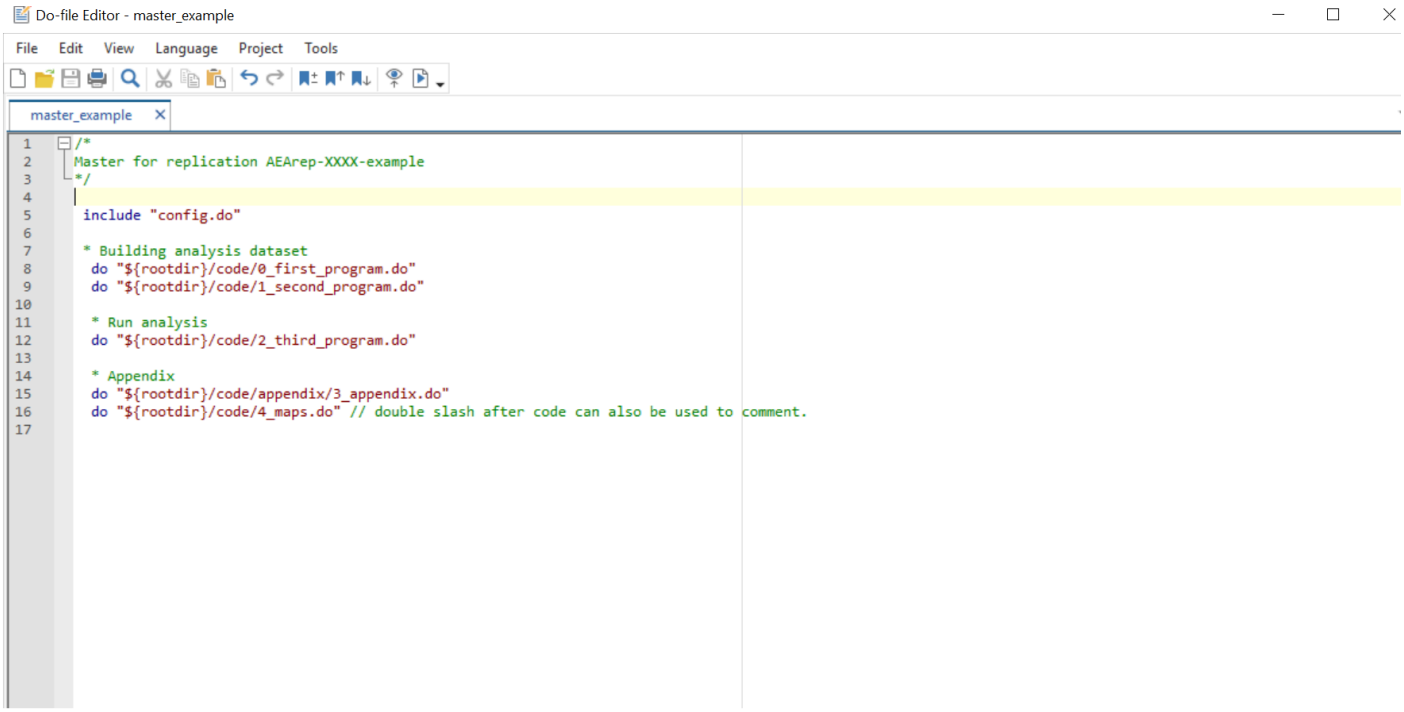
3. In the first line write `include "config.do"`
4. Write the command `do` and the (quoted) path of each program that needs to be run. Write them in the correct sequence.

Example:

```
include "config.do"

* Assuming scenario "A"

do "${rootdir}/code/0_first_program.do"
do "${rootdir}/code/1_second_program.do"
do "${rootdir}/code/2_third_program.do"
do "${rootdir}/code/appendix_code/appendix.do"
```

5. Save your main file.

At the end your main .do file may look like this:

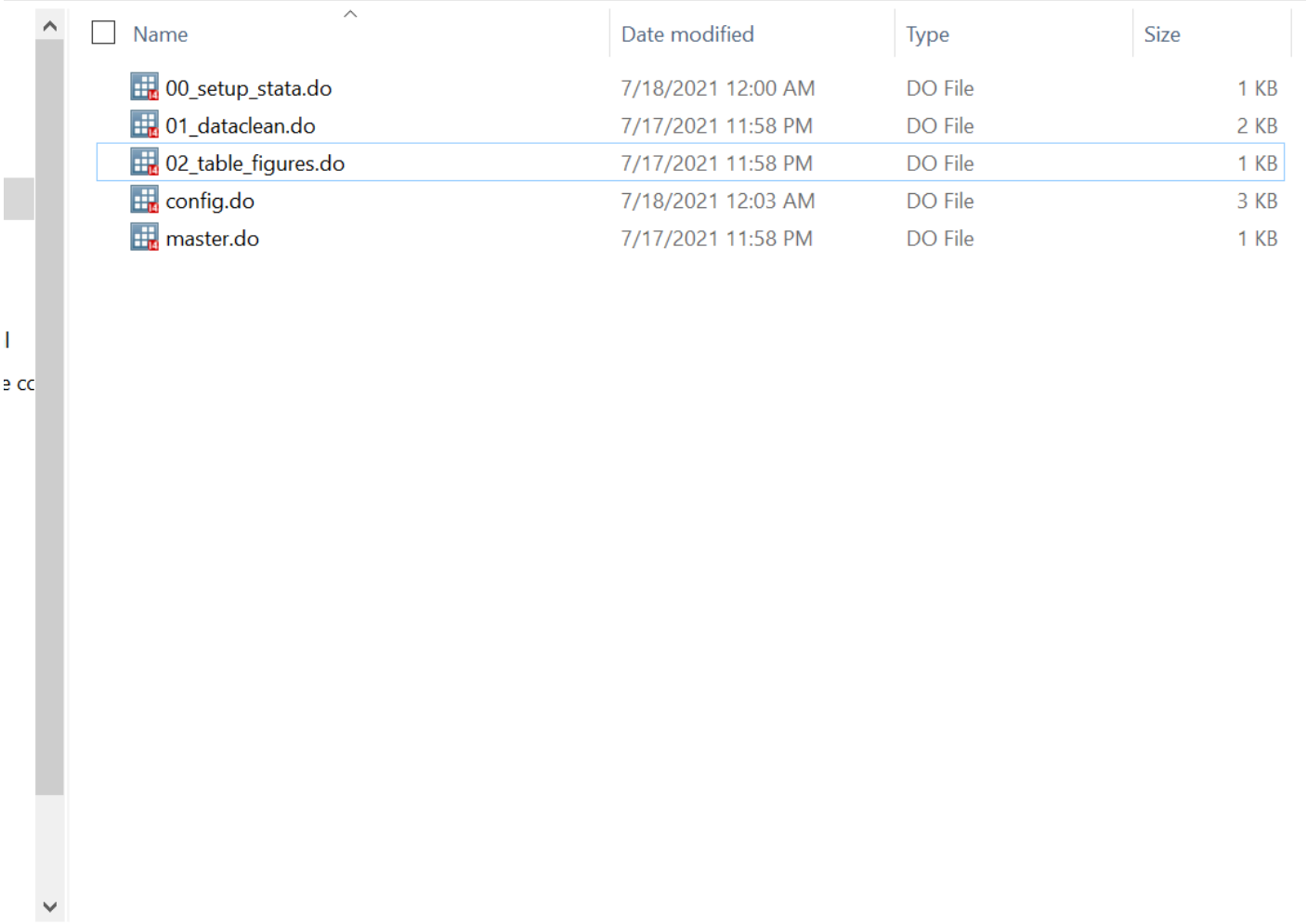


With your main do file done, continue with Step 2.

Step 2: place config.do where the main .do file is located

> **[ACTION] Copy the `template-config.do` to the right location.**
>
> You should copy it (using Windows actions, or command line) and paste it into the folder where the main file is located. Change the name from `template-config.do` to `config.do`

The folder with the code, whether is the root directory or a subfolder, should look something like this:



Step 3: include config.do in the main .do file

[ACTION] Add a call to the `config.do` file in the main .do file.

Open the main .do file. In the beginning, add the line to include the `config.do`.

(You already did this if you **created** a main .do file in Step 1.)

```
include "config.do"

/* This is Master do file */
```

Save.

- In some cases, authors will have code at the start of their dofile that does “housekeeping”, for example

```
        * Housekeeping
                cap log close
                set more off
                clear all
                estimates drop _all
                eststo clear
                macro drop _all
```

In this case, you should add `include "config.do"` **after** such commands, to avoid problems. Technically, the command `clear all` undoes many of the settings that our `config.do` does.

Step 4: Modify config.do

Note

`config.do` does 4 things:

- Creates a global variable called “rootdir” with the local path to the root directory.
- Creates a logs files.
- Sets a path to save the packages to be installed in the replication repository, and
- It allows you to install the packages simply by listing their names.

A crucial function of `config.do` is that it allows for the local installation of Stata packages, which is important for two reasons. First, it will enable us to check for the completeness of replication materials. Second, when running code in servers, we often do not have the necessary permissions to install Stata packages freely.`config.do` allow us to installed packages in the replication directory.

More information about modifying `config.do` can be found in “Using Config.do”.

Step 5: modifying paths if necessary in author-provided code

[ACTION] Modify paths if necessary.

- Check the Readme and determine if (and where) the root directory should be modified.
- Open the .do file to be modified (probably the main .do file) and set the global variable `$rootdir` as the path.
- Save.

To run the code, we need to make sure that Stata can access the locally-saved data, access the packages that will be installed, and save the output in the computer where you are running the code. To do that, we often need to change some directory paths defined in the .do files provided. This step may vary in each replication package, so you need to look at the README instructions closely. Some packages may not require any change, while others may require a little more work.

However, the typical case will only require one modification, either to the main .do file or to a program called by the main .do file, where you define the path of the location of the replication package. This location is what we refer to as the “root directory”. Once this change is made, the code provided (if it follows good practices) will define every other path relative to the root directory.

Example

In the author's main file, a global variable "maindir" defines the path of the root directory as:

```
clear all

/* This is Master do file */

global maindir "C:\Users\Author\Dropbox\Project1" // this is the path to the repository
global data "$maindir/data" // path to data folder
global figures "$maindir/figures" // path to figures folder
```

You would add `config.do` and change the global. After the change:

```
clear all
include "config.do"

/* This is Master do file */

global maindir "$rootdir" // this is the path to the repository
global data "$maindir/data" // path to data folder
global figures "$maindir/figures" // path to figures folder
```

Step 6: Modifying the main .do file

> ⚠ **You may have to modify some additional code in the `main.do` file.**
>
> If the `main.do` provided by the authors contains `run` commands, you must replace them with `do`.
>
> - `run` will hide much of the output.
> - `do` will show output, in particular when code goes wrong.

So if the `main.do` looks like this:

```
**************************

run "${do}/2_transform_data.do"

**************************************
* 3 * Append Denmark, Norway, Finland and Sweden
* depends on the previous step
**************************************

run "${do}/3_appending.do"

**************************************
* 4 * Prepare the data for the ridgeline plots and maps
* depends on the previous step
**************************************

run "${do}/4_ridgeline_prep.do"
```

(etc.), then you must modify it so it looks like this:

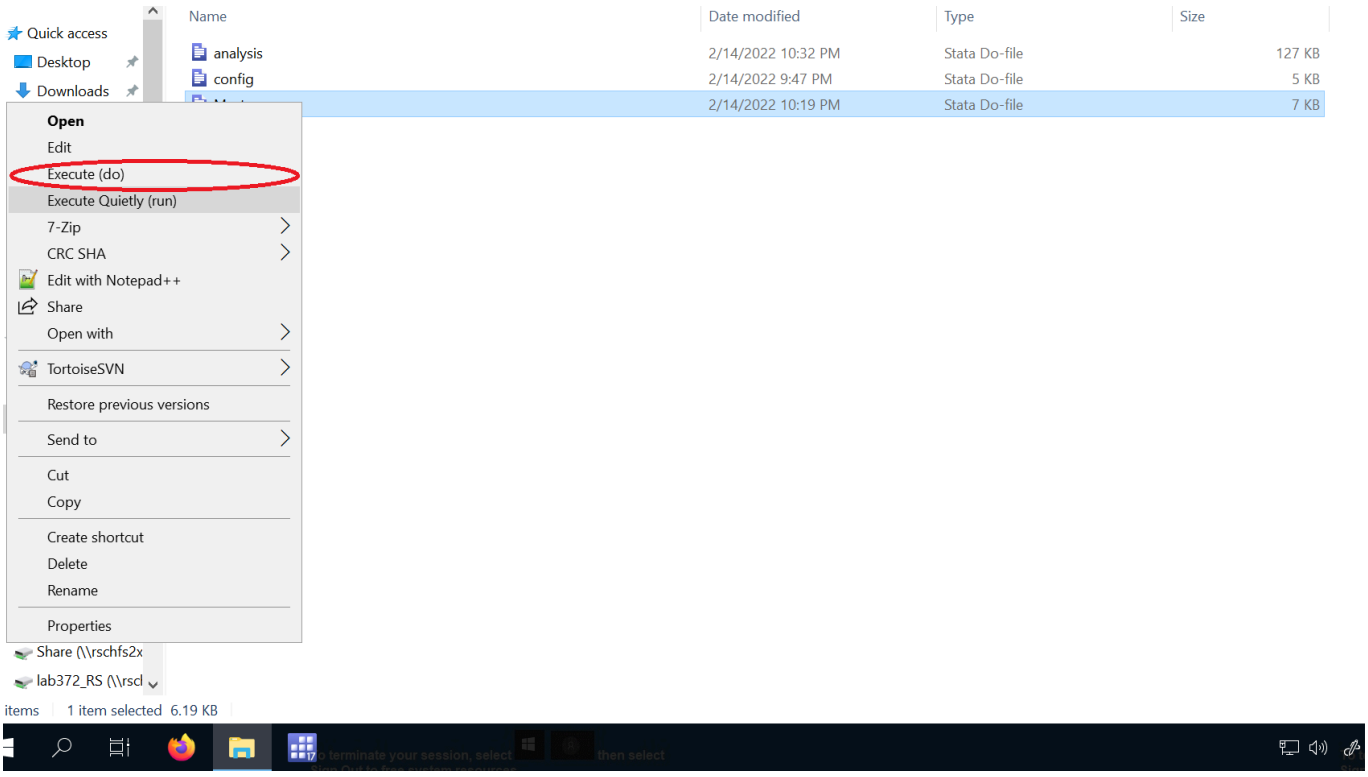
```
*************************

do "${do}/2_transform_data.do"

***************************************
* 3 * Append Denmark, Norway, Finland and Sweden
* depends on the previous step
***************************************

do "${do}/3_appending.do"

***************************************
* 4 * Prepare the data for the ridgeline plots and maps
* depends on the previous step
***************************************

do "${do}/4_ridgeline_prep.do"
```

Save it, and proceed to the next step.

Step 7: Run the Code

Windows | **Mac/Linux** | **Windows Double-click** | **Windows terminal**

> **[ACTION] Right click** on the master .do file and select the option `Execute (do)`.



This option will set the working directory to the location where the `master.do` is. It opens Stata and will show the processes in the Stata window.

Checking for a complete run, debugging and running the master in pieces

After running the code, the log files will need to be checked for a complete run. **Use Visual Studio Code to open and inspect log files**. Any bugs that prevents a complete run will also show up in the log files.

If a you find a bug that is simple enough to fix, you can make changes to the do files. Then, you can right click on the master file and select `Execute (do)` as this option will open Stata, allowing to run the code interactively.

If you decide the code needs to be run in pieces (this is NOT ideal)

- In the master .do file, you can comment out (using the symbol *) the programs that are not to be run and save the master.
- Then, you can right click on the master and select the option ``Execute (do)`.

When debugging is complete, you can uncomment all programs in the master and make a clean run, using again `Execute Quietly (run)`.

> Consider how much time a complete run would take before you run everything one last time. If it would take too long, you may want to skip a complete run, but ensure that you have log files for all partial runs. Make a note of this in the report.

Using scan_packages.do

When setting up to run the code, we ask you to check the completeness of the information on system requirements. Often, authors do not list out packages they installed that are not default packages in STATA. The authors should list them in the README (even when they provide ado files!), but it does not always happen. To help you identify these packages, we provide an useful tool for this exercise.

> **Note**
>
> As of Jan 2023, this will already have been run by the Pipelines, with output in `generated/`

- Locate a directory named “tools/Stata_scan_code/”.
- Change the following command in line 11 with your system information:

  ```
  global codedir "XXXCODEDIRXXX"
  ```

  You should locate the directory where the codes are (typically `112233`, the openICPSR space number):

  ```
  global codedir "../../112233"
  ```

  This will be the directory where the output excel file will be saved.

- Execute the dofile.
- Locate the file “`candidatepackages.xlsx`”, use the information there, and remember to push the file to the repository.

Stata on Linux

> **BioHPC specific**
>
> On BioHPC, Stata is only available on ECCO nodes, and is not described in the general software notes. See Stata notes for BioHPC using ECCO. Follow the notes on how to set up `modules` at https://labordynamicsinstitute.github.io/ecco-notes/docs/biohpc/software.html#using-module.

Finding Stata

- There are multiple versions of Stata on BioHPC: 14, 16, and 18
- You should always use `stata-mp`
- Stata is not "in the path", so you should do one of the following:
  - Reference the version of Stata explicitly: `/usr/local/stata16/stata-mp` or `/usr/local/stata18/stata-mp`
  - Add Stata to the path:

```
export PATH=/usr/local/stata16:$PATH
```

and then simply use `stata-mp` in the same session (the `PATH` gets reset once you log out from a session or close the terminal.

TMP directory

Ensure that the `tmp` directory being used is running on the BioHPC `/workdir` space, by running the following commands before executing your program(s):

```
export STATATMP=/workdir/netid/tmp
mkdir $STATATMP
```

before launching Stata.

Debugging Stata

Also check the LDI Replication Lab Wiki!

Basic debugging

Ado files not found

Issue: When an error message such as `xml_tab command not recognized` or `outreg2 command not recognized` appears…

Solution it usually means that you need to first install the command before running the .do file. Try

```
ssc install outreg2
```

or possibly

```
search this_great_command
```

followed by

```
net install  this_great_command, from(https://this_great_command.com/stata)
```

In some cases, this will be a reference to the Stata Journal packages, which do NOT have the same name as the command they provide:

```
search frmttable
```

yields:

```
Search of official help files, FAQs, Examples, and Stata Journals
------------------------------------------------------------------

SJ-12-4 sg97_5  . A programmer's command to build formatted statistical tables
        (help frmttable if installed) . . . . . . . . . . . . . . J. L. Gallup
        Q4/12   SJ 12(4):655--673
        create formatted tables from statistics and write them to
        Word or LaTeX files
```

Further below, you see this in a different way:

```
5 packages found (Stata Journal listed first)
---------------------------------------------

sg97_5 from http://www.stata-journal.com/software/sj12-4
    SJ12-4 sg97_5. Update: A programmer's command... / Update: A programmer's
    command to build / formatted statistical tables / by John Luke Gallup,
    Portland State University / Support:  jlgallup@pdx.edu / After
    installation, type help frmttable
```

Now you see how to construct this with a one-line net install command:

```
net install <pkgname>, from(<pkgurl>)
```

where you replace `<pkgname>` with the name of the Stata Journal package (here: `sg97_5`) and the `from()` command gets the URL that is listed in the help.

For `frmttable`, this yields:

```
net install sg97_5, from(http://www.stata-journal.com/software/sj12-4)
```

Warning

Please be sure to make a note of this installation in the REPLICATION.md!!

You must include all these commands in the `config.do`. For all `ssc install` commands, add them to the appropriate line at the top of the `config.do`.

Great tip:

When searching for a package, the Stata help browser will show in the address bar "`net describe this_great_command, from(https://this_great_command.com/stata)`. Simply copy and paste into your `config.do` and change `describe` to `install`.

unknown egen function wtmean()

Add package `_gwtmean` to the `config.do`, or `ssc inst _gwtmean`

unknown egen function ….

Add package `egenmore` to the `config.do`

struct_ ms_vcvorthog undefined (`ivreghdfe`)

You may find this with `ivreg2` or `ivreghdfe`:

```
struct ms_vcvorthog undefined
(817 lines skipped)
(error occurred while loading ivreghdfe.ado)
r(3000);
```

Solution: Ensure that you are using the `config.do` correctly, your `config.do` installs the relevant package (`ivreg2`, `ivreghdfe`, etc.), and that `ranktest` is also included. The code in the `config.do` that calls

```
mata: mata mlib index
```

should fix any additional issues.

Last estimates not found (`ivreghdfe`,`reghdfe`)

If you then get

```
last estimates not found
```

try installing from the Github website for `ivreghdfe` (see this link):

> **Warning**
>
> These commands should be included in the `config.do`!

```
* Install ftools (remove program if it existed previously)
cap ado uninstall ftools
net install ftools, from("https://raw.githubusercontent.com/sergiocorreia/ftools/master/src/")

* Install reghdfe
cap ado uninstall reghdfe
net install reghdfe, from("https://raw.githubusercontent.com/sergiocorreia/reghdfe/master/src/")

* Install ivreg2, the core package
cap ado uninstall ivreg2
ssc install ivreg2

* Finally, install this package
cap ado uninstall ivreghdfe
net install ivreghdfe, from(https://raw.githubusercontent.com/sergiocorreia/ivreghdfe/master/src/)

* Rebuild mata library index
mata: mata mlib index
```

This problem is *always* fixable, contact your Team lead if you run into persistent problems.

`assert_msg() not found` (`reghdfe`)

This seems to happen when an older version of `reghdfe` is run (i.e., with `version(3)` flag). One solution appears to be to

- install the latest version of `reghdfe` etc. from Github (see above)
- Add the command `ftools, compile` before the `mata: mata mlib index` line in the `config.do`

`reg2hdfespatial` and `ols_spatial_HAC`

These programs are usually included. If they are not, they can be installed using the following commands:

```
cap mkdir reg2hdfespatial
copy http://www.trfetzer.com/wp-content/uploads/reg2hdfespatial-id1id2.ado reg2hdfespatial/reg2hdfespatia
copy https://www.dropbox.com/s/pf2vtvgqhjk7rc8/spatial_HAC.zip?dl=1 spatial_HAC.zip
unzipfile spatial_HAC.zip // should create a directory "spatial_HAC"
local pwd : pwd
adopath ++ "`pwd'/spatial_HAC"
adopath ++ "`pwd'/reg2hdfespatial"
```

Installing grc1leg

If you encounter the `grc1leg` package, it cannot be installed by `ssc install grc1leg`. Use the following:

```
net install grc1leg, from("http://www.stata.com/users/vwiggins/")
```

Exporting Graphs

If a manuscript contains figures but the authors do not include code to automatically export those figures, there exists very straightforward code to fix this.

Stata package information: Stata Manual.

Syntax:

> graph export newfilename.suffix [, options]

Example:

> graph export Figure.pdf

> graph export Figure.png

I want to run my program in an older version of Stata?

Use the version command to emulate running the program on an older version. For example, to emulate Stata 12.1, use the following line:

```
version 12.1
```

This fix may be relevant when running into problems with the tobit regression. This does not always work. We have some other, somewhat more involved solutions, involving running Stata in Docker and have access to Stata versions 12, 13, 14, 16, and 17.

MAIN DO FILE

Running other do files

Some code has a main "`replication.do`" or "`main.do`" etc., which will have lines such as

```
do tables.do
```

This is a problem, because any parameters from main.do, such as changing where you install Stata packages (`sysdir PERSONAL "/path/to/somewhere"`) are lost…

Replace the above with

```
include "tables.do"
```

and you will be fine.

The difference is that the first one executes the `tables.do` in a somewhat fresh environment. The second one simply includes the code (as if copying-and-pasting) in the `main.do`, by reference, and thus inherits everything that is transient (such as tempfiles, etc.)

This may not always work - sometimes authors rely on that reset that happens when you use the `do tables.do` version.

Always use `include config.do` for `config.do` files however.

Running Stata from the command line

It sometimes is more convenient to run Stata from the command line. How to do this varies by operating system. In almost all cases, you need to know where Stata is installed.

- Windows: See this Stata page. In essence, on CISER nodes, navigate to the directory where the Stata program you want to run is located (here: `master.do`). Then, in the address bar of the file browser, type `powershell`. In the command line window, type

```
& 'C:\Program Files\Stata16\StataMP-64.exe' /b do .\master.do
```

will run the program, and create a logfile `master.log` (and any configured log files). Note that you can also right-click and request “Run (silently)”.

- Linux/MacOS: Similar to the Windows instructions, in a terminal, navigate (`cd /path/to/directory`) to the directory with the program you want to run. Most Linux systems will have the Stata command in the “path”, meaning it will be automatically found.

```
stata-mp -b do master.do
```

will run the program.

Stata Errors

`Varlist required` error:

- May occur if the author calls upon a varlist that s/he forgets to define. Check the other .do files to see if it has been defined elsewhere. If the definition is obvious (based on the article & data) insert a line of code defining it yourself.

`Too many values` error:

(SOLUTION?)

`Varlist not allowed` error:

(SOLUTION?)

`Maxvar too small` error:

- Run .do file using Stata-MP and add `set maxvar 32767` (maximum number of variables) to the `config.do`.

`File ___ could not be opened` error:

- May appear after the `esttab` command, or when Unix and Windows paths are mixed, or when running with Network paths (of style `\\network\path\to\folder`)

```
esttab using "$res/Cage_Rueda_Table_4.tex"
```

Solution 1 (most likely)

The directory `$res` has one or more elements that are missing (ie a subdirectory does not exist). For instance, if `$res` resolves to `D:/project/tables` but the subdirectory `tables/` does not exist, Stata will fail.

You need to create the directory, as part of the code (and make a note of this in the report). Add the following line above the problematic line, or in the `config.do`:

```
cap mkdir "$res"
```

(NOTE: this will *still* fail if there are additional parts of the path that do not exist.)

Solution 2 (preferred if Solution 1 does not solve the problem)

Verify that `$res` does not contain a Windows "network path" (`\\rschfs\users\A-K\fellow\project`). Rather, it should contain a "letter drive" (`D:/project`). On Unix systems, it may help to remove relative paths (`../project`) and use a fully-qualified path (`/home/user/fellow/project`)

> For reference: Filed with the package author on Github

If `$res` does actually contain a Windows network path, install the newest Github version of `estout`:

```
net install estout, replace from(https://raw.githubusercontent.com/benjann/estout/master/)
```

(this is also an optional install in the LDILab's `config.do`).

`Options IK, CCT, and CV have been deprecated` error:

May occur when using the `bwselect()` option after the `rdrobust` command. To fix this an older version of the `rdrobust` command must be installed. Install this by typing

```
net install st0366.pkg, from("http://www.stata-journal.com/software/sj14-4/")
```

into the `config.do`. *Don't forget to make a note of this in the Replication.txt!!*

Problems with “xi” command:

The outdated command xi may sometimes cause errors that cannot be remedied by using an older version of Stata. To update the syntax follow this example:

```
xi: gen i.year i.adjustedmsc
```

This line of code currently produces the error message “invalid name”. Correct this by dividing up the command into:

```
xi i.year
xi i.adjustedmsc
```

Problems with the “kdens” command:

May require the installation of `kdens` and possibly `moremata`.

.tex files cannot be opened

(this is a variant of the “`File ___ could not be opened` error”)

Basic idea: Try setting your global directory (however that is specified in the code) to reference the name of the local drive on CISER.

Instead of setting the directory as something like:
`\\rschfs1x\userRS\K-Q\msw274_RS\aea_workspace\stata-training-redo\aer_replication`

Replace the beginning part with (where anything after the U: is going to be specific to your project and set up):
`U:/aea_workspace/stata-training-redo/aer_replication`

Suggested solution

- Make the following modifications to the “config.do”
- modify, if necessary the last line (where “global rootdir” is set) to reference the local drive on CISER global rootdir “U:/Documents/Workspace/aearep-950/112343/replication_package” Note the use of “/” instead of “” - please always do so.
- if the use of esttab using “$rootdir/output/table.tex” does not work, then try the next solution. Otherwise, you are done. Note the use of the quotes! they are important in this case, because the path contains spaces!

Expanded solution:

- add a line global texdir “${rootdir}/output” (preferred) or the full path, to the BOTTOM of the config.do
- then replace esttab using “$texdir/table.tex”

If you do it like that, your modified programs will never have your personal path in the program code, only $rootdirandtexdir. Those are defined in the “config.do”, and are easy to change by the next replicator.

Specific to certain systems

RedCloud

Problems with temp files

Some (user-written) Stata packages do improper handling of Windows network paths. This affects the use of temp files.

Symptom

```
> graph ..., saving(`tempfile')
(file \\rschfs2x\share\statatemp\ST_7390_000002.tmp not found)

file \\rschfs2x\share\statatemp\ST_7390_000002.tmp saved as .gph format

...
> graph combine `tempfile` other_file.gph, ...

file \rschfs2x\share\statatemp\ST_7390_000002.tmp not found
```

(the error is not specific to the `graph` command). Note the missing double-backslash in the path.

Solution

You need to redirect the STATATMP directory to a non-networked drive. Use a terminal, and set the `STATATMP` variable before launching Stata. In all examples, we assume that the new directory is `U:\Documents\tmpdir`. If it doesn't exist, create it. The path to Stata may be different on different systems, use command line expansion to find the right path.

Bash | **Powershell** | **Command Prompt** | **Alternate**

```
export STATATMP="/c/Users/$USERNAME/statatmp"
mkdir "$STATATMP"
cd "/l/workspace/aearep-5697/203501/Data and do-files"
"/c/Program Files/Stata18/StataMP-64.exe" -b master.do
```

This normally succeeds.

CCSS Classic

When trying to install some extension Stata shows an error message such as: `_file c:\ado\plus\next.trk already exists_`

This issue usually occurs when running on a university system where you do not have rights to install to C:

Solution: Be sure you are including `config.do`, which has lines like

```
global basepath "/path/to/your/project/directory"
sysdir set PERSONAL "$basepath/ado/personal"
sysdir set PLUS "$basepath/ado/plus"
```

where you would create a “ado” directory in the “project” folder you are currently using for the replication (i.e., `..../10.1257/mac.12345/replication_netid/ado`). See the template-config.do.

Stata will then install the new extension into that directory instead of c:\ado\plus (source)

R-related procedures

In this section, we will show you a few things related specifically to running code reproducibly with R. For more general debugging tips for R and other computer languages, see our wiki.

Using config.R in R

In “Verification” stage, we ask you to keep a log of what you do. Moreover, authors often use packages that are not default programs of R. We provide `template-config.R` in the template repository you clone which addresses these problems. In this section, we will walk you through how to update the `config.R`, in the next section how to run R in a way that generates log files automatically.

> **Warning**
>
> If the author use `renv`, or `groundhog`, or other package management systems, you should NOT use `config.R`. Instead, follow the author’s instructions for setting up the environment, and proceed to running code in R.

Why do we have to install programs?

- R, or other statistical software, does not provide all the packages (or “libraries”, or “modules”) that enable or facilitate the analysis. Therefore, many user-written programs or extensions are publicly available for downloads. For R, this is most often comes from CRAN, but the specific “mirror” of CRAN that is used may vary. You install packages with something like `install.packages("package name")` or multiple packages with `install.packages(c("package1","package2"))`. You might also see the use of `remotes::install_github("r-lib/conflicted")` (to install a package from Github) or `install_version("devtools", "1.11.0")` to install a specific version.
- We differ in installation process from many others in the sense that, we want to install programs in a specified directory that is NOT a system directory.
    - This is to ensure that the set of packages used by replication package is complete. A complete replication package should be stand-alone, regardless of packages installed elsewhere in the machine that program is run on.

Explaining template-config.R

Start by copying the `template-config.R` into the authors’ code directory.

Installing packages

Any libraries identified by the authors should be listed here, unless the authors already provide a setup program, or lines that install these packages.

```
global.libraries <- c("foreign","devtools","rprojroot")
```

For instance, if the authors say you need `ggplot2` and `nonsenseR`, then add them to this line (and remember to keep case exactly as the authors provide it, so `nonsenser` is not the same as `nonsenseR`).

```
global.libraries <- c("foreign","devtools","rprojroot","ggplot2","nonsenseR")
```

S-drive, L-drive

In some cases, authors provide us privately with data that is not part of the public replication package (the part on openICPSR is generally public). We put this on the L-drive, or what used to be called the S-drive. Put the location of that here, if any:

```
s-drive <- "L:/Workspace/aearep-9999-implicit-nda"
```

> **Note**
>
> If you are working on Windows (e.g. CCSS-Cloud) then you would need to use `/` or `\\` to write filepaths or use the file.path() function. So, for example, the above would become:
>
> ```
> s-drive <- "L:\\Workspace\\aearep-9999-implicit-nda"
> ```

Wherever the author later references the confidential data, you can insert this placeholder, for instance:

```
# original author reference
# ols.data <- readRDS("data/confidential/analysis.Rds")
# you change it to
ols.data <- readRDS(file.path(s-drive,"data/confidential/analysis.Rds"))
```

Directory paths for log files, libraries, and other things

`config.R` creates a subdirectory for log files, but does not automatically create the log file (in contrast to Stata). Here, you should add any additional directories that your debugging identifies as being necessary. Do write any path names with `/`, not `\`, and leave the directory names already listed untouched.

```
create.paths <- c("logs","libraries")
```

For instance, if the authors state that output should be written to “outputs”, you can add

```
create.paths <- c("logs","libraries","outputs")
```

The `config.R` will create these directories if they do not exist, later on.

Base (root) directory

The base directory (or here, `rootdir`) is the directory that contains the replication package, as intended by the author. How do you figure that out?

Example 1:

```
aearep-9999/
    123456/
        Replication-package/
            code/
            data/
            README.pdf
```

In this case, the base directory is `aearep-9999/123456/Replication-package/`.

Example 2:

```
aearep-9999/
    123456/
        code/
        data/
        README.pdf
```

In this case, the base directory is `aearep-9999/123456/`.

You do not actually need to hard-code this in the `config.R` file. We will use a package called `here`, which can detect this automatically, **if** some files are present:

- The author has a `rproj` file - it will take that as the base directory.
- The (hidden) file `.here` is present - it will take the directory that contains that as the base directory.

The `here` package will get confused by the presence of **our** git setup, so if the two above files are not present, we need to manually create the latter:

```
cd 123456/Replication-package
```

or

```
cd 123456
```

and then

```
touch .here
git add .here
```

depending on the case. Now R will set the root directory correctly.

> **Note**
>
> If for some reason that does not work, simply override the automatic detection, by setting the `rootdir` manually, using `/` or `\\` as appropriate for your OS:
>
> ```
> rootdir <- "C:/user/Workspace/aearep-9999/123456/Replication-package"
> ```

Package installation source

As mentioned earlier, there are multiple sources for the R libraries. We generally use the Posit Package Manager (PPM), which provides a snapshot functionality. We pick a date that is close to the date you run this, check whether it is a weekday (because for some reason, PPM does not take snaphshots on weekends), and then configure R to look there for packages. This is both faster and more reliable - mostly. It can sometimes fail.

```
posit.date <- Sys.Date() - 31
# posit.date <- "2020-01-01" # uncomment and set manually if the above does not work
```

If you have to re-run this multiple times, and add on packages, this might get out of sync with the first time you ran it, so you might adjust the `posit.data` manually.

Installing packages

If the author's code does not provide install commands, you will need to add any missing packages to a particular location in the `config.R`:

```
# global.libraries <- c("devtools","rprojroot")
# For example, you can add on two additional ones:
global.libraries <- c("devtools","rprojroot","readxl","ggplot2")
```

This will then install the libraries locally.

System information

We require system information as part of the replication package. This is because some commands are sensitive to the OS, R version, machine type, etc. We use the `sessionInfo()` command to get this information.

How to use config.R

Rename the config file.

The template is called `template-config.R`. In order to use it, rename it to `config.R` and move it into the right folder. If there is a **main** file created by the author, put it next to that. If there is NOT a **main** file, put it into the folder that the author says to run the code from.

> **[ACTION]** Check the README or the repository and determine if a master R file was provided.

Include config.R or create main.R

If there is a main file…

If there is a main file (say `main.R`), you should put the following line at the beginning of the `main.R`:

```
source("config.R", echo = TRUE)
```

> **Caution:**
>
> If there are lines such as `rm(list=ls())` at the start of the `main.R`, you should put the `source()` statement AFTERWARDS (otherwise all the information in the config.R would get cleared):
>
> ```
> rm(list=ls())
> source("config.R", echo = TRUE)
> ```

```
# Clear the workspace
rm(list = ls())

# insert config.R
source("config.R", echo = TRUE)
```

If there is NO main file…

- If there is no main file, rename `config.R` to `main.R`, and add the authors' code files to the end:

```
source("01_data.R", echo = TRUE)
source("02_analysis.R", echo = TRUE)
source("03_figures.R", echo = TRUE)
```

> **Caution:**
>
> If there are lines such as `rm(ls())` in the various files provided by the author, you should comment them out:
>
> ```
> # rm(ls())
> ... (rest of code)
> ```

Running Code in R

Although, there are plenty of ways to run code in R, our goal with these instructions is to show the easiest way to do it, by minimizing both the manual steps replicators have to go through and the chance of making a mistake that prevents a successful run.

Why do we need log files?

- Log files record each step of the analysis and its results as a text. It also records error messages if you encounter any error upon running the code.
- There are other purposes to have log files, but for us, it is to communicate with other team members.
  - When a replicator submits the report, a preapprover (and an approver) needs to verify how the code ran. It is to ensure that any discrepancies we find are not due to mistakes on our end.
  - A log file is crucial for this verification. Otherwise, preapprovers and approvers have to run the code again to verify which is not an ideal use of time, nor an efficient way to process the case.

> **The following instructions are for Windows.**
>
> See Instructions for BioHPC for running R code generally on Linux, and specifically on BioHPC.

Step 1: check for a “main” .R file

A `main.R` file is a R script that will call, in the correct sequence, all the programs necessary to construct analysis datasets, do all computations, and produce figures and tables. If a `main.R` file exists, it should be mentioned in the README. In most cases, running a single `main.R` file is sufficient to complete the reproduction. In general, a main script does not need to be a .R file. However, we will focus on cases where all work done in R is reduced to executing a single .R file.

> **Note**
>
> If there is no `main.R` file, you should create one. See previous section.

Step 2: Run the Code

We will run R from the command line to create log files. We will use the `main.R` file to run the code.

> **Note**
>
> If the R code fails, you will see this in the `Rout` file, which you can open in VS Code. It is OK to debug using Rstudio, but the final (clean) run after all debugging, or any run that needs outside help, needs to be logged, and run from the command line.

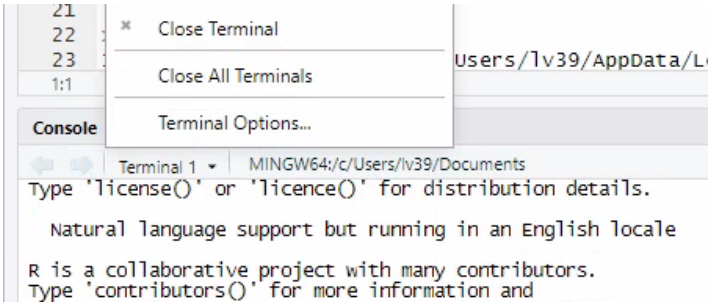
Setting up the environment to run the code

Windows | **Mac/Linux** | **BioHPC RStudio Server**

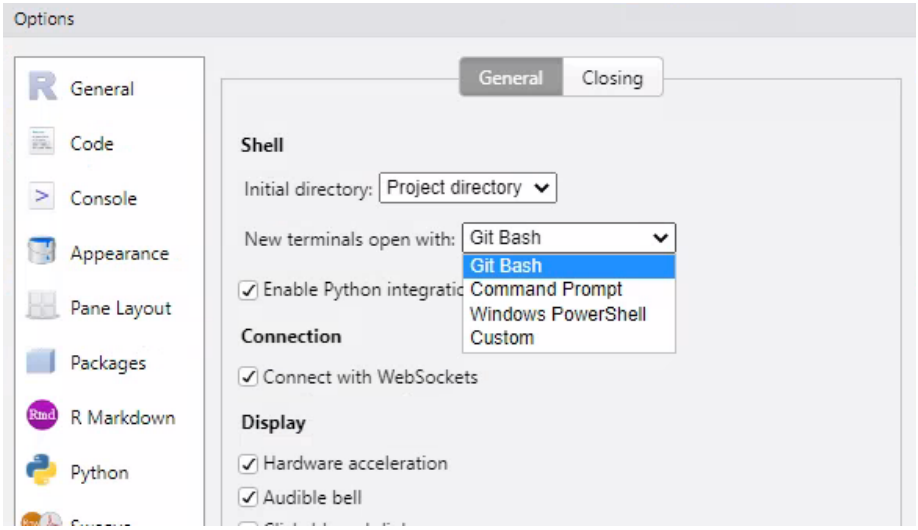
> **Tip**
>
> The most convenient way to run from the command line is to use Rstudio. Open Rstudio, and configure the **Terminal** as follows:

- Click on the Terminal tab in the bottom left corner of Rstudio

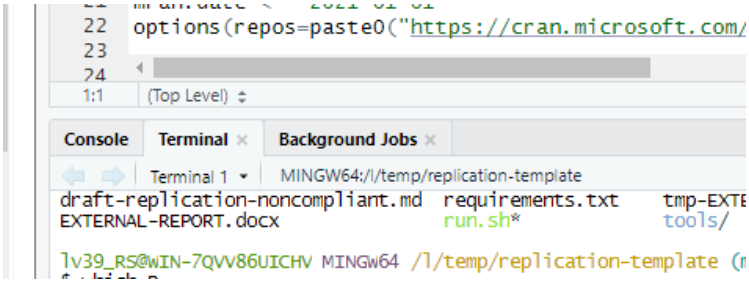
- Click on the downward-arrow to open up the menu for Terminal. Select `Terminal Options...`



- Select “Git Bash” from the options.



- Click “OK”
- Now open up the Terminal tab in the lower-left corner of the RStudio window:



Run the code

Once the above is done, running R is simple:

```
R CMD BATCH --verbose --vanilla main.R main.$(date +%F_%H-%M-%S).Rout
```

This will create a `main.(DATE).Rout` file, which you can open up in VS Code. You must commit this file to Bitbucket. Every run will create a new `main.(DATE).Rout` with a slightly different date-stamp.

Note

If the author's main filename has spaces in it, you will need to use quotes when inputting the filename into the bash command to say `R CMD BATCH "Main File.R"`

If the author uses `Rscript` ... Click to hide

… then use the following command line (adjusting the authors' accordingly)

```
RScript  --verbose main.R > main.$(date +%F_%H-%M-%S).Rout 2>&1
```

> Consider how much time a complete run would take before you run everything one last time. If it would take too long, you may want to skip a complete run, but ensure that you have log files for all partial runs. Make a note of this in the report.

Possible failures

`main.R` and `.Rprofile` are not in the same directory

If the `main.R` file is not in the root directory (i.e., the same directory as the `.Rprofile` file), then you need to use a workaround. Call the `main.R` file from a wrapper script in the root directory, or using the source command directly:

```
R  --no-restore -e 'source("scripts/main.R")' main.$(date +%F_%H-%M-%S).Rout
```

Use of Rstudio API

We sometimes see authors use the Rstudio API. In most cases, this will prevent the code from running in the `R CMD BATCH` method. Also in most cases, this can be remedied easily.

For instance, the following code leverages the Rstudio API to figure out the location of the code:

```
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
```

If using the `config.R` setup, this can be easily replaced with

```
setwd(rootdir)
```

or

```
setwd(file.path(rootdir,"code")
```

(if the file in question is in a subdirectory of the rootdir)

Failure due to `renv`

The package management system `renv` needs to be initialized. Possible failures include:

- the required `.Rprofile` file is missing (it is hidden from view in the file browser, but it must be there).

Solution: create a minimal `.Rprofile` file in the root directory with the following content:

```
source("renv/activate.R")
```

- the `renv` environment was not initialized.

The first time running code on a system, `renv` needs to restore packages.

Solution: Add the following line to the authors' `main.R`, before any other code that uses packages:

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

You might also simply need to invoke R (in the same directory as the `renv` setup) and run `renv::restore()` once manually.

Failure due to `segmentation fault`

On CCSS Cloud, running R code that requires a high amount of memory can cause R to fail.

Solution 1: Check how much memory storage is available in CCSS Cloud in the Task Manager and run the R file when memory is highest (usually at night)

Solution 2: Run the main file, but call only one R file at a time by commenting out all of the other R files. Rerun the main file for each additional R file that is included in the main file.

Running R on Linux systems

Linux allows us to run specific versions of R, but may also pose particular challenges when it comes to installing packages.

> **BioHPC specific**
>
> See R notes for BioHPC.

Specific versions of R

You may either need to use `module` or `docker` to run specific versions of R. Both can work on BioHPC.

> **BioHPC specific**
>
> Use `module avail` to see which versions of R are available. Use `module load R/4.2.1-r9` to load a specific version.

Installing packages

R Packages Through CRAN

R on Linux will usually install packages from CRAN, and compile from source. This may be feasible to circumvent by using Package Archives, and defining the specific operating system you are using.

> **BioHPC specific**
>
> BioHPC uses `Rocky Linux 9`.

This is the default for `rocker/` provided Docker images.

R Packages through Conda

On BioHPC, installing packages through R can fail, with errors such as

```
------------------------------ ANTICONF --------------------------------
Configuration failed to find libgit2 library. Try installing:
 * brew: libgit2 (MacOS)
 * deb: libgit2-dev (Debian, Ubuntu, etc)
 * rpm: libgit2-devel (Fedora, CentOS, RHEL)
If libgit2 is already installed, check that 'pkg-config' is in your
PATH and PKG_CONFIG_PATH contains a libgit2.pc file. If pkg-config
is unavailable you can set INCLUDE_DIR and LIB_DIR manually via:
R CMD INSTALL --configure-vars='INCLUDE_DIR=... LIB_DIR=...'
-------------------------- [ERROR MESSAGE] ---------------------------
<stdin>:1:10: fatal error: git2.h: No such file or directory
compilation terminated.
--------------------------------------------------------------------
```

This occurs since libraries on BioHPC are often installed in directories which R does not expect. To solve this, we use conda. Conda handles dependencies of both R packages and other libraries. See installation details [here] [https://biohpc.cornell.edu/lab/userguide.aspx?a=software&i=574#c]. To create a conda environment, run the command

```
conda env create --name <name> r-base=<version>
```

- This creates a conda environment with the proper R version installed. After activating the environment, install R packages with the command

```
conda install conda-forge::r-tidyverse
```

R packages on conda often follow the format `r-<package>`. You can search for packages [here][https://anaconda.org/].

On Anaconda, R packages are often hosted by the R repository, and conda-forge. Packages from the R repository tend to be pickier about version compatibility than those from conda-forge.

Errors

Package not available

If you get

```
3: package ‘NameOfPackage’ is not available for this version of R
```

check

- the package may have been obsoleted, and removed from CRAN. You may need to use Package Archives, or versioned installers:

```
install_version("mypackage", "1.15")
```

- your version of R is too young (see above for specific versions)
- your version of R is too old (see above for specific versions)

MRAN no longer available

Some older packages may attempt to use MRAN - the “Microsoft R Application Network”. This was shut down on July 1, 2023 (see this notice).

Solution: Switch to Posit Package Manager (PPM).

Running R from other software

Sometimes, authors will provide code that runs R from within, say, Stata or MATLAB. This is equivalent to calling R from the command line, though the specific command line will differ by system.

> **Warning**
>
> If R is calling other software, then consult the other sections in this guide.

The example here works through a specific example where R is called from Stata. Adapt as necessary for other code.

An example

Stata code calls out to R. This may be done in the following way (partial code)

```
global r_path "/something/here" //Set path to R program (Adjust this to your local R installation path)
cd "$do_path"
shell "${r_path}/R" --vanilla <"$do_path/Figure1.R"
```

You need to figure out the path to R on your system.

Windows | macOS | Linux

> ⚠ **Shell is not `bash`!**
>
> On Windows, the `shell` command is not `bash`. It will call `cmd.exe` command instead.

The easiest way may be to navigate to where R is usually installed. This would be

- `C:\Program Files\R\`

with version-specific subdirectories, e.g.,

- `R-4.4.1`

The full path to the R software is then

- `C:\Program Files\R\R-4.4.1\bin\x64\R.exe`

So the above would then be (note the forward slashes, not backslashes)

```
global r_path "C:/Program Files/R/R-4.4.1/bin/x64" //Set path to R program (Adjust this to your local
cd "$do_path"
shell "${r_path}/R" --vanilla <"$do_path/Figure1.R"
```

Caution

Always verify that the code actually ran as intended!

Debugging R

See below, or our wiki.

Library installation problems

> You are not allowed to install packages in C: because you do not have administrator rights.

This will happen on computers where you are a guest (not the owner), for instance CCSS Cloud or RedCloud.

Solution:

Run the following in an R console (or the console in Rstudio):

```
dir.create(Sys.getenv("R_LIBS_USER"), recursive = TRUE)  # create personal library
```

Then restart R, and try the install again.

Using different versions of R

For Linux, see R on Linux.

For Windows and MacOS, when using Rstudio, see Rstudio help pages.

Note that this only works if there is, in fact, another R version installed. Click to hide

You can find older versions of R at CRAN. To install, simply run the program, select an installation location under `C:\Users\[YOUR USER NAME]\AppData`, and uncheck any additional checkboxes during the installation process. You can then **browse** from the `R Selection Dialog` mentioned at the Rstudio help pages to find that version of R.

Pathnames

Use **single** forward slashes everywhere (source):

For a Windows path that would read `F:\\rschfs1x\userRS\F-J\XXXX_RS\Documents\Workspace\app.20160004`, use

```
lest<-read.dta("F:/rschfs1x/userRS/F-J/XXXX_RS/Documents/Workspace/app.20160004/ddd_long.dta")
```

Even better, use the `file.path()` function:

```
rootdir <- "F:/rschfs1x/userRS/F-J/XXXX_RS/Documents/Workspace/app.20160004"
lest    <- read.dta(file.path(rootdir,"ddd_long.dta")
```

Even better better, use the `here()` function used in the `config.R` (see notes in the file on how to create the `.here` file).

```
rootdir <- here()
lest    <- read.dta(file.path(rootdir,"ddd_long.dta")
```

Package Dependency - if you want to use an earlier version of a package

The trick is to use the COPY of the CRAN repository at ~~Microsoft, called the MRAN.~~ Posit/Rstudio, called the "Posit Package Manager".

At the top of any R code add the following lines:

```
ppm.date <- "2019-09-01"
ppm.url  <- "https://packagemanager.posit.co/cran/"
options(repos=paste0(ppm.url,mran.date,"/"))
```

Note

Even better, the `config.R` already has lines like this, use them!

You should use a judicious "`ppm.date`", probably based on the date JUST prior to the release of the next version of R. You can look up those dates here.

So for instance, if you want the last MRAN date prior to the release of R 4.0, you might choose "2020-03-15" (mid-March, prior to the release of R 4.0). At that time, the latest R version was R 3.6.3.

You should then be able to install packages in a version that is compatible with R 3.6.3 from the MRAN.

All package install functions in R should work "as-is", unless they explicitly download github versions or similar ("`remotes::install_github()`" or similar).

Special note for BioHPC

On BioHPC, which runs a Linux version called "Rocky 9.0", you should use the following instead:

```
ppm.date <- "2019-09-01"
ppm.url  <- "https://packagemanager.posit.co/cran/__linux__/rhel9/"
options(repos=paste0(ppm.url,ppm.date,"/"))
```

This will (hopefully) install pre-compiled packages, rather than trying to compile every package from source, which is a LOT faster.

`trying to use CRAN without setting a mirror`

For example,

```
install.packages("reshape2")
Installing package into 'C:/Users/Mbrown_RS/AppData/Local/R/win-library/4.2'
(as 'lib' is unspecified)
Error in contrib.url(repos, "source") :
  trying to use CRAN without setting a mirror
```

If you get this error, you need to set a CRAN repository within R. The best way is to set it globally, once and for all (as per this note. You can do this by manually editing the `.Rprofile` file, or simply running the following command:

> **Note**
>
> Even better, this is already configured in the `config.R` - use it!

```
echo '# set the default repository
local({
  r <- getOption("repos")
  r["CRAN"] <- "https://cran.rstudio.com/"
  options(repos = r)
})
# create the local library if it does not exist
if ( !file.exists(Sys.getenv("R_LIBS_USER") ) ) {
   dir.create(Sys.getenv("R_LIBS_USER"),recursive=TRUE)
} ' > $HOME/.Rprofile
```

but see about the correct setting for `r["CRAN"]` here

‘lib = “/usr/lib64/R/library”’ is not writable

You will probably see this error on Linux, but possibly on Windows as well, when running a package installation command (`install.packages()`) for the first time ever for a particular version of R on that system. The reason is that the first time around, R will need to create a directory in your space, but cannot do so automatically, as it needs permission.

The solution is to run the same version of R, on that system, interactively.

```
R
>
```

You can then install any package, though an excellent choice are `here` and `renv`:

```
install.packages(c("here","renv"))
```

You will then be prompted to create a personal library, because the system library is not writable:

```
install.packages(c("here","renv"))
Warning in install.packages(c("here", "renv")) :
  'lib = "/usr/lib64/R/library"' is not writable
Would you like to use a personal library instead? (yes/No/cancel)
```

You should now say “yes”.

```
Would you like to create a personal library
‘/home/vilhuber/R/x86_64-suse-linux-gnu-library/4.2’
to install packages into? (yes/No/cancel)
```

You should say yes again. You **may** then be asked

```
--- Please select a CRAN mirror for use in this session ---
```

Select a CRAN mirror (`0-Cloud [https]` is a good choice), and proceed. R will then install the packages and their dependencies.

> **Note**
>
> However, in doing so, it also created `/home/vilhuber/R/x86_64-suse-linux-gnu-library/4.2`, which now exists. You should now go back to the original code you were running, and run it again, it should work.

Exit R, and proceed as above:

```
> q()
Save workspace image? [y/n/c]: n
```

rJava Issues

Below you will find details of a rJava error and how it was resolved.

rJava Installation Error

```
Error: Error installing package 'rJava':
================================
* installing *source* package 'rJava' ...
** package 'rJava' successfully unpacked and MD5 sums checked
** using staged installation
Generate Windows-specific files (src/jvm-w32) ...
make: Entering directory '/c/.../Temp/RtmpUVjaDY/renv-package-new-13b84f6a7cd0/rJava/src/jvm-w32'
dlltool --as as  --input-def jvm64.def --kill-at --dllname jvm.dll --output-lib libjvm.dll.a
gcc  -O2 -c -o findjava.o findjava.c
gcc  -s -o findjava.exe findjava.o
make: Leaving directory '/c.../Temp/RtmpUVjaDY/renv-package-new-13b84f6a7cd0/rJava/src/jvm-w32'
Find Java...
ERROR*> JavaSoft\{JRE|JDK} can't open registry keys.
ERROR: cannot find Java Development Kit.
  Please set JAVA_HOME to specify its location manually
ERROR: configuration failed for package 'rJava'
* removing 'L:/Workspace/aearep-XXXX/123456/renv/library/windows/R-4.4/x86_64-w64-mingw32/.renv/1/rJava'
install of package 'rJava' failed [error code 1]
```

Potential Resolutions

The resolutions specified here are only suitable for Windows systems like CCSS-Cloud. For Linux, we will create documentation as we encounter and resolve errors.

First, confirm whether azulJava files are present in the locations specified in the examples below. They won't be on all systems. If you find the files then, at the beginning of the R script, place the following:

```
Sys.setenv(JAVA_HOME='L:\\common\\azulJava\\jdk\\jre')
install.packages("rJava")
library(rJava)
```

If that doesn't work, try:

```
Sys.setenv(JAVA_HOME = "L:/common/azulJava/jdk")
install.packages("rJava")
library(rJava)
```

MATLAB-related procedures

In this section, we will show you a few things related specifically to running code reproducibly with MATLAB. For more general debugging tips for MATLAB and other computer languages, see our wiki.

Adding `config.m` to MATLAB and path names

When you replace hard-coded filepaths in Matlab, please do the following:

Say the author has code like

```
xlsread('C:\Users\me\submission\AEJMacro\yesterday\data.xlsx')
```

You should do the following:

- Copy `template-config.m` to the directory of the Matlab script you are running, and call it `config.m`.
- Adjust the options.
- Add the line `config` to the main Matlab script.

Then

```
% Add this ONCE at the top of the program. Do NOT repeat it.
config
% wherever the hard-coded path appears, replace with
xlsread(fullfile(rootdir,'data.xlsx')
```

This works on any platform.

Running MATLAB without the desktop GUI and with log file

Windows | **Linux** | **Docker**

See these instructions for finding the Matlab binary on the system. However, this should work “out of the box” on CCSS-managed systems from the **Bash** prompt.

```
start matlab -nosplash  -batch  "addpath(genpath('.'));main;exit"  -logfile matlab-$(date +%F_%H-%M-%S
```

where `main.m` is the Matlab program you want to run (you omit the `.m` when calling it).

Note

This will exit, and first appear to do nothing for a few seconds. It will in fact open a Matlab window in the background (check your taskbar after a few seconds). At the end of the MATLAB code, you may need to manually exit MATLAB, but do so only when it is clear that MATLAB has ended!

Running Dynare from MATLAB

Dynare is often used as an extension from within MATLAB. Files are usually identified as `.mod`, though many other countries also use that extension.

Warning

Use the precise version of Dynare specified by the authors. If that version is not located on `L:\common` (CCSS Cloud) or `/home2/ecco_lv39/software` or as a Docker file, then talk to your supervisor.

Obtaining Dynare

Dynare is available for download from https://www.dynare.org/download/. But pay attention to the details for the platform where you will run this!

CCSS Cloud | **BioHPC**

> **Do not attempt to install Dynare on CCSS or BioHPC computers.**
>
> However, it is not necessary to *install* Dynare, it is sufficient to unpack the ZIP files. Try the information below first:

If a specific version is mentioned and is not there, download it from the Dynare website - choose the **ZIP** version. Then unzip it to the `L: drive` (move it around if necessary so you get the same structure as in the example above, i.e. `L:\common\dynare\(SOME VERSION)\matlab`). Then include the `addpath` command as before.

Configuring MATLAB to recognize Dynare

- Be sure to include the `config.m` file as outlined in Configuring MATLAB!
- Inspect the last part of the `config.m`, it should have code much like the following:

```
%%% Dynare settings
%
% The following are possible Dynare settings. Uncomment the one you need.

% dynarepath = "/Applications/Dynare/4.6.1/matlab"
% dynarepath = "L:\LDILab\dynare\dynare-4.5.7\matlab"
%dynarepath = "L:\common\dynare-4.5.7\matlab"

%
% Then uncomment the following line:
%
%addpath(genpath(dynarepath))
```

- You will want the path for the system you are working on. If CCSS Cloud, use the `L:` path. If on BioHPC, you will need to add a path like `/home2/ecco_lv39/software/dynare-x.y.z`.
- Then uncomment the `addpath()` statement at the end.

Your modified `config.m` should look somewhat like this:

```
%%% Dynare settings
%
% The following are possible Dynare settings. Uncomment the one you need.

% dynarepath = "/Applications/Dynare/4.6.1/matlab"
% dynarepath = "L:\LDILab\dynare\dynare-4.5.7\matlab"

dynarepath = "L:\common\dynare-4.5.7\matlab"

%
% Then uncomment the following line:
%

addpath(genpath(dynarepath))
```

The Matlab program should now run, and call out to the Dynare software as needed.

Additional info

- Building Docker image with MATLAB Toolboxes
- Dynare Docker images with MATLAB

Debugging MATLAB-related problems

The .m program is using a software extension to MATLAB called “Dynare”

See Using Dynare in MATLAB.

Mention of MEX programs, what do I do?

(or: MEX files provided for Linux/MacOS, but you need to run on Windows)

- There should be `.cpp` files provided - if not, they should be requested from the author
- From within Matlab, run `mex name_of_file.cpp` (though this should also be provided by the author).

MATLAB complains about a missing Toolbox

Check that the Path includes said Toolbox, see https://it.mathworks.com/help/matlab/ref/pathtool.html. It may also be that we are not licensed to use that Toolbox. Make a note of the error in the replication report.

MATLAB uses too much memory or too many CPU cores (parpool)

When running MATLAB on BioHPC (or other clusters), you will sometimes find that authors have hard-coded the number of CPU cores to use, or might ignore the amount of memory needed.

Code might look like this:

```
%% Parallelizing code

c = parcluster;
c.NumWorkers =30;
parpool(30)
```

The following code might be helpful in making this somewhat easier to debug:

```
%% Parallelizing code dynamically

% 1. Get the CPU count from SLURM environment
% SLURM_CPUS_PER_TASK is set when you use --cpus-per-task
num_cpus = str2double(getenv('SLURM_CPUS_PER_TASK'));

% 2. Fallback: If running locally or variable isn't set, default to 1 or a safe number
if isnan(num_cpus)
    num_cpus = 1;
    fprintf('SLURM variable not found. Defaulting to 1 worker.\n');
else
    fprintf('Detected %d CPUs assigned by SLURM.\n', num_cpus);
end

% 3. Initialize the pool
c = parcluster;
c.NumWorkers = num_cpus;
parpool(c, num_cpus);
```

Python-related procedures

In this section, we will show you a few things related specifically to running code reproducibly with Python and Jupyter notebooks. . For more general debugging tips for Python and other computer languages, see our wiki.

Python package installation

Most systems will have the standard Python package installer `pip` already installed, so you should be able to use it.

If `pip` is not available… Click to hide

Should for some reason `pip` not be installed, you can install `pip` into your user library using the procedures outlined at https://pip.pypa.io/en/stable/installation/#ensurepip. In a nutshell, for Linux

```
python -m ensurepip --upgrade
```

should work.

Best practice to reproduce a Python paper (Python environments)

You should create a Python environment that is dedicated to the project. See Anaconda instructions as one possible method, venv as another one, though others exist.

Native Python `venv`

Here's `venv` version in a nutshell (full guide). Open up a shell.

- Ensure `venv` exists:

```
pip3 install pyenv
```

- Create a new environment

```
python3 -m venv /path/to/new/virtual/environment
```

or if using relative paths

```
python3 -m venv env
```

which will create `/path/to/new/virtual/environment` or (relative to your current working directory) `env`. That directory will now contain all of your project-related Python packages.

To activate:

```
source env/bin/activate
```

On Windows Bash (depends on install)

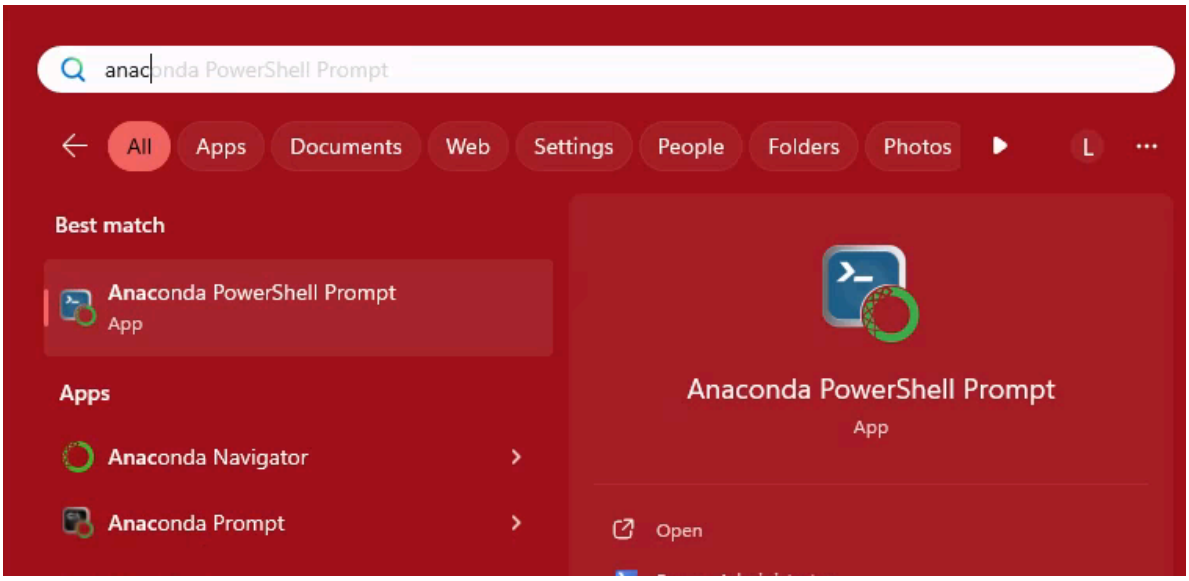
```
python -m venv env
source env/Scripts/activate
```

To deactivate:

```
deactivate
```

Anaconda method

On Windows, use `Anaconda Powershell Prompt` or `Anaconda Prompt` Click to hide



On BioHPC Click to hide

Follow instructions at BioHPC on how to install `miniconda` in your home directory, then add the line

```
source $HOME/miniconda3/bin/activate
```

at an appropriate location in the code (for instance, replacing `module load conda` in the SLURM batch file), or interactively.

In the shell of your choice, run

```
conda create -n (your_env_name)
```

where ideally, `(your_env_name)` is the name of the JIRA issue, in **lower case** (e.g., `aearep-123`).

Then, activate the environment:

```
conda activate (your_env_name)
```

To deactivate, run

```
conda deactivate
```

Making Python code dynamic

In general, Python code should not have hard-coded paths. Python programs are aware of their own location, and other directories should be relative to that. However, some authors may still follow (econ-specific) norms, and hard-code paths. Similar to our approach for Stata, R, and MATLAB, you can use the `rootdir` principle to make the code more portable/dynamic.

In that case, do the following:

Say the author has code like

```
xlsread('C:\Users\me\submission\AEJMacro\yesterday\data.xlsx')
```

At the top of the program, add the following lines:

```
import os
rootdir = os.path.realpath(__file__)
```

Then, wherever the hard-coded path appears, replace it with:

```
xlsread(os.path.join(rootdir,'data.xlsx'))
```

Alternatively, if the author uses a change of working directory, then

```
import os
os.chdir(os.path.dirname(os.path.abspath(__file__)))
```

would change the working directory to the location of the file being run.

Installing packages

Sometimes, authors will list the packages they used. There are a few options:

`requirements.txt` file | `environment.yml` file | List of packages

If the authors provide a `requirements.txt` file, you can install all the packages at once. From an appropriate terminal, run:

```
pip install -r requirements.txt
```

Using Anaconda Package manager

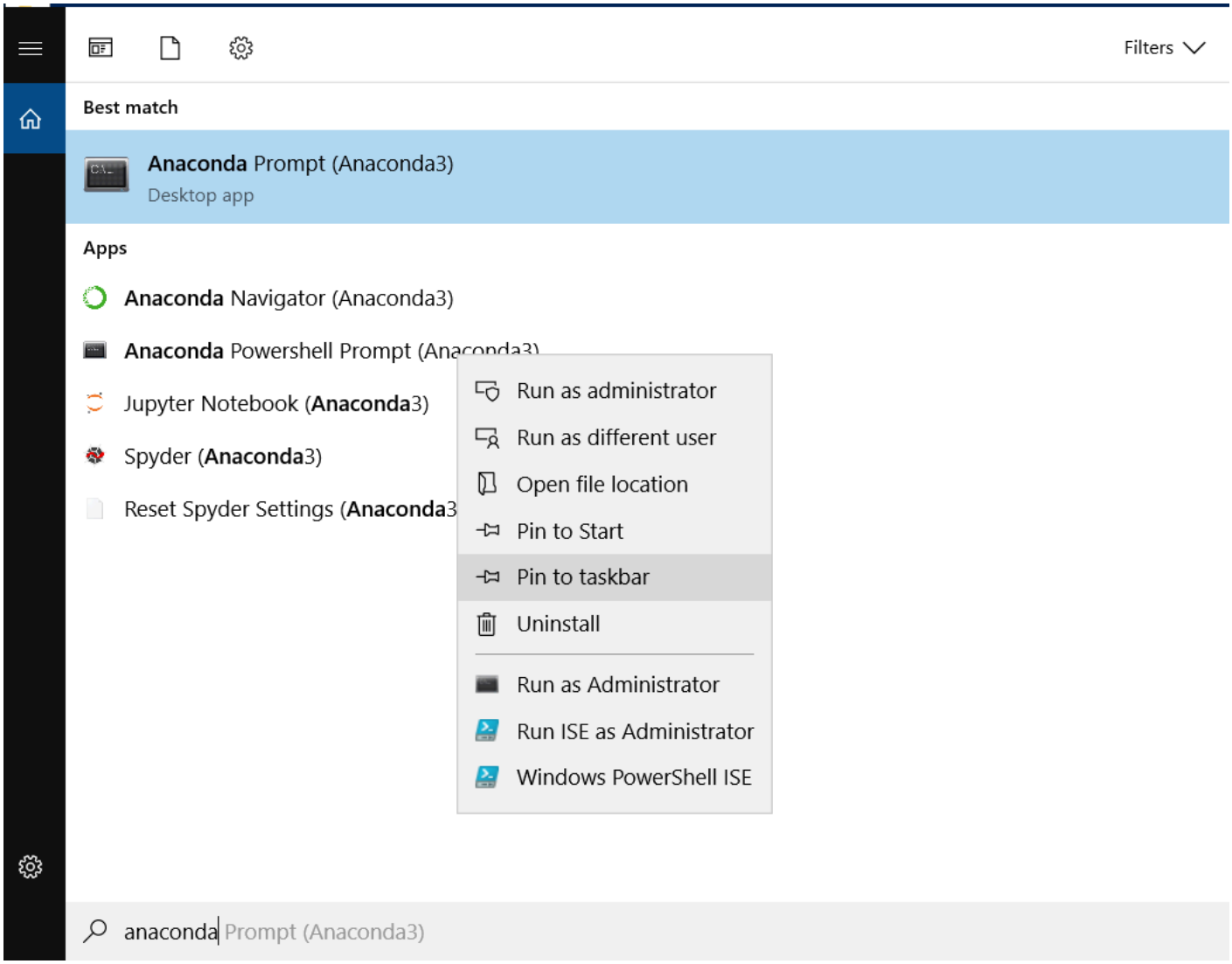
This is known to work on CISER (CCSS-Classic).

This may not be the way it works on CCSS-Cloud. Click to hide

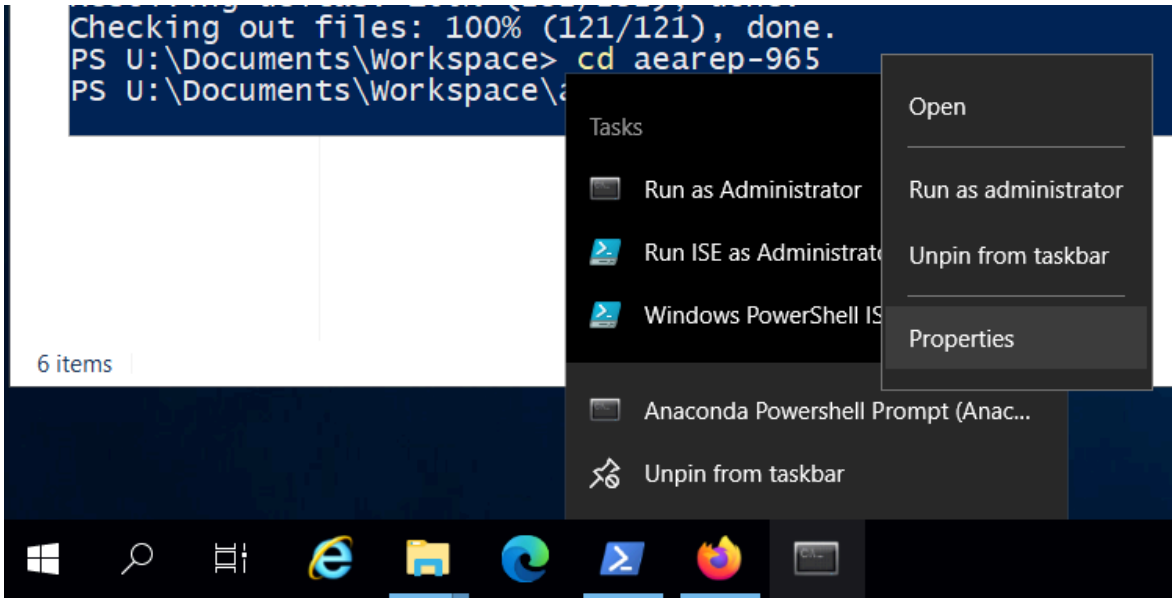
Needs an update

If using the default “Jupyter” link in the Start Menu, the working directory won’t be right. Assuming that you have set your Workspace to `L:\Documents\Workspace`, the following will create a Jupyter Notebook in the right location (thanks to Louis Liu for creating this Howto)

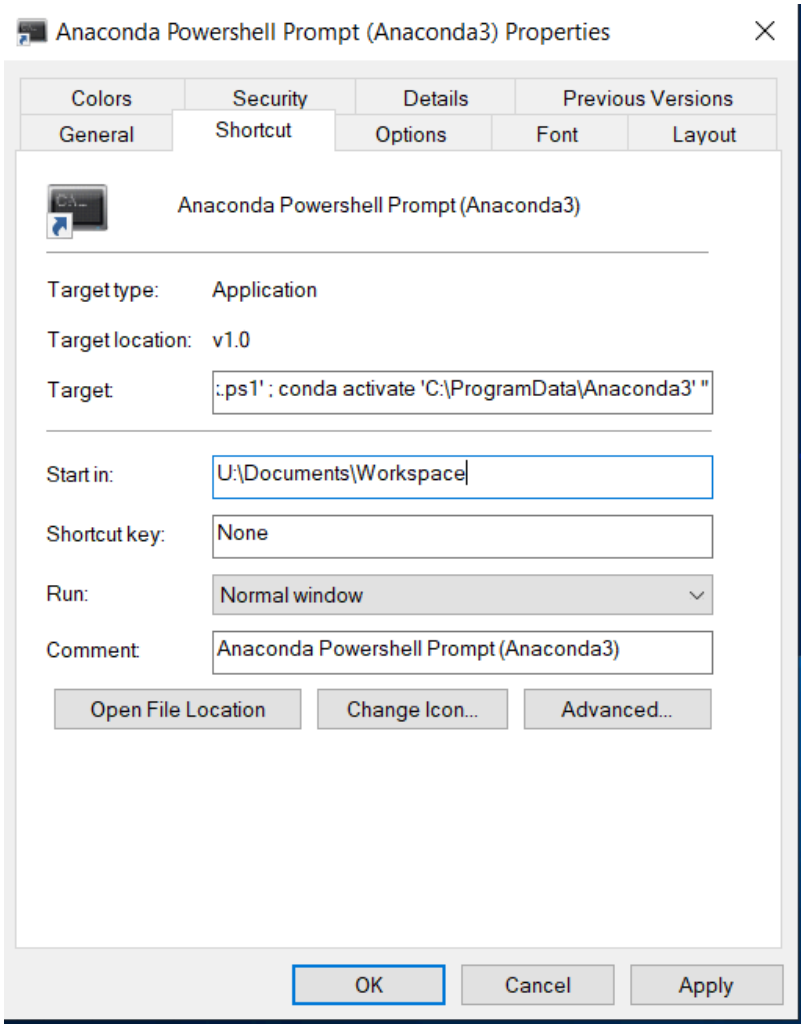
Search “anaconda prompt” from the start menu. right click on the app when it appears and pin it to the taskbar.



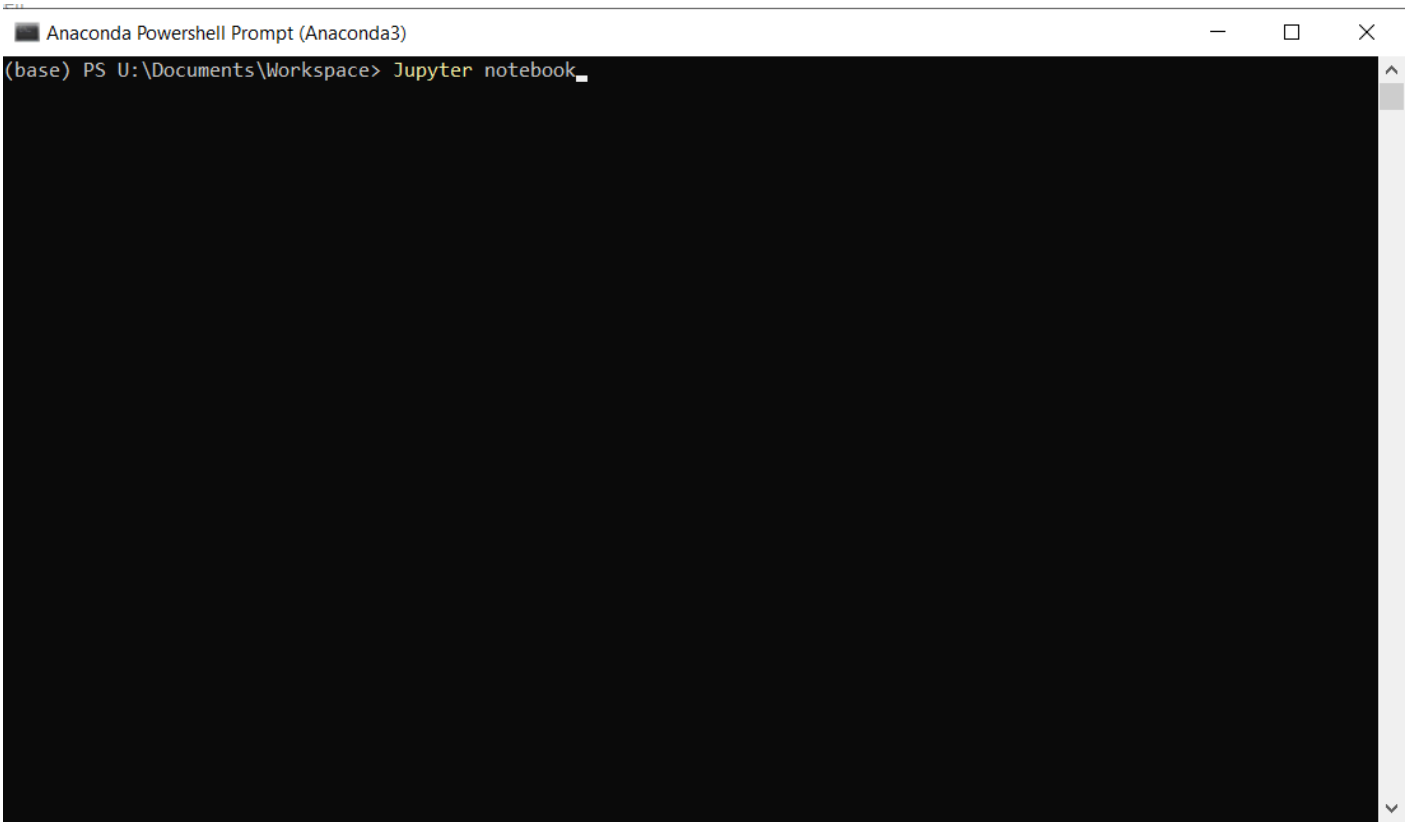
Right click on anaconda prompt in the taskbar (looks like a black window, similar to command line or terminal). Right click on “anaconda powershell prompt” in the tasks menu that pops up, and then properties.



In the properties window, go to the shortcut tab and change the “Start in:” field to U:\Documents\Workspace or whichever directory you keep your bitbucket repos in. Click apply.



Next, click on the anaconda prompt shortcut in the taskbar. When anaconda prompt opens, enter the command “`Jupyter notebook`”



Running Jupyter Notebooks

Interactively

Conda | **VS Code**

In order to run Jupyter notebooks that you started from the Conda prompt, do the following, once you have opened the Jupyter Notebook or Jupyter Lab interface:

- Navigate to the directory where the notebook is located
- Open the notebook
- Clear all the cells: `Cell` -> `All Output` -> `Clear`
- Run all the cells: `Cell` -> `Run All`
- Save the notebook: `File` -> `Save and Checkpoint`

From the command line

You should also be able to do the following from the command line:

Linux/macOS | **Windows**

If you have a LaTeX installation, you can convert the notebook to a PDF using the following commands:

```
# requires a latex installation
pip install nbconvert ipykernel
```

Define the notebook name:

```
NOTEBOOK=/path/to/notebook.ipynb
```

```
jupyter nbconvert --ClearOutputPreprocessor.enabled=True --to notebook  \
  --inplace --execute "$NOTEBOOK"
jupyter nbconvert --to pdf   "$NOTEBOOK"
```

Alternatively, you can convert the notebook to a PDF more closely resembling the HTML view as follows. First, install additional packages:

```
pip install nbconvert[webpdf] ipykernel
pip install pyppeteer
```

Then run this code:

```
jupyter nbconvert --ClearOutputPreprocessor.enabled=True \
  --allow-errors \
  --inplace  --to notebook --execute "$NOTEBOOK" || echo "Errors occurred while executing"
jupyter nbconvert --to webpdf --allow-chromium-download "$NOTEBOOK"
```

Document the packages YOU used

If you had to iteratively install packages, you should run the following command at the end of your whole process:

```
pip freeze > requirements.txt
```

and add that output (the contents of the `requirements.txt` file) to the repository, and to an appendix in the report. Take care to not overwrite author-provided `requirements.txt` files.

Julia-related procedures

In this section, we will show you a few things related specifically to running code reproducibly with Julia. For more general debugging tips for Julia and other computer languages, see our wiki.

The following instructions are primarily for Linux. Click to hide

You may need to use Conda on our Windows systems.

Managing multiple Julia versions

Julia has a native version manager. Follow instructions at JuliaLang/juliaup to install it - it will be installed in your personal directory (so is not shared). Then, to add a particular Julia version,

```
juliaup add 1.9.2
juliaup default 1.9.2
```

and your next call to Julia will launch 1.9.2.

Julia is picky about how environments are passed to workers

```
julia --project=julia16 -p 4
```

should start with the project installed in folder `julia16` and 4 worker processes. In fact, the worker processes ignore the project packages, and the whole thing fails (see this link).

Workaround:

```
export JULIA_PROJECT=julia16
julia -p 4
```

seems to work.

Importing someone else's Julia environment

An author using Julia may have included `Project.toml` and/or `Manifest.toml`, which specify Julia version and dependencies. To activate this environment (one time action):

```
julia --project=project/path -e 'using Pkg; Pkg.instantiate()'
```

where `project/path` contains `Project.toml` and `Manifest.toml` (typically, the code directory). (more information here).

Alternatively, you can add

```
using Pkg
Pkg.instantiate()
```

at the top of the main file (if there is one), and invoke this

```
export JULIA_PROJECT=project/path
julia -p 4 main.jl
```

In many cases, `project/path` will be `.`, i.e., `main.jl` and the TOML files are in the same directory.

Docker-related procedures

In this section, we will show you a few things related specifically to running code reproducibly with Docker. For more general debugging tips for Docker and other computer languages, see our wiki.

(coming soon)

Checklist

- [ ] If running licensed software (e.g. MATLAB, Stata), know where the license is stored. If you do not, you cannot run the relevant Docker image.
- [ ] Docker (or equivalent) is installed
- [ ] The replication package's main directory will be mounted into the Docker container. The precise location will depend on the Docker image. Know where it is.

Windows | **Mac/generic Linux** | **BioHPC**

> **Note**
>
> Docker does not run on CCSS Cloud! These instructions may work on a personal laptop.

See Docker on Windows.

Run the code

Docker on Windows

> **Warning**
>
> **NOTE:** *This document is a work in progress.*

> **Note**
>
> This will only work on a laptop or desktop. It does not work on (university-provided) cloud systems!

Install

- Follow these instructions to install Docker Desktop for Windows.
- Pay close attention to the system requirements:
  - WSL 2
  - Hyper-V
- For anyone using a University computer it may be necessary to run as administrator.

Testing Setup

- Open up a `Command Prompt` and type `docker --version`. If working, it should output a Docker version and build number.
- In the same shell, type `docker run hello-world` to test if your installation is working correctly. If working, you should get the following:

```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```

> **Note**
>
> In contrast to most other documentation in this manual, the examples below use the old Windows `Command Prompt`, not `bash` and not `PowerShell`.

Stata Imperfect Example

1. Start Docker Desktop.
2. Download example project from ICPSR.
3. Open up a `Command Prompt` (cmd) and cd into the workspace folder you would like to place the project contents.
4. In the command line: `Unzip C:/path/to/Downloads/118568.zip`.
5. Download data from here - Alaska only.
6. Unzip into “data” folder.
7. Test Docker: `docker --version` or `docker run hello-world`.
8. Navigate to dataeditor dockerhub and click on the Stata version you have on your workspace > Tags > Copy the docker pull command, i.e.: `docker pull dataeditors/stata16:2021-06-09`
9. Paste into command line.
10. Set version: `set VERSION="16"` and image: `set IMG="dataeditors/stata16:2021-06-09"`.
11. Test with: `echo %VERSION%`
12. Next, cd into the “programs” folder: `cd /programs`.
13. Update the globals in `config.do` to reflect that the “programs” directory is the current working directory.
14. Run the Docker image:

```
 docker run -it --rm ^
-v C:/path/to/stata_license/stata.lic.%VERSION%:/usr/local/stata%VERSION%/stata.lic ^
-v %cd%:/code ^
-v %cd%/../data:/data ^
%IMG%
```

15. This opens an interactive Stata session.
16. Execute the master file: `do master.do`
17. Hit error:

```
save `dtahu', replace  /* save housing unit data */
(note: file /programs/../data/outputdata/housing.dta not found)
file /programs/../data/outputdata/housing.dta could not be opened
r(603);
```

18. Looks like there's no folder "data/outputdata/". A global was already defined for this so we can add `cap mkdir $outputdata` to the `config.do` file to create that folder. Re-run `master.do`.
19. Another error: `command latab is unrecognized`. Add the "latab" package to `00_setup_stata.do` and re-run.
20. No "tables" folder in the root directory to send the output tables to. However, this folder wasn't mapped to the Docker image so we'll have to create the directory and map it in our docker run command.
21. End the Stata session: `exit, STATA clear`.
22. Make the tables directory from the command line: `mkdir tables`.
23. Adjust the docker run command to account for the results folder:

```
 docker run -it --rm ^
-v C:/path/to/stata_license/stata.lic.%VERSION%:/usr/local/stata%VERSION%/stata.lic ^
-v %cd%:/code ^
-v %cd%/../data:/data ^
-v %cd%/../tables:/tables ^
-w /code ^
%IMG%
```

24. Again, and execute `master.do`.
25. Done!

It is also possible to execute the program outside of an interactive Stata session:

1. Use the following docker command:

```
 docker run -it --rm ^
-v C:/path/to/stata_license/stata.lic.%VERSION%:/usr/local/stata%VERSION%/stata.lic ^
-v %cd%:/code ^
-v %cd%/../data:/data ^
-v %cd%/../tables:/tables ^
-w /code ^
--entrypoint /bin/bash ^
%IMG%
```

2. Issue the following shell command:

```
stata -b do master.do
```

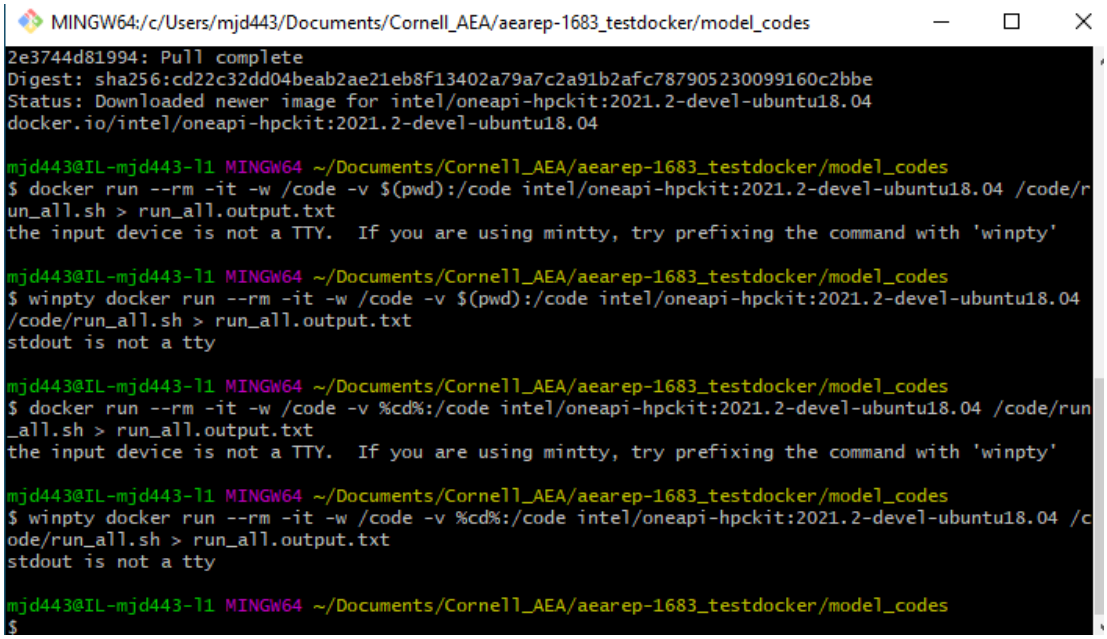
to execute `master.do`.

Fortran Example

1. Downloaded the Intel OneAPI-hpckit by entering the following into the command line:

```
docker pull intel/oneapi-hpckit:2021.2-devel-ubuntu18.04
```

2. Navigate to the directory with all the Fortran model codes: `cd PATH/to/model_codes`
3. Hit multiple issues attempting to run in Bash shell:



4. On Windows 10, opened a `Command Prompt` (cmd) instead.
5. Typed the following into the command line to run the shell script which executes all Fortran code:

```
docker run --rm -it -w /code -v %cd%:/code intel/oneapi-hpckit:2021.2-devel-ubuntu18.04 /code/run_all.sh
```

- This maps the current directory (model_codes) to the docker image as "/code" and runs `run_all.sh`, creating the log file `run_all.output.txt`.

Wait, what is docker and what did we just do?

Docker is a platform of services for the use of 'containers'. These containers allow the sharing of programs or apps in isolation of their enviroment. In the imperfect example above, we 'mounted' our local folders to the container and then run code inside it using the Stata 16. We did the same thing in the Fortran example, except then we used a Fortran container. It may be helpful to know the difference between a docker image and a docker container.

It may also be helpful to understand the basics of the commands we just run:

1. `docker pull dataeditors/stata16:2021-06-09`: This command is pulling the image "dataeditors/stata16" with the tag "2021-06-09" so we can used it in our machine.
2.

```
docker run -it --rm ^
  -v C:/Users/mjd443/Documents/aea-licenses/stata.lic.%VERSION%:/usr/local/stata%VERSION%/stata.lic ^
  -v %cd%/programs:/programs ^
  -v %cd%/data:/data ^
  -v %cd%/tables:/tables ^
  %IMG%
```

We used `docker run` to create a container based on a specified image (dataeditors/stata16) and then start it. Lets undestand some of the options we used:

- `-i` is short for `--interactive`, `-t` is short for `--tty` and is used for allocating a pseudo-TTY (whatever that means). They are often used together, as we did: `-it`. They allow us to work interactively with Stata inside the container.
- `--rm` is used to automatically remove the container when it exits.
- `-v` is used to bind mount a volume. In other words, we are mapping a folder in our machine to where we want that folder to be in the container (`-v our/folder:container/folder`). So `%cd%/programs` is our current.directory/programs folder, and we "mount it" in the container, creating a "programs" folder inside it. We did the same for the stata license, as well as for "data", and "tables" folders.

Note that at the end of the line we have `%IMG%`, which is a local variable we previously had set to be equal to the image we are using: `dataeditors/stata16:2021-06-09"`. All of that allowed us to have an interactive Stata inside the container with access to the data and .do files that we needed to replicate the project.

3. In the Fortran example when we used:

```
docker run --rm -it -w /code -v %cd%:/code intel/oneapi-hpckit:2021.2-devel-ubuntu18.04 /code/run_all.sh
```

where the option `-w` sets the working directory inside the containers to a the existing folder called "code". Then, we "mounted" the current working directory in the host machine to the "code" folder inside the container. We use the image `intel/oneapi-hpckit` with the tag `2021.2-devel-ubuntu18.04` to create and start a container that run the the bash script as described previously.

Mathematica-related procedures

In this section, we will show you a few things related specifically to running code reproducibly with Wolfram Mathematica. For more general debugging tips for this and other computer languages, see our wiki.

Files

Mathematica uses the file extension `.nb`. This includes both code and the output. You will run the code, and then save the output in two ways: `.nb` and as a PDF.

> **Note**
>
> At this time, Mathematica is only available on CCSS machines.

Instructions

Follow these instructions for each file:

- Open each `.nb` file, and do the following:
  - `Cell` -> `Delete all output`
  - `Evaluation` -> `Evaluate Notebook`
  - `Save`
  - `Save as...` -> `PDF`
- Then **completely** exit Mathematica again!

> **Note**
>
> You can paste this into the `Replication Steps` section when appropriate:
>
> ```
> - Opened each `.nb` file, and did the following:
>   - `Cell` -> `Delete all output`
>   - `Evaluation` -> `Evaluate Notebook`
>   - `Save`
>   - `Save as...` -> `PDF`
> ```

Then commit both the re-executed notebooks and the new PDF files to your repository.

Variations

There are a lot of different ways to run Mathematica code. Should this not work, talk to your supervisor.

Rare software

In this section, we will show you a few things related to various other software that don't occur very frequently. Please feel free to add information to this section as you experience the steps needed to do so.

Ox-related procedures

In this section, we will show you a few things related specifically to running code reproducibly with Ox. For more general debugging tips for this and other computer languages, see our wiki.

Files

Ox uses the file extension `.ox`.

Instructions

Ox is not installed on any of our machines. The best way to run it is to use Docker on BioHPC.

- Run the Docker image you will have built.
- Inside the running container:
  - Run `oxl filename | tee filename.log` to save the log files (modify `filename` as needed.)
  - If need to modify files as instructed by the authors, exit the Docker container and type `gedit filename`, or do so from your VSCode window.

Mosek

Describe the steps needed to request a license, install the software, and test that it works.

External reproducibility checks

In some cases, reproducibility checks must be conducted externally—particularly when restricted-access data is involved. This section outlines how to integrate such reports and ensure they are properly archived in our systems.

When External Reproducibility Reports Are Used

Instances that require the use of restricted-access data for a full reproducibility check can occasionally be sent out for an external report. These cases often arise when data cannot be shared with our team.

For more information, see the AEA Policy and Protocol on Third-Party Verifications, which describes who can conduct these external reports and the expected process.

> 🔒 In all cases where an external reproducibility check is used (e.g., CASCAD or World Bank), the **“external validation” checkbox in JIRA must be checked**.

Integrating the External Report

The external report must be included in the repository. It is **not sufficient** to leave it attached to a JIRA ticket or email.

1. Start with the internal `REPLICATION.md` report.
2. **Do not delete** any section titles from the internal report.
3. In relevant sections, add a note such as:

   > “Please refer to the external report below.”

4. Review the external report and ensure all **bugs** and **discrepancies** are recorded in the **Summary and Findings** section of the internal report.

Writing an External Report

If you have agreed to provide us with an external reproducibility check, we kindly ask that you follow these instructions.

The manuscript, code, and data are confidential at this stage

Do not post them to any public place. Use the resources we provide you with, or those that your institution provides you with. While we may provide you with Github or Bitbucket repositories, those are to remain private, i.e., require a login for access.

Once you have completed the task, and sent back all changes you had to make, or sent us the report from a secure system, delete all data and code when told to do so (don't do this immediately, as we may have clarifying questions).

Use a separate environment to run the code where possible

While you *can* use your laptop, you should follow certain procedures to both not affect the reproducibility check nor affect your usual working environment. Notes below.

Access to code and data

You may have been provided access to code in two different ways

openICPSR link | **Bitbucket link**

You may have been provided with a link to openICPSR. If so, you should download the code and data from there.

Follow the procedures in the authors' README

You should follow instructions as closely as possible. However,

- you are not required to do extremely tedious or time-intensive manual steps.
- you are not required to "bog down" your personal laptop (see resources)
- you should lightly improve on the procedures, by following our replication procedures (below)
- you are not required to install software or packages that might mess up your own research

Reproduction procedures

In addition to any setup instructions from authors, a few things to take into account

- Use the resources we provide you with, where possible
- Use the procedures to use "config.do" or similar files.
  - This is always possible for Stata. See instructions.
  - Similar procedures are available, to some extent, for R (use separate libraries, if possible, or the `renv` or similar packages). See instructions.
  - Similar procedures are possible for Python and Julia (use `environments`)
  - Matlab and SAS will always use whatever is installed system-wide - this is a known caveat.

Log everything

Create a log file, if possible, for every run.

- For Stata, our template config.do file will handle this, if used correctly. See instructions.
- For R, use "R CMD BATCH" to run code, even when using Rstudio (use the Terminal tab). See instructions.
- For Matlab, where possible, use the command line method of launching it. See instructions. Alternatively, use "`sink`", but note that it might interfere with some programs.
- For Julia and Python, we have no good solutions other than to use the command line where possible, and capture the output.

Once you have a log file, commit it.

Document everything

Keep a journal of what you are doing. You should be able to point to the journal, together with a log file or screenshots, to document problems, and how you solve them.

An example:

> - Downloaded code and data from openICPSR
> - Added line to use `config.do`
> - Ran `main.do` as instructed by the author
>   - I used the "right-click" method on Windows
> - Code stopped at the third step, looking for package `xyz`
> - Added the package to the relevant section in `config.do` so it would get installed, and ran the entire `main.do` again
> - Programs finished but no figures were output. Inspection of the code showed that they only display on-screen. Added `graph export` as PNG files at all relevant parts, then ran entire `main.do` again.

If the authors' instructions say to "view" something interactively, investigate native methods (`graph export`) to capture the information. Otherwise, use screenshot to capture the information. Make a note of that, too.

Commit all logs, outputs, etc. to Git

All logs and outputs, but not data, should be committed to Git.

Compile a report

Our standard report template is included: external-REPORT.md.

- Don't forget to report your computer and software configuration
- The report should be committed to git as well. There is no need to convert it to PDF.

Send a report

Reports should be committed to git (**not** sent by email). An email notification that all is complete is sufficient.

> Please “reply-all” to the email you received.

Final confirmation

We will confirm to you when we have exhausted all our questions, at which point we will ask you to delete all code and data.

Wrap-up: Delete all code and data

Once you have completed the task, and committed all changes to the Git repo we provide you with, or sent us the report from a secure system, AND have answered all of our clarifying quesitons, please delete all data and code.

Resources

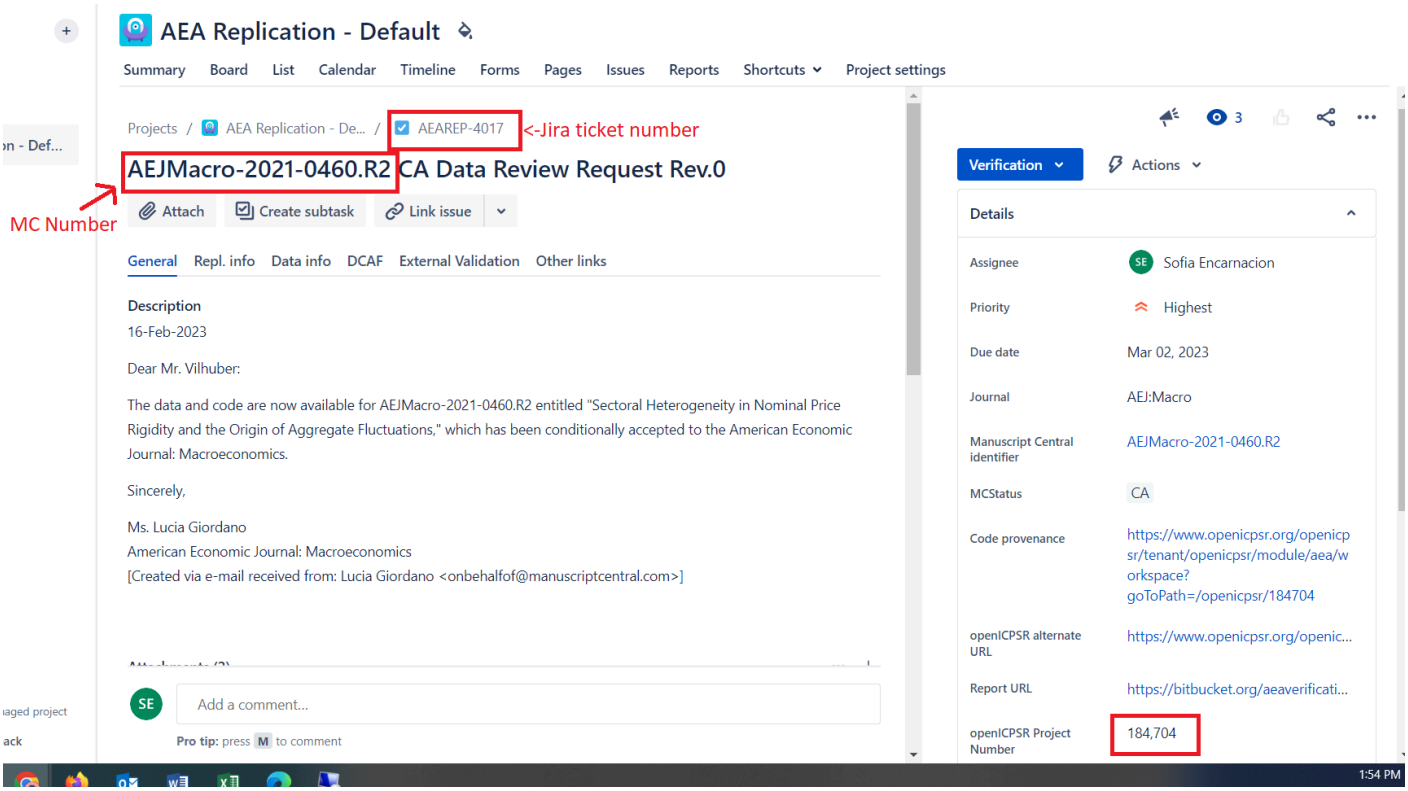
You can have access to all the resources we regularly use:

General | **Cornell-affiliated**

- BioHPC compute cluster. If you do not have an account, request one. Even if you have an account, request to be added to the “lv39” lab.
- CodeOcean - Don’t forget to share with “dataeditor@aeapubs.org”. Instructions are here
- WholeTale
- Github Codespaces - with some caveats, talk to us
- Your laptop

Glossary

Example of Jira ticket:



Terms

- **Jira ticket number** - The main way to refer to your cases. If you need help, this number is necessary for us to know
- **MC Number** - (or Manuscript Central Number) This is how we identify the manuscript. Only necessary to know for the replication report header
- **open ICPSR** - The repository that holds the data and code that the author has submitted. This will eventually be published.
- **bitbucket** - The repository that holds the replication report, code-check.xlsx, and any other items necessary to write the report. You will add the code from open ICPSR via Pipeline or manually. This repository is private. Authors will only see the replication report.
- **Jira workflow** - the workflow is how we identify at a quick glance what progress you've made in the case
- **manuscript** - the article that the author plans on getting published. It contains the tables and figures you'll be checking. The manuscript is attached to the Jira ticket with the title "PDF_Proof"
- **environments** - where you will be running the code
  - **CISER/CCSS**
    - Cloud
    - Classic
  - BioHPC
- **The Wiki**- The wiki has computational guidance, suggestions, tips and tricks.

- **RR**
- **CA**
- **Revision** - A Revision is the second time we see a case. A bitbucket repository already exists for this case.

Setup Checklist

Accounts you will need to sign up for (action required)

- [x] Cloud computing account (for computing access).
- [ ] You will need openICPSR account, in order to download pre-publication materials. Please be sure to use your Cornell e-mail (or if not affiliated with Cornell, use your institutional email).

Accounts you will be signed up for (no action required)

- [x] Atlassian account
  - Bitbucket for access to the internal Git repos
  - Jira account for the internal issue tracker
- [x] Email is used for the mailing list ldi-lab-l@cornell.edu

Software to install on your laptop

- [ ] Remote access app for CCSS Cloud Computing. Follow the current CCSS instructions for your operating system. Windows users should install the official “Windows App” from the Microsoft Store rather than the older Microsoft Remote Desktop app.
- [ ] Git command line tool (download the software, then follow the guide on Installing Git on your computer
- [x] Command line (We will use the “Bash shell”, but PowerShell and Zsh will work, too)
  - Windows: “Git bash” comes with the Git install above. You could also use WSL2 (OPTIONAL, Pro)
  - macOS: Use the “Terminal” app (pre-installed)
  - Linux: Use any “terminal” app (pre-installed)
- [ ] Visual Studio Code (download location), a powerful text editor
  - [ ] Set up settings sync (OPTIONAL, recommended)
  - [ ] Install Plugins: Excel to Markdown table
  - [ ] Install Plugins: Markdown Preview Enhanced
  - [ ] Install Plugins: Auto-Open Markdown Preview (OPTIONAL, can sometimes be inconvenient)
  - [ ] Install Plugins: markdownlint will debug your Markdown

Other software is optional. You will use statistical software on other computers that we will get you access to.

Text editor vs. Word processor

You want to use a **text editor**, not a **word processor**. The difference: a **text editor** creates simple text files, without fancy formatting.

If you are creating code in Stata, Matlab, or Rstudio, you are using a customized text editor, sometimes called an **IDE** = "Integrated Development Environment". That is OK. But remember that all such program code is a straight text file.

General purpose text editors can view and edit them all, although they may lack some of the fancy features that make programming easier in the dedicated IDE.

Availability and Suggestions

	OS	Laptop	CCSS-RS	Custom node
MS Visual Studio Code	All	Suggested		Yes
Notepad++	Windows		Yes	Yes
vi	Mac, Linux			

Note

- It is possible to synchronize VSCode settings and modules between your laptop, the cloud, and CCSS-RS.

Help

- Git cheatsheet and another one
- Markdown cheatsheet

Privacy

We need to cover two sorts of privacy: the privacy of those whose materials we verify, and your own privacy. There are limitations to both, but we attempt to protect privacy as much as possible.

Privacy of Replicators

You are tasked with reproducing articles. Much as referees for journals mostly remain anonymous, we want you to remain anonymous as well.

- You *may* reveal yourself to authors later (after the task is completed), if you wish.
- You should *not* contact authors unless authorized by the Lab Leader. Normally, all such communications go through the Lab Leader.
- We do *name* you (to thank you) in the annual report, but do not attribute your work to any one article.

- In the empirical analysis of all the articles, we replace your netid and name with an anonymous (and untraceable) identifier. So we can track that you have done Articles A1, D57, and Z31, but nobody knows that it was **you**.

There is “leakage” of information:

- In order to download materials, you need to login to openICPSR, and have the ability to download from specific deposits. This *does* reveal your name to the depositors. This is currently a technological constraint, and cannot be avoided without great complications.
    - If you have concerns, please let us know, and we will find a workaround.

Should you ever be contacted in some unacceptable fashion by authors, you should immediately contact the Lab Leadership.

You can, and you **should**, reveal your affiliation with this project! You can (and you should) be proud of the work you will do or have done, and you are allowed (and you should) reference this project as an accomplishment.

The Privacy of Authors

When we do pre-publication verification, this is equally important.

- You are never allowed to reveal that the author has submitted to the journal
    - This includes when you need to contact third parties for materials that are part of the replication materials. In case of doubt, contact Lab Leadership.
- You are never allowed to reveal anything about the analysis that the author is conducting, and that you are reproducing, to anybody outside of this group.
- You must never put the code, the article, or the data on a location where others outside of this group could access it
    - Bitbucket within the `aeaverification` project is OK, do not attempt to make a repository public (even if it may seem convenient not to have to enter your login etc.)
    - Remove the files from your laptop as soon as you are done with it (after `git push`, of course)
    - You may remove them from CISER nodes, but those will be cleansed later
    - Do not email or otherwise disseminate (twitter, facebook, snapchat, whatever) the files received, or any other information about the papers

Communication

Medium

Email

We use email (yes, we know 🙄) through a mailing list

- Ask lots of questions
- Communicate with each other (answer each other’s questions)

When you expect to be absent/ cannot handle a request in a timely fashion

- email the AEA Data Editor and assistants

Jira

For pre-publication tasks, we also use Jira.

- you can comment on Jira in the web interface
- in the Jira app on your phone
- by response email

When you are assigned a task in Jira, and do not expect to be able to start work on it immediately, please let us know ASAP.

Github

We have a wiki with guidance, suggestions, tips and tricks.

- look there first for answers
- if you get an answer by email, or figure it out yourself, add it to the FAQ

Language

Don't write anything you wouldn't want your British aunt (or your mom, or your dad) to see. Write cleanly and concisely.

- much of what we write will be seen by others
- much of what we write is public (Wiki)
- the reports you write should be concise, and shareable (we expect to do minimal editing before sending it out to authors)

For reference: Authors who have seen the documents written by this group

- Esther Duflo (Nobel Prize Economics 2019)
- John Abowd (Chief Scientist at the US Census Bureau)
- Janet Yellen (former Chair of the Federal Reserve Board)
- Thomas Piketty (famous economist)

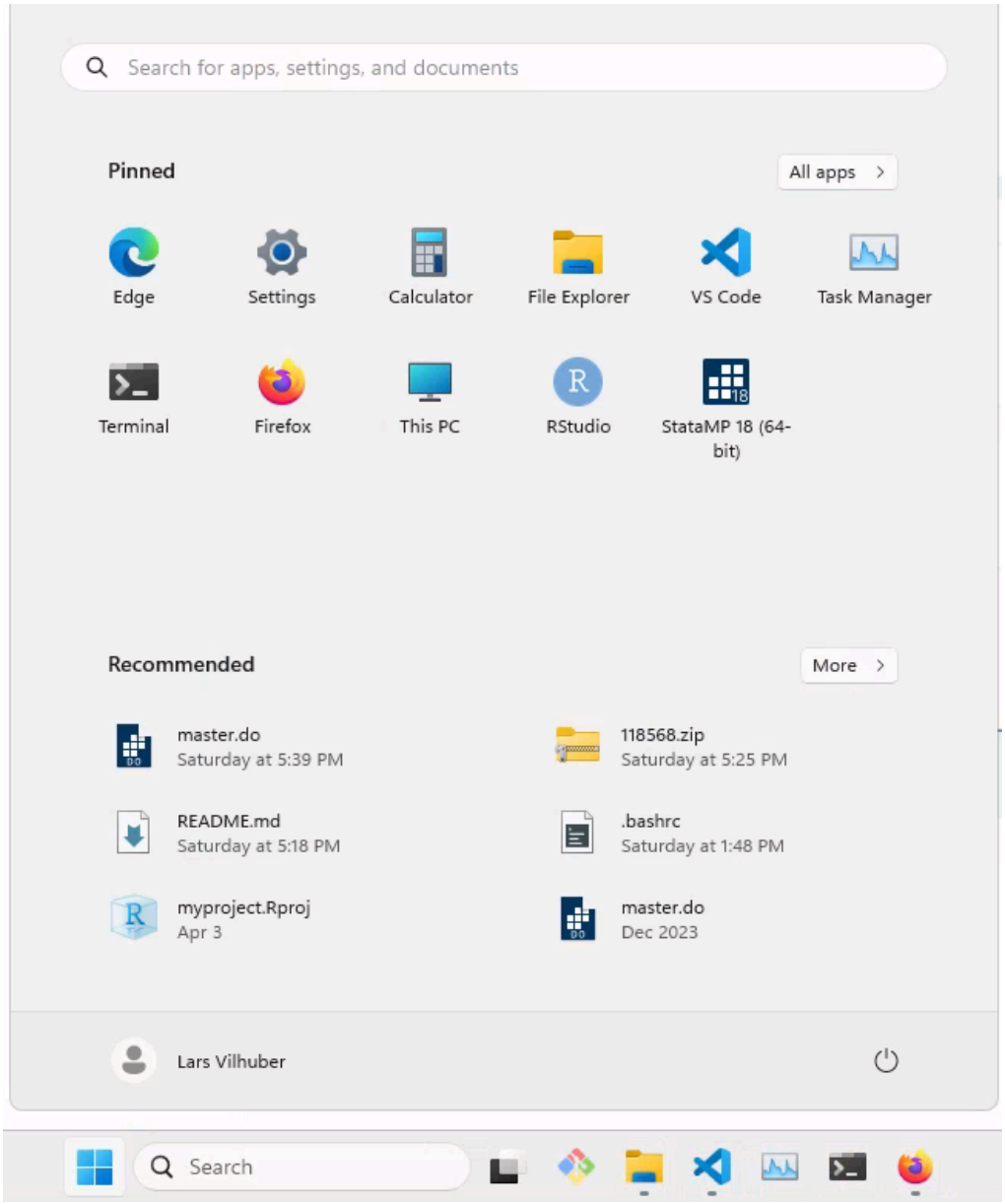
Identifying differences between original manuscript and reproduction

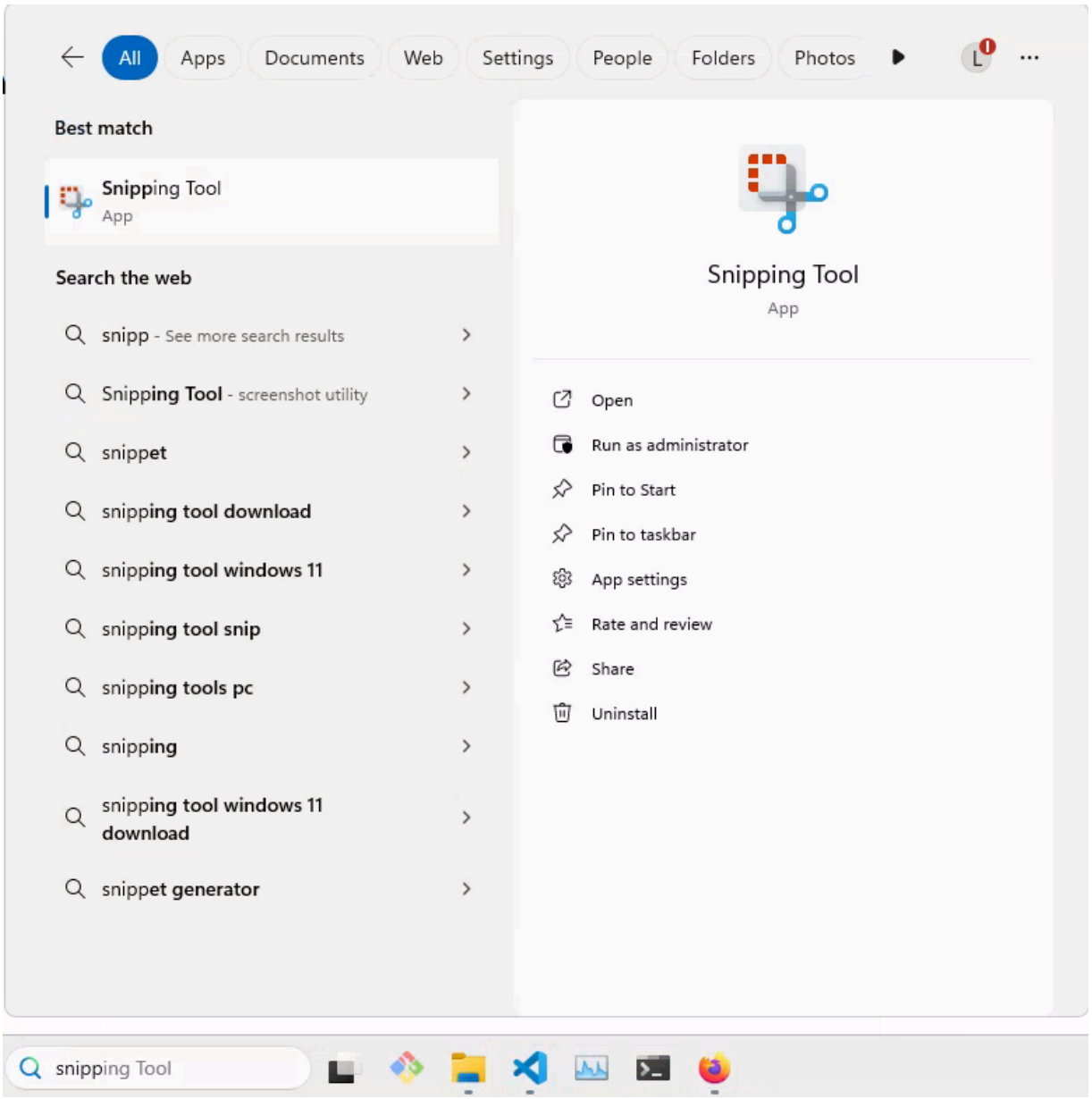
In the following sections, we will provide a few tips on how to identify differences between the original manuscript and the reproduction. In practice, you will want to use either graphical or textual annotations. The tools that you will use will vary depending on where you do the annotations - a Windows computer (Cloud or laptop), or a macOS laptop.

Using screenshot and annotation tools on Windows

Search for the “ Snipping Tool”

The “Snipping Tool” is a built-in Windows tool that allows you to take screenshots of your screen. You can search for it by typing “Snipping Tool” in the search bar.





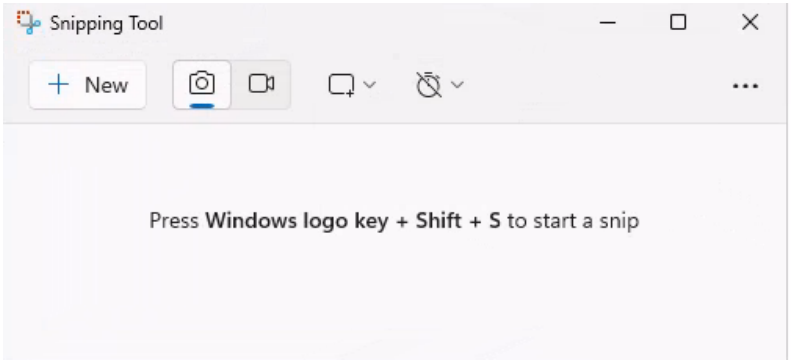
You can also pin it to the Start menu or the task bar, for easy and quick access.

> **Tip**
>
> You can also use the shortcut `Windows + Shift + S` to open the tool.

Configure the “Snipping Tool”

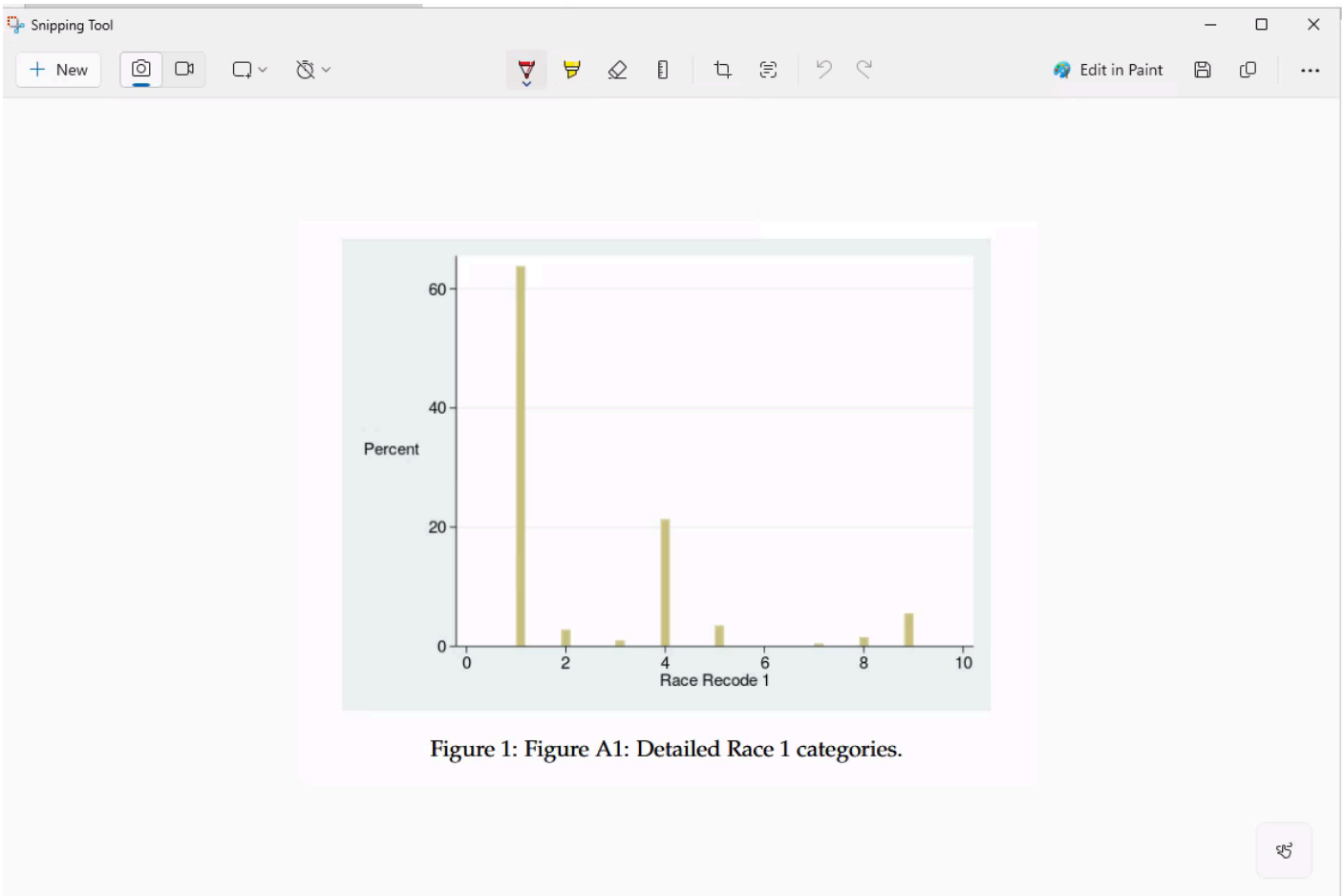
- You will want to record a picture, not a movie.
- You will want to select a rectangle, not the entire screen or a window.

It should look something like this:



Take a screenshot

- Open the Snipping Tool.
- Click on “New”.
- Select the area you want to capture.



Save the screenshot

For the purposes of the reproducibility check, you should call the file something that represents the table or figure you are capturing.

- If you are capturing a table from page 3, you could call the file `table3.png` and save it in a **new** directory called `paper/`.

- If you have only a few screenshots, save it in the project root as `table3-paper.png`.
- Avoid spaces in the file name.
- Do not use the default name `Screenshot YYYY-MM-DD HHMMSS.png` as it is not descriptive, and contains spaces.

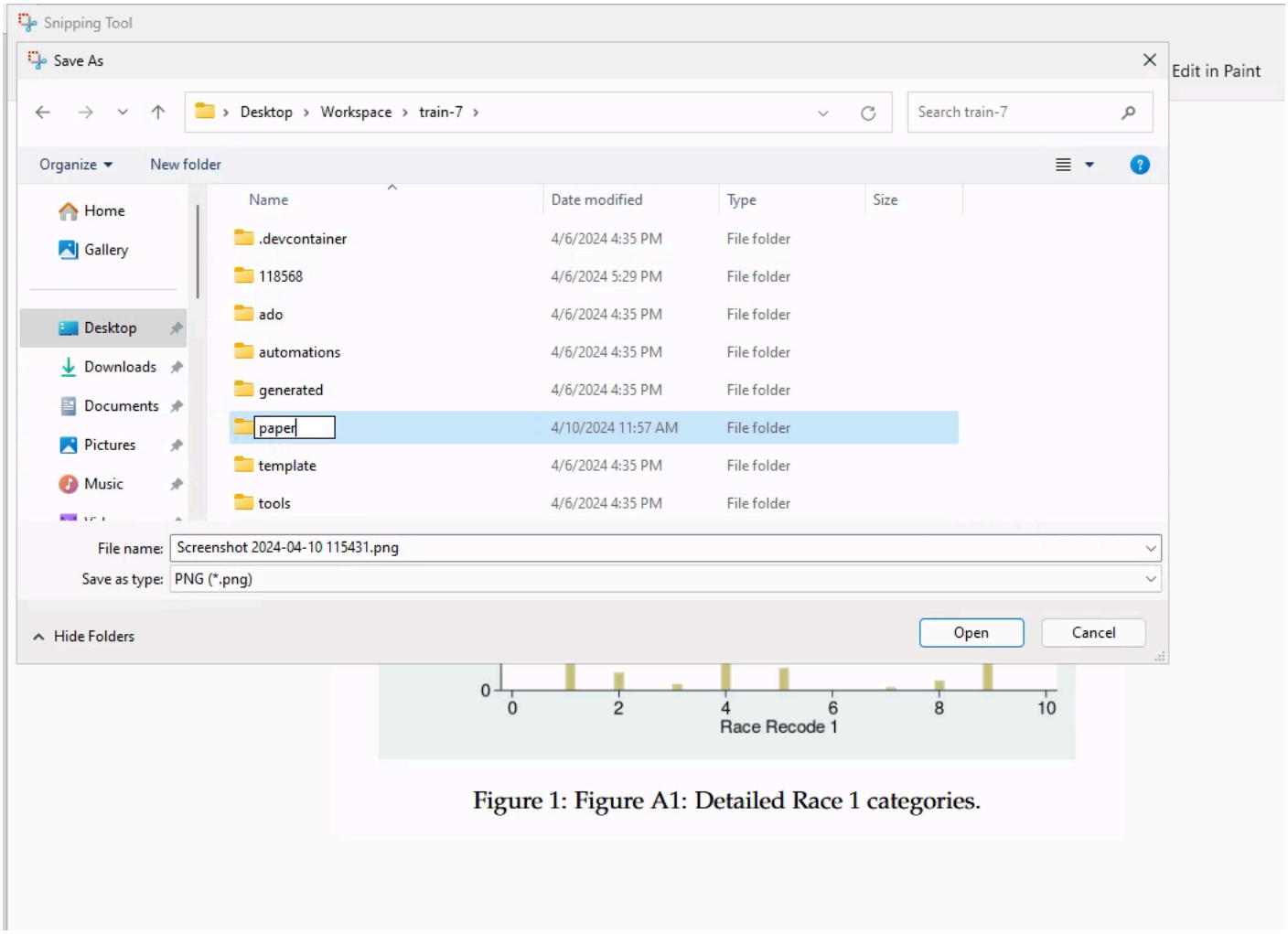


Figure 1: Figure A1: Detailed Race 1 categories.

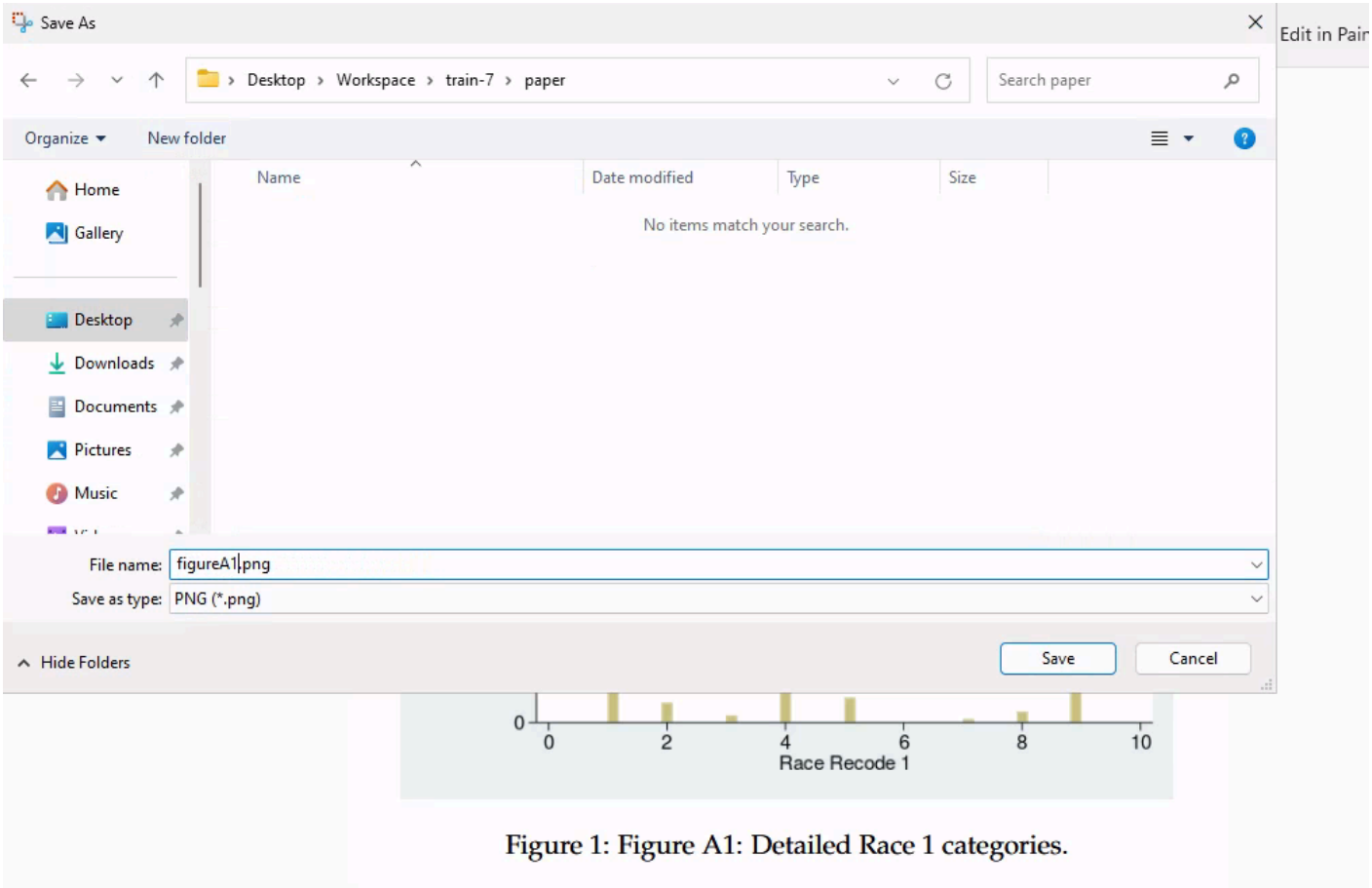


Figure 1: Figure A1: Detailed Race 1 categories.

Annotate the screenshot

You can use the “Snipping Tool” to annotate the screenshot. You can do this before saving the screenshot, or afterwards. If the latter, simply save over the original screenshot.

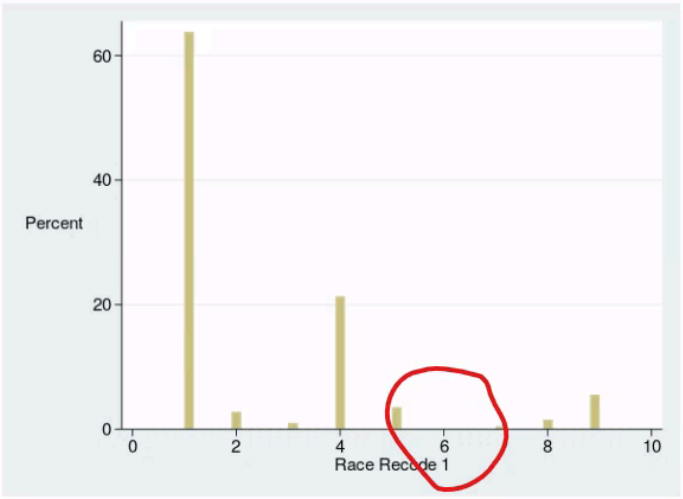


Figure 1: Figure A1: Detailed Race 1 categories.

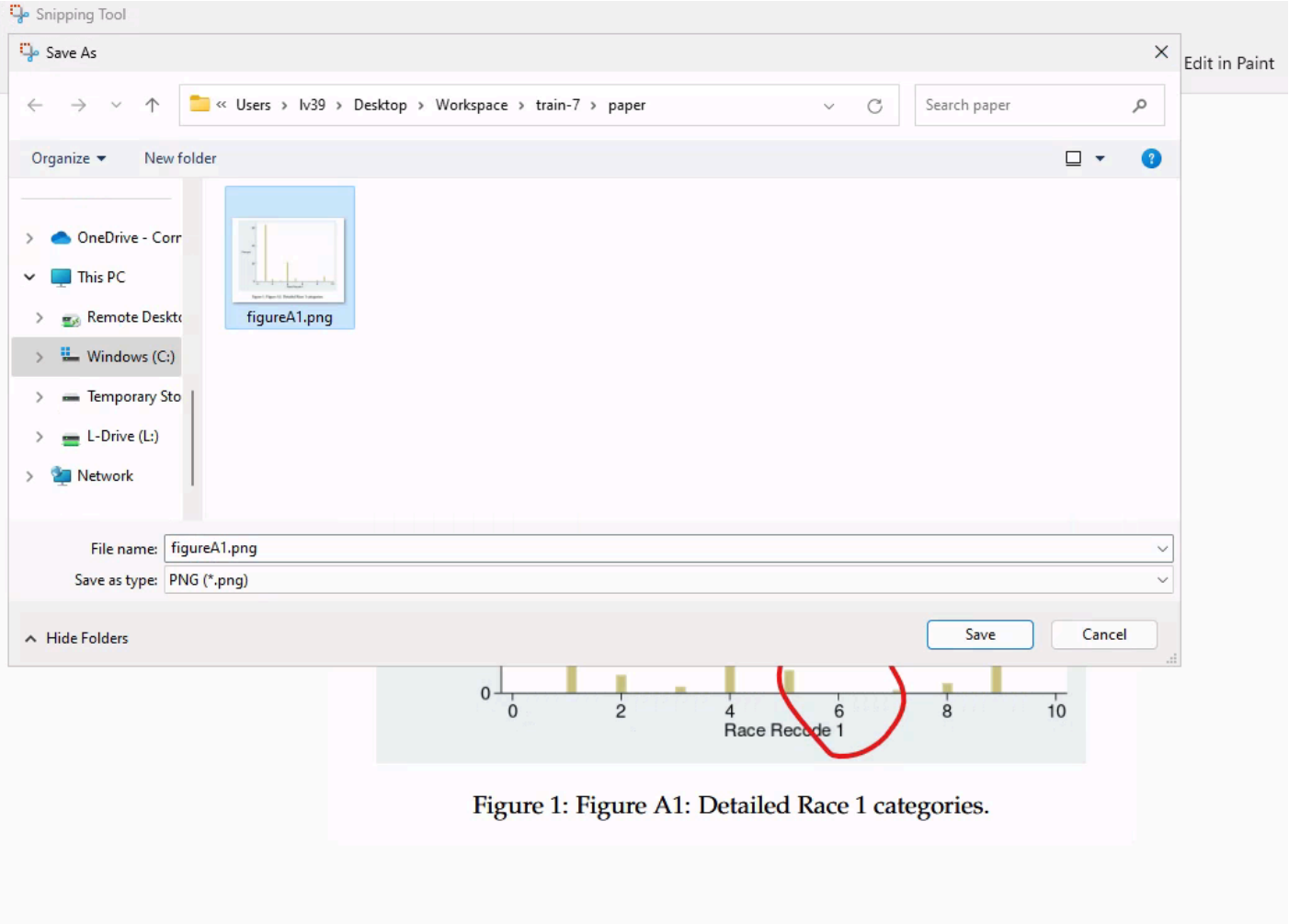


Figure 1: Figure A1: Detailed Race 1 categories.

> **Tip**
>
> Other tools may also exist - choose your favorite tool!

You can now include the screenshots, and the original picture, in the report.

Using screenshot and annotation tools on macOS

Coming.

Include Screenshots in the Report

Once you have collected all screenshots, and annotated any discrepancies, you will include them in the report.

Describe the differences in words

Describe what you see. In the “Findings” section, there are separate Table and Figures sections. List out, as a bulleted lists, the differences that you see:

```
- Table 1: The reproduced standard errors are all larger in the 6th column.
- Figure A1: There is a bar missing at the value "6"  in the histogram.
```

Copy from the `code-check.xlsx`

After the list of all the changes, you should also copy from the `code-check.xlsx` (use the VSCode Excel to Markdown table extension).

Include the screenshots

Use the Markdown syntax to include the screenshots.

- for reproduced figures, use the path to the figure as produced
- the figure must be in PNG or JPEG format. Markdown cannot include PDF or Word files. If you need to convert, contact your supervisor, or check to see if there are pipelines available to convert files.

You can use the following syntax:

```
Figure A1 as reproduced:

![Figure A1 as reproduced](123456/figures/figureA1.png)

Figure A1 from the original manuscript:

![Figure A1 from the original manuscript](paper/figureA1.png)
```

Downloading Data

Most data we receive comes from pre-publication openICPSR deposits. However, they may sometimes be privately provided and are then stored on a secure shared drive, or they may come from other trusted repositories (e.g., Zenodo, Dataverse), as well as restricted data.

Sometimes, when we process data on BioHPC, it may be more convenient to download the data to CCSS Cloud, and then transfer to BioHPC. See Globus transfers (this chapter) for more information. The use of `sftp` is another option.

Using pre-publication openICPSR Projects

Typically the AEA Data Editor team will access code and data provided by authors that is stored on openICPSR.

> Cornell replicators: You will need to set up an openICPSR account using your Cornell email.

Note

Not all deposits are provided to us on openICPSR. For instructions on how to handle other institutions (Zenodo, Dataverse, etc.), see these notes.

Basics of openICPSR

- Authors will create a draft deposit that contains the replication archive for their paper.
- Each deposit is identified with a six digit number (`123456`)
- You will download the project and commit the code (but not the data) files to the corresponding Bitbucket repo.

Downloading from openICPSR

Downloading using a script | **Downloading a project manually**

The most convenient way to download the information is to use the LDI-specific script `tools/download_openicpsr-private.py` (be sure to have set up your environment first). There are also scripts for other repositories that may work (type `ls tools/download*` to get a list of the most current ones).

Short version of call to script

While in your repository directory, open a (Git) Bash shell, and type

```
python3 tools/download_openicpsr-private.py 123456
```

which will download a file `123456.zip`. If this has not been done before, it will unzip it into a directory `123456`.

You may need to adjust `python3` to be `python`, depending on your environment.

Full options for script

```
python3 tools/download_openicpsr-private.py 123456 . mylogin@cornell.edu
```

where

- `.` indicates the directory to download the ZIP file to (here, the current directory)
- `mylogin@cornell.edu` is the login you use on the openICPSR website (typically, `netid@cornell.edu`)

See appendix on how to set up the environment to make the script simpler (i.e., allow the short form to run).

Expected output

You should see the following output:

```
openICPSR downloader v2024-02-01
No debug:None
Getting session cookies...
Initiating OAuth flow...
Logging in...
Accessing files...
Getting file info...
Downloading file: 123456.zip
```

If you see anything other than `123456.zip` (replace with your openICPSR project number), then consult the debugging steps below.

When it fails

(provide some guidance when it fails)

Reminders

Normally, none of the actions below are technically possible, but you should nevertheless follow these guidelines:

- Do not publish the openICPSR projects.
- Do not upload files to the projects.
- Do not share projects with others unless instructed otherwise.
- Do not share screenshots with others

See Privacy section about expectations on privacy.

Alternate sources of data

We sometimes encounter replication packages that reference Dataverse or Zenodo. Download those as they are found on those sites. Extract them as you would the openICPSR downloads, but please pay attention to the directory naming structure:

Source	DOI or project	Directory name (`DIRNAME`)	Pipeline Variable
openICPSR	https://doi.org/10.3886/E**123456**V1	`123456`	`123456`
openICPSR	openicpsr-**123456**	`123456`	`123456`
Dataverse	https://doi.org/10.7910/DVN/RE5ZVI	`dv-10.7910-DVN-RE5ZVI`	
Dataverse	https://doi.org/10.3456/ABCDE	`dv-10.3456-ABCDE`	
Zenodo	https://doi.org/10.5281/zenodo.7041706	`zenodo-7041706`	`7041706`
World Bank	https://doi.org/10.60572/101y-vn15	`wb-101y-vn15`	`101y-vn15` or entire DOI

If a `Pipeline Variable` is given, use that in the Bitbucket Pipeline when creating or updating the repository.

Usage

Whether manually downloading the files or using the scripts, once the ZIP file is downloaded, you would still use

```
    unzip -n NAME_OF_ZIP.zip -d DIRNAME
```

where `DIRNAME` is defined as in the table above, and `NAME_OF_ZIP.zip` is whatever you downloaded from the repository. In the case of the [openICPSR download script]((get-the-data), the file downloaded from openICPSR is typically called `DIRNAME.zip`, e.g, `123456.zip`. It will be called something else in most other cases.

Utility scripts

We have a few scripts, some of which have not yet been integrated into the pipeline. All can be run from the command line instead of manually downloading the repository. All are in the `tools` directory of your issue-specific repository. If not, refresh the tools.

- **download_openicpsr-private.py**: For downloading from private openICPSR deposits (also integrated into the pipeline)
- **download_dv.py**: For downloading from Dataverse repositories.
- **download_osf.sh**: For downloading from OSF repositories.
- **download_worldbank.py**: For downloading from World Bank Reproducibility Repository, including the public reproducibility report!
- **download_zenodo_public.sh**: For downloading from public Zenodo repositories
- **download_zenodo_draft.py**: For downloading from private/draft Zenodo deposits (also integrated into the pipeline)

Accessing privately provided data

When there is privately provided data

In some cases, authors will provide us privately with data we cannot publish.

> **Warning**
>
> VERY IMPORTANT: You must treat these data as confidential, never remove them from CCSS!

You will know where the data are by looking at the JIRA issue. You will often (but not always) see a **subtask** which we use to request the confidential data:

You should then look into the "`Data`" tab for the field `Working location of restricted data`:

How to prepare privately provided data

CCSS Cloud **CCSS Classic**

- Log into remote desktop as usual.
- The restricted access data will be stored in the L drive. If you haven't mapped the L drive, do that first (instructions here).
- Navigate to the folder `Restricted-Access-Data` and find the corresponding folder that is in the Jira field, e.g., `L:\Restricted-Access-Data\aearep-4400`.
- If there is a ZIP file (looks like a folder, but is not), right-click and choose `Extract All` before working in the folder

How to use the data in this folder

There are two situations.

1. The folder contains ONLY the confidential data. This *should* normally be the situation…
2. The folder contains a copy of the openICPSR deposit, but with the confidential data included.

There are two ways to run code using the data in this folder:

Run all programs from within this folder

1. Copy all programs to this folder, do not take the data out of this folder.
2. However, you will need to transfer modified code, log files, and output back to your regular "cloned" folder (i.e. `aearep-3756`), so that you can commit all changes (including to code files) back to the repository (this may require manual tracking of changes).

Run from regular folder, reference data in restricted-data folder

More robust, but maybe a bit more work, is to modify the code to reference the confidential data every time it is called.

- In the `config.do`, is a line `global sdrive ""`. Modify that line to read `global sdrive "L:\Restricted-Access-Data\aearep-3756"`
- Run the code in its usual location. When the code encounters (absent) confidential data in the usual location, it will break/stop.
- Everywhere you encounter references to confidential data in the code, e.g., `use "${datadir}/mysuper.dta"`, modify the code to reference the L-drive: `use "${sdrive}/mysuper.dta`.
- commit all code modifications, output, and log files as you normally would.

Examples

These are *simple* examples. Most situations will be more complex.

Stata Example 1

Some replication packages make this very straightforward, usually by the inclusion of a global specifically set for a "`confidential`" data folder. It may contain an **original** master file that looks something like this…

Original master file

```
*                                        Master File                                                      *
*
**********************************************************
**********************************************************

*******************************************
*** Set-up the directories and install packages
*******************************************

***set the working directory here:
global path "C:/Users/author/path"

*** The raw data folder contains two subdirectories, one for the public data and one for the confidential
global rawdata "$path/data"
global public "$rawdata/public"
global confidential "$rawdata/confidential"

global code "$path/code"
global results "$path/output"
```

Adjusting for LDI specific situation

1. Include the `config.do` in the master file as usual.
2. In the `config.do`, adjust the (existing) global for the L: drive: `global sdrive "L:\Restricted-Access-Data\aearep-4991-nda_Implicit"`. The global is defined at the very end of the `config.do`.
3. In the master file, incorporate the `rootdir` from the `config.do` as the main global `path`. This should reflect your workspace on the U: drive.
4. Incorporate the new global `sdrive` to set the authors' global `confidential`.

Updated master file

```
*                                        Master File                                                      *
*
**********************************************************
**********************************************************

include "config.do"

*******************************************
*** Set-up the directories and install packages
*******************************************

***set the working directory here:
global path "$rootdir"

*** The raw data folder contains two subdirectories, one for the public data and one for the confidential
global rawdata "$path/data"
global public "$rawdata/public"
global confidential "$sdrive/confidential"

global code "$path/code"
global results "$path/output"
```

Globus Transfers

Globus is a file transfer service that allows you to transfer large amounts of data between different systems. It is particularly useful for transferring data between BioHPC and CCSS Cloud, as well as between different research institutions.

There are two components to using Globus: the web interface, and (in our case) the taskbar icon on CCSS Cloud.

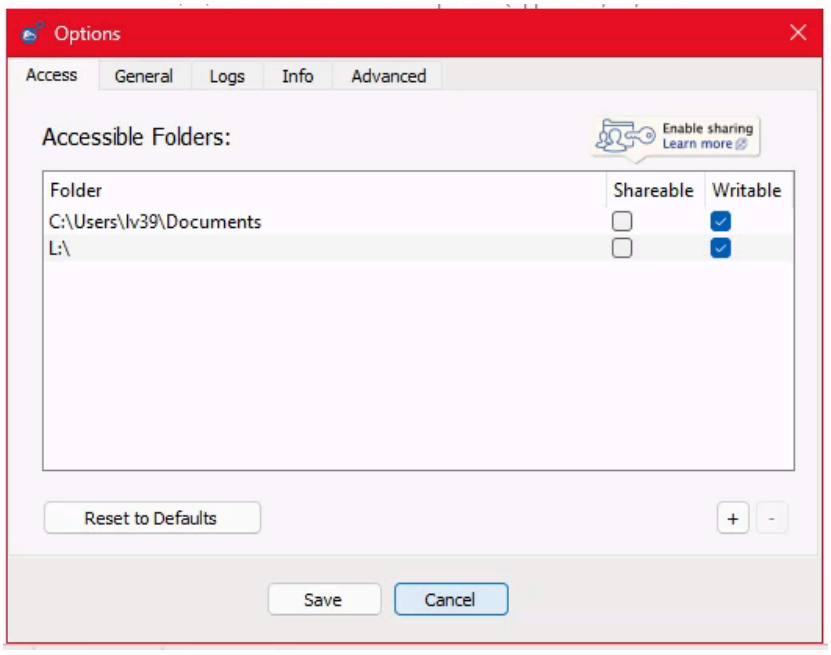
Globus setup

- create a Globus account
- configure the personal Globus endpoint on CCSS Cloud (call it `CCSS for LDI-AEA`)

Globus on CCSS Cloud

Look for the Globus icon on the desktop:

- Click on it. If it says “Another instance is running”, you are all set.
- Now look for the Globus icon to the right in the Windows taskbar. Right-click on it and select `Options`:



- Ensure that it contains at least the following entry:

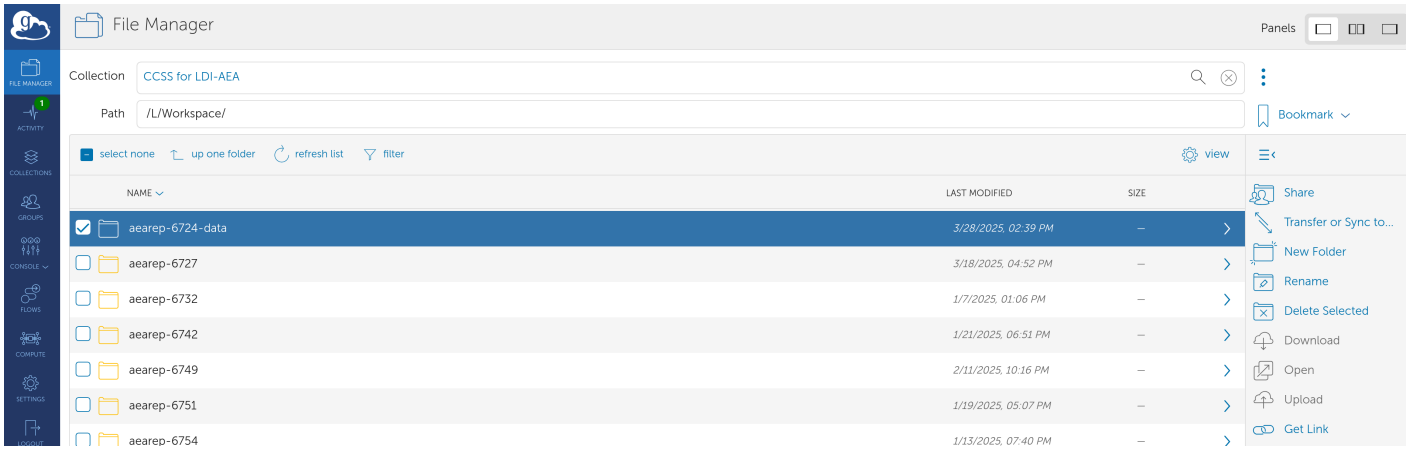
```
L:\
```

Transferring files

Transferring files is triggered from the Globus web interface. You will need to log in with your Cornell credentials. You can then select the source and destination systems, and transfer files between them.

Select the CCSS “endpoint”

In the navigation pane, you should be able to search for `CCSS for LDI-AEA` or whatever you called it in the `Collection` field. By default, it will open on the first entry in the list configured earlier. Go “up one folder”, and you should see “/L/”. Select it, and navigate to the folder where the files or folders are located you wish to transfer.



On the right panel, you will see `Transfer or Sync to...`. Select it. You will see a second pane, similar to the first one. In the `Collection` field, search for “biohpc cornell”. You should see

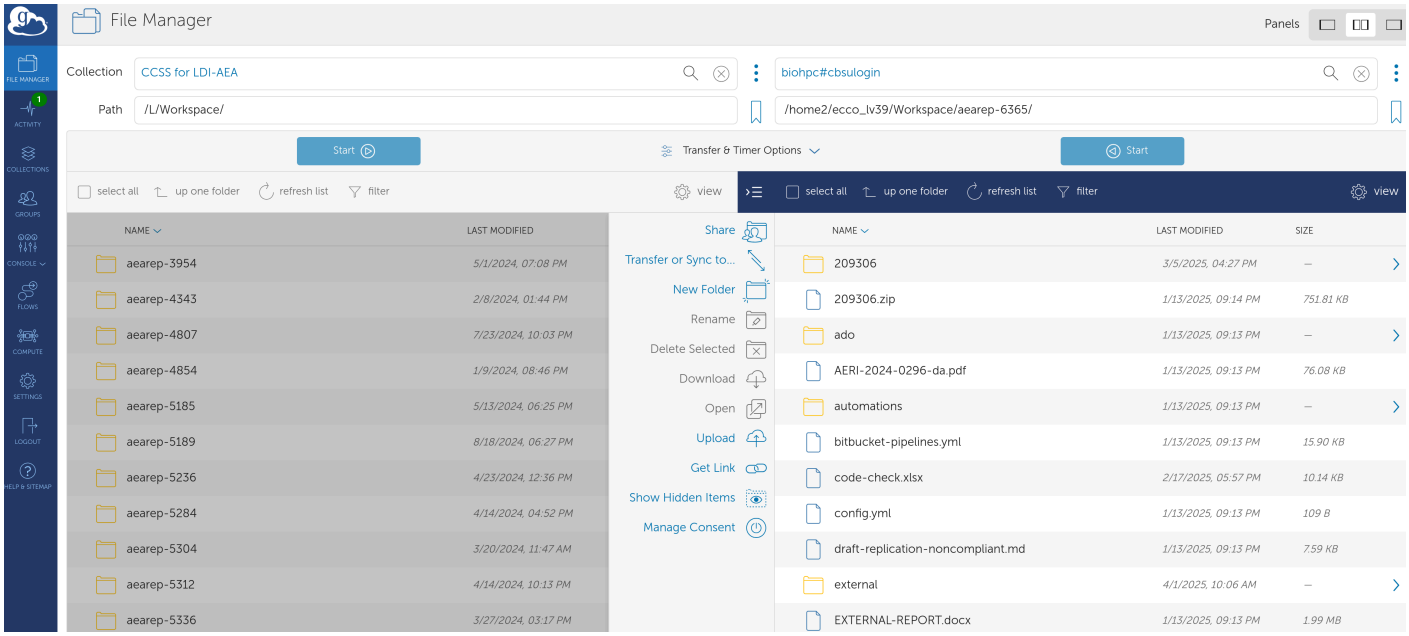
```
biohpc#cbsulogin
biohpc#cbusulogin2
biohpc#cbsulogin3
```

Select any one of them, you may need to authenticate with your BioHPC login credentials.

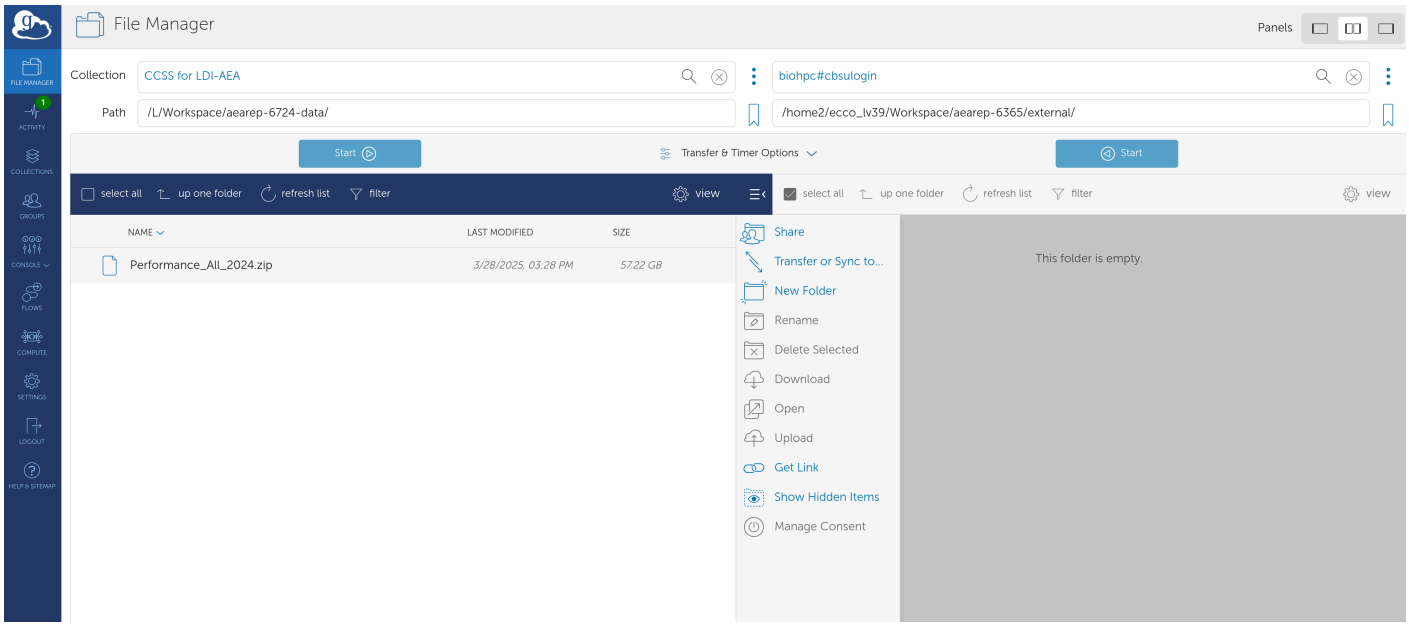
In the `Path` field, type `/home2/ecco_lv39/Workspace`. You should now see the list of directories there. Navigate to where you want to put the file. For instance,

```
/home2/ecco_lv39/Workspace/aearep-xxxx
```

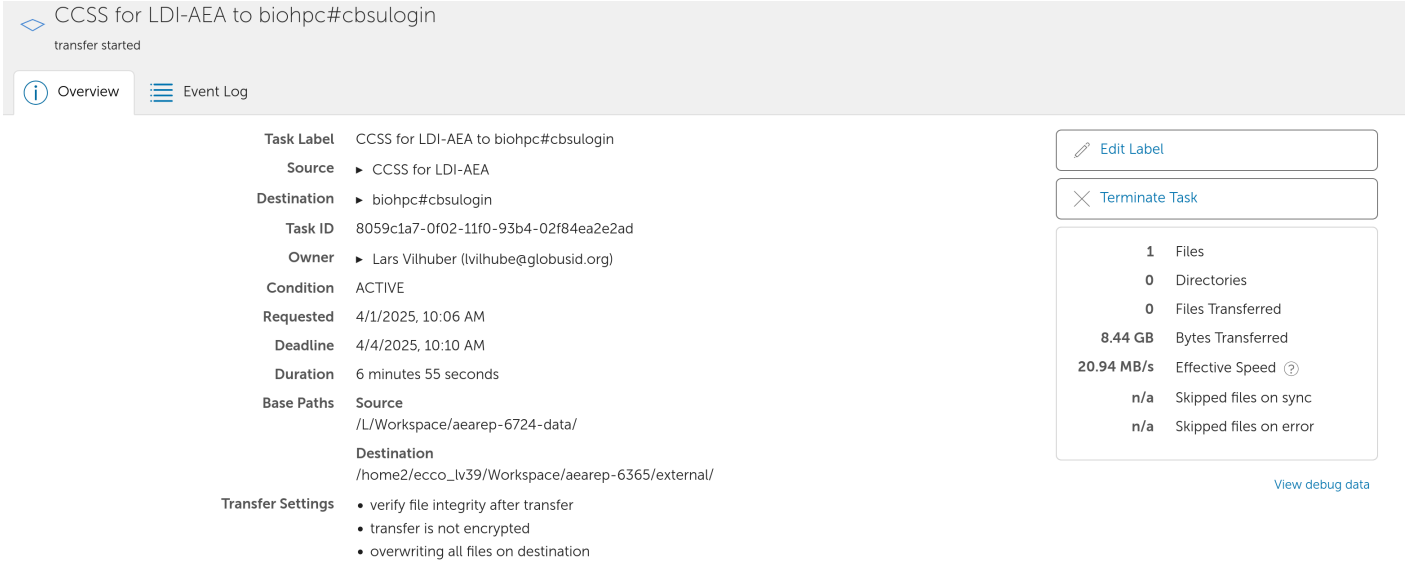
where `xxxx` is the number of the replication package you are working on.



Now start the transfer in the direction you wish to transfer, see the `[Start]` button above the pane where you want to transfer files FROM.



Your transfer will be queued. The pop-up will allow you to view the progress of the transfer (click on it):



Warning

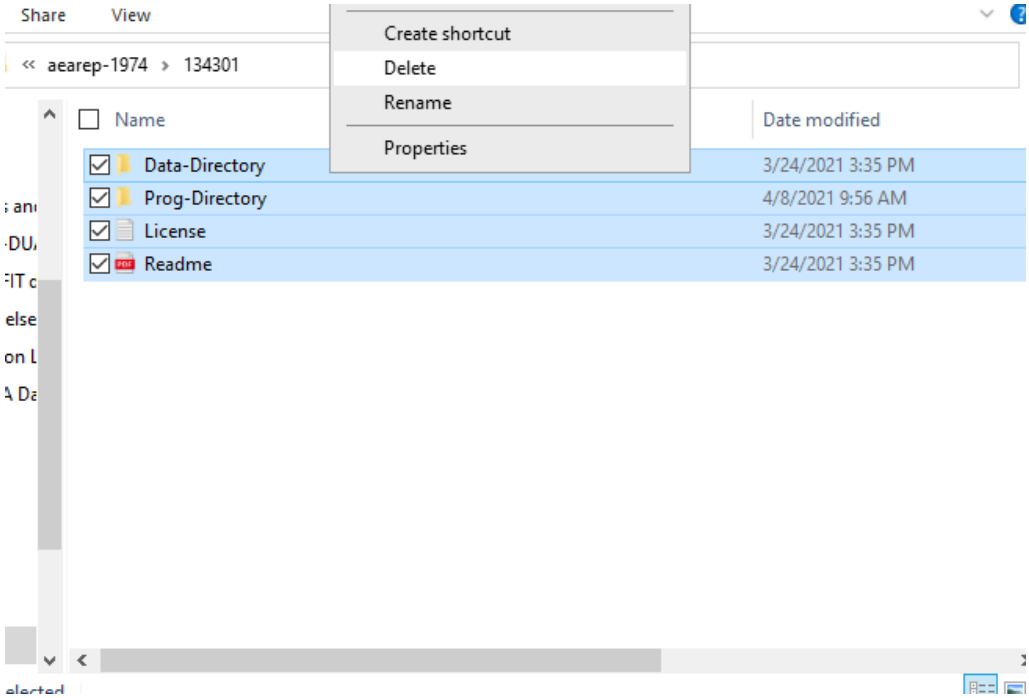
While the transfer is in progress, you should NOT log out of the CCSS Cloud instance. If you do, the transfer will be interrupted.

Updating Replication Materials after Revisions

While you move through the revision workflow, you will find instructions to download the updated ICPSR deposit and commit new files to the Bitbucket repository, while removing old files that are no longer present. This appendix will detail the most efficient way of executing this process.

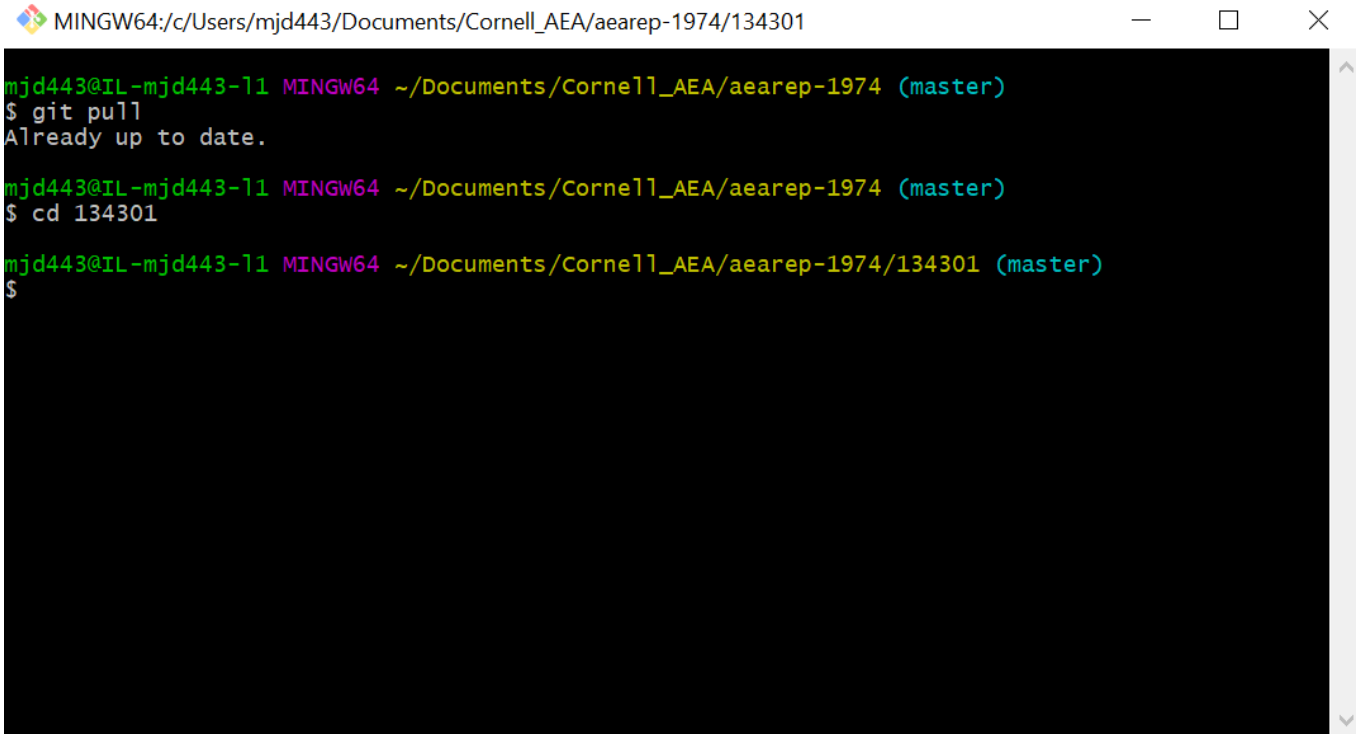
1. On the ICPSR deposit, navigate to “View Log” under “Share Project” and “Change Owner.” This log will tell you all the changes that have been made to the deposit and when. Verify that changes have been made to the deposit since the recent revisions were requested. If no changes have been made since the previous round, it may be a sign that the authors have created a new deposit instead of updating their existing deposit. If no changes have been made, contact a supervisor.
2. Download the ICPSR deposit.
3. Navigate to the existing `aearep-xxxx` directory on you workspace.
4. Within the directory on your workspace click into the ICPSR folder (i.e. `123456`).
5. Move any important files (results, logs, etc.) from the first round to their own subfolder of the root directory.
6. Navigate back to the root directory (`aearep-xxxx`), right click and select “Git Bash Here” to open a bash shell.
7. **Delete the files**. It is extremely important that you do this manually and **NOT** through “git rm”

- Option 1: Use the File Explorer (Windows) or Finder (macOS). Select everything and delete.

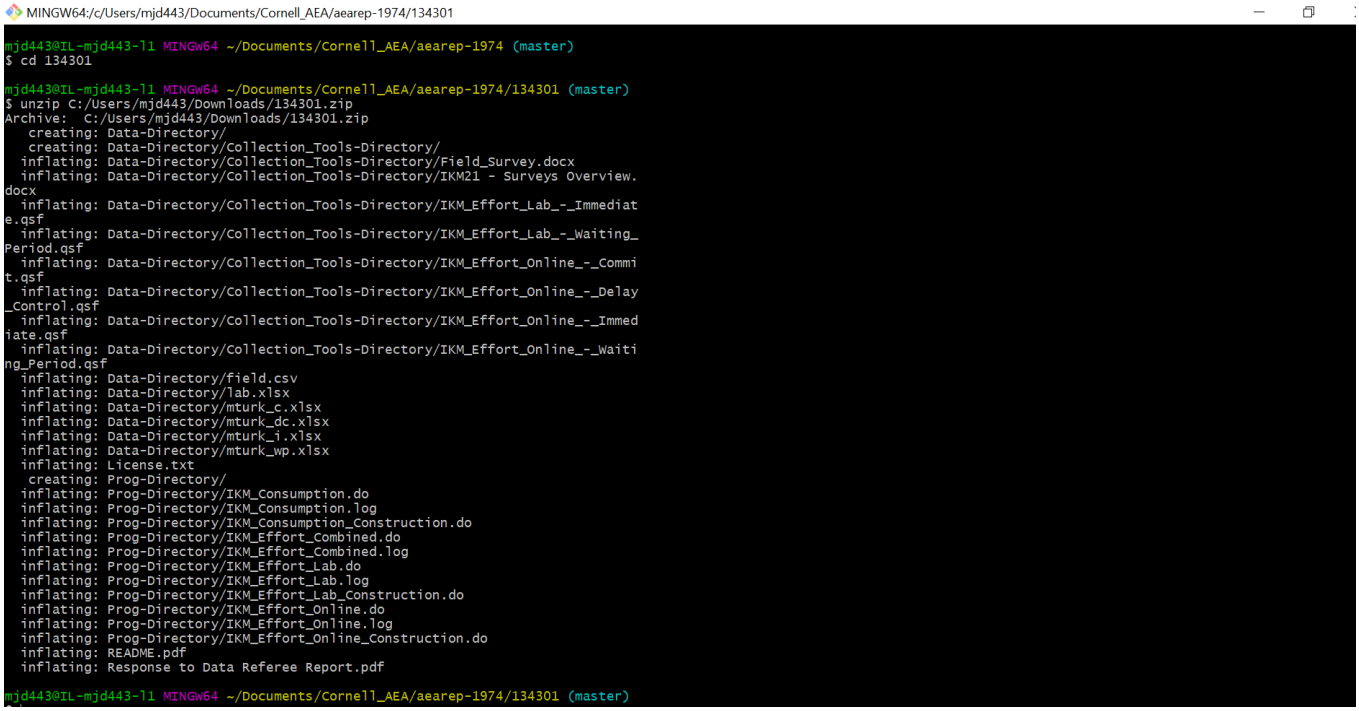


- Options 2: In the bash shell, type `rm -rf 123456/`

8. In the bash shell, cd into the ICPSR folder, `cd 123456` (you may need to recreate it: `mkdir 123456`)



9. In the bash shell, unzip the download of the updated ICPSR deposit `unzip path/to/Downloads/123456.zip`.



10. In the bash shell, type "`cd ..`" to move back to the root directory.
11. In the bash shell, type "`git status`" to see what files are new, modified, or deleted.
12. Git add everything in the ICPSR folder, e.g. `git add 123456`. This will add all new files, commit the deletion of any obsolete files, and record the changes in any files that were modified or moved
13. Git commit with the "-a" flag: `git commit -m "My great message" -a`. This will capture any previously manually deleted files.
14. Now push the updated ICPSR folder.
15. In Bitbucket, you should be able to see the specific changes made to any files (with the same naming convention), that were modified.

TL;DR

```
rm -rf 123456
mkdir 123456
cd 123456
unzip ../123456.zip
cd ..
git add 123456
git commit -m 'Updates by author' -a
```

Note: If the structure of the directory is different or files have been renamed, git will treat them as new files instead of allowing you to view specific modifications.

Access to Computers

You will be working on a variety of computers

- your laptop (Windows, OS X, Linux) to access the various resources
- a computer to run the code

    - your laptop (but not necessarily)
    - a Windows remote desktop
    - possibly a Linux cluster
    - WholeTale (experimental)
    - CodeOcean
    - Github Codespaces
- Bitbucket

General considerations

You only need to use **one** of them, but which one may depend on the particular circumstances of the project you are working on.

What can you do where

In principle, assuming you have the necessary software, you can work on any computer. Just remember to `git push` all changes back to Bitbucket.

Where is the data

We are currently exploring how to make the data *mobile*. At present, we do not want the data in Bitbucket, so you will need to re-download the data on each computer you intend to **run programs**. So you should decide on one particular computer. You can edit documents and code anywhere else, but again, remember to `git push` any changes from your *other* computer, and to `git pull` on your *compute* computer.

What software

By default, you should expect to run the code on , which has a broad selection of software.

If you actually have the software on your laptop, you should feel free to run code there, but see the caveat below. We will not purchase software for your personal laptop, and we do not provide you with a computationally capable laptop.

Some software is not available or installable on all systemS. If you encounter the following, you should check with your supervisor:

Software	Computing environment
Dynare	Red Cloud Windows node, CCSS, CodeOcean
ArcGIS	CCSS
Fortran compiler	BioHPC linux cluster, Docker
C compiler	BioHPC linux cluster, CodeOcean (depends on compiler)
Eviews	Not currently available
RATS	Not currently available (trial license for laptop)

Much statistical software loads data into memory. Your laptop has a limited amount of memory (in 2018, between 2GB and 8GB, rarely more). CCSS-RS nodes and BioHPC nodes can have between 256GB and 1024 GB of memory!

What if the code runs for a long time / I need to run to class / I need my life = Twitter back on my laptop

One of the advantages of running on the CCSS-RS or BioHPC nodes is that you can *disconnect* from the server, while leaving your programs running. That is one of the reasons to use them instead of your laptop.

Requesting Access

See setup checklist.

Details

The following sections will walk you through the various options.

Connecting to remote Windows servers

We have access to three sets of remote desktop Windows servers:

- Cloud CCSS (beta)
- CCSS-RS (classic)
- RedCloud (summer only)

> **This can be confusing!**
>
> Pay attention to the precise access instructions, as they may be substantially different for each server.

Connecting to different remote computers

Cloud CCSS | **CCSS-RS classic** | **RedCloud Server**

Important

- **Instructions**

Please be sure to do this:

- use the L drive for all “Workspace” folders (i.e., the clone of the Bitbucket repository)
- use the D drive for those that are large and need fast storage (but be aware that D drive is also wiped)
- **ALWAYS** `git commit` and `git push` all changes before you logout, every time you log out.

Warning

Please note that anything saved in the C drive (Documents, Desktop, or Download folders) and D drive may be deleted at any time (reboot/security update), and there is no way to recover the deleted files.

Please be sure to follow all instructions, in particular the ones about mapping network drives.

You can connect to CCSS Cloud Computing via a **web browser** or via the a Remote Desktop (RDP) client. Note that for some strange reason, Windows users need to use a DIFFERENT RDP client than the one that comes with their system (and used for CCSS classic). See the instructions provided by CCSS and these older instructions, as well as this link at Microsoft for the right client for your laptop’s OS. Web browser instructions are below, for convenience.

- Open the latest version of a web browser (Chrome, Safari, Firefox or Edge). · Go to following link: https://windows365.microsoft.com/ent#/devices.

If those fail… Click to hide

use the https://client.wvd.microsoft.com/arm/webclient/ (V1) client or the newer one https://client.wvd.microsoft.com/arm/webclient/v2/index.html (V2)

- Sign into Microsoft with your regular Cornell University email and password.
  - Cornell email address. Ex: jrg363@cornell.edu
  - Cornell email password. Ex: Cornell password
- Click on “Research Servers”, and on the pop-up, click on Allow to proceed.
- Sign in (again) with your regular Cornell University email and password.
  - Username: Cornell email address. Ex: jrg363@cornell.edu
  - Password: Cornell email password. Ex: Cornell password

Mapping Network Drives

- You should map the following network drive, following the CCSS storage/network-drive instructions linked above. However, use `L:`, not `Z:`. `L:` is the LDI-specific drive letter for this setup.

- `\\ccssilr.file.core.windows.net\lv39` to drive letter `L:` (you can call it “LDILab Drive”)
- We will be using that drive letter often.

Signing out or disconnecting

- If you have set a replication package’s code to run, **do not sign out/ log off** - disconnect.
- If you are done for a few days, and have nothing running, then **sign out**.

Disconnecting

Close the browser tab, or close the application by the usual methods. This will leave your code running!

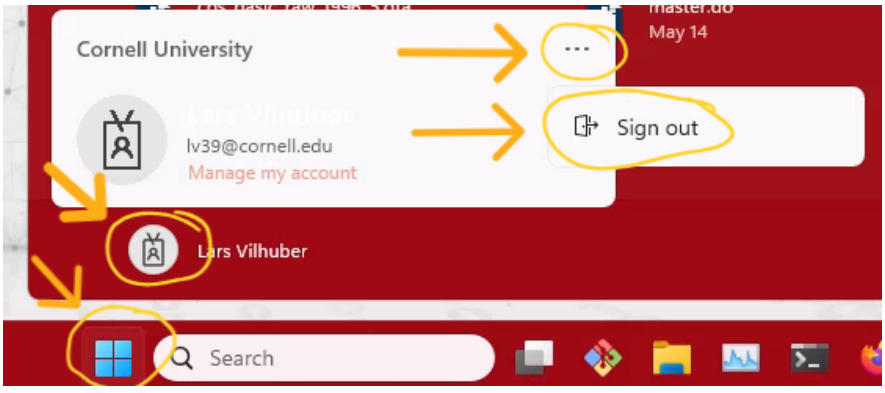
Signing out

It is important to **sign out** when you do NOT have jobs running. When you no longer have a job running, it saves everybody resoures. Your data will still be accessible when you sign back on.

Manually Signing out

Current method | **Old method**

- Open Start Menu
- Click on the Profile icon of your name on the left.
- Click on the three dots at the top right of the popup menu
- Select ‘Sign out’



Automatically Signing out

To configure your job to sign out automatically at the end, these are the instructions provided by CCSS:

Stata | **R** | **Matlab** | **Python**

```
/* Use the code below at the bottom of the Stata
   "main" or "master" script to automatically sign out */

shell shutdown -l
```

Connecting to remote Linux servers

We have access to various Linux clusters:

- BioHPC
- Occassionally NBER linux servers
- Others, as provided by authors

First-time setup

Run this ONCE the first time you ever access any Linux servers:

```
echo "umask 007" >> $HOME/.bash_profile
```

Then do the usual Bash setup. That should work on nearly any Linux server.

BioHPC | **NBER servers**

Request an account

1. Go to the BioHPC account request page, and create an account on the BioHPC cluster.
2. Then contact BioHPC support, requesting to join the ECCO group and lv39 (Lars') "lab" (`ecco_lv39`).
3. If you are logging on from outside the Cornell network you will need to setup Two Factor Identification.

Consult the ECCO general documentation for more details on the BioHPC cluster for economists!

Access a node

See Getting Started Guide and Remote Access. SSH is the best path (if you don't need graphical applications). See Access via VSCode for a more user-friendly way to use SSH to access the server.

> **Tip**
>
> For first-time access, set up your bash environment.

Accessing Linux nodes with VSCode

- Check that you have installed the Remote-SSH extension on VSCode.
- Open VSCode and select the Remote-SSH extension from the Command Palette.
- Enter the host name when prompted. The host name should follow this naming convention:
  - BioHPC: "netid@cbsuecco##.biohpc.cornell.edu".
  - NBER: "loginid@nber##.nber.org".
- You may be prompted to "Select the platform of the remote host". If so, select the "Linux" option in the drop down menu.

> **Tip**
>
> If you get the following message in the bottom right: “Setting up SSH Host BioHPC: (details) Initializing VS Code Server”, then it may be waiting for you to enter your TFA code

- Enter your account password when prompted.
- Once connected,
  - `Open Folder` and navigate to your working directory.
  - open a new terminal using the “Terminal” option in the top menu of VSCode (or `Ctrl-``).
- You should now be able to work on the Linux server via command line.

Some benefits of connecting to BioHPC with VSCode: You can view/edit programs, check log files, and run jobs simultaneously in a given instance of VSCode. Note that you should still use `tmux` within the VSCode terminal, in case of a disconnect.

In particular, you can navigate to your working directory and `git clone` the Bitbucket repository (using either the command line, or VSCode prompt to `Clone Repository`). VSCode recognizes Git, so you can visually navigate through tracked and untracked files via the lefthand side menu.

For more information, see VSCode Remote Development using SSH.

> **Tip**
>
> A tutorial video (thanks to Lars’ former RA Ilona Khimey) is available at Cornell Video-on-demand.

Where to run code

BioHPC | **NBER servers**

You should use `/home2/ecco_lv39/Workspace` to clone your Bitbucket repository.

```
cd /home2/ecco_lv39/Workspace
```

> **Tip**
>
> For instructions on how to setup and run code see ECCO Notes

Running Code

Running code on Biohpc is not as simple as opening the Stata GUI. Instead, you submit jobs through the SLURM scheduler. A detailed walkthrough can be seen here.

Modules

[details on running modules]

SBATCH

To submit a job to the server, you create a script with a header. Below is an example from the tools folder in the replication template.

Tip Click to hide

You can use the template by copying it into the authors working space, e.g.,

```
cp tools/sbatch-shell.sh 12345/path/to/code/sbatch.sh
```

Then customize it to fit your needs.

```
#!/bin/bash
# Job name:
#SBATCH --job-name=RunStata
#
# Memory
#SBATCH --mem=32G
#
# Request one node:
#SBATCH --nodes=1
#
# Specify number of tasks for use case (example):
#SBATCH --ntasks-per-node=1
#
# Processors per task: here, 8 bc we have Stata-MP/8
#SBATCH --cpus-per-task=8
#
# Wall clock limit: adjust accordingly. Format is HH:MM:SS or DD-HH:MM:SS where DD are days.
#SBATCH --time=00:00:30
#
# Email?
# Probably do not need "--mail-user=youremail@cornell.edu"
# Just add your email to the file "$HOME/.forward"
#
#SBATCH --mail-type=ALL
#
## Command(s) to run (example):
#
# Stata example
#
/usr/local/stata16/stata-mp -b main.do
#
## Matlab - will run "main.m", output to "main.log"
## Assumes you have done the setup at https://labordynamicsinstitute.github.io/ecco-notes/docs/biohpc/slu
# module load matlab/2023a
# matlab -nodisplay -r "addpath(genpath('.')); main" -logfile main.$(date +%F_%H-%M-%S).log
#
# R example - caution with version and parallel processing
module load R/4.4.2
R CMD BATCH main.R main.$(date +%F_%H-%M-%S).log
```

Each item in the header specifies some detail for the server. Jobs submitted must specifiy levels of compute (such as memory required), time to run, . A good starting point for these details is an authors `README`.

To view the status of your jobs, you can run

```
squeue
```

or

```
squeue -u <netid>
```

Interactivly

Sometimes, it can be helpful to run code interactively, rather than through SBATCH, on the BioHPC shell. Details here.

The most important idea is not to run code on the login node of BioHPC. You must transfer over to a computing node, using

```
srun --pty bash -l
```

Here, you can load modules, run programs, etc.

Obtaining Data on Biohpc

From A Deposit

The instructions from the manual Deposit Download Instructions will work. Note that you need to use the scripts instead of doing it manually. That page also has a link to non-icpsr deposit instructions

From Another Source

The following page Obtaining Data on Biohpc has extensive instructions on how to get data onto Biohpc from other sources. When obtaining data on your personal laptop is easy, then using WinSCP will be straightforward.

Additional setup and tips-and-tricks

Utilize `tmux`

If the Linux server has `tmux` installed, use it to make a persistent session that survives disconnects.

> **Tip**
>
> Cheatsheet: https://gist.github.com/MohamedAlaa/2961058

1. Login via SSH
2. Launch tmux with a session name that makes sense, e.g. `tmux new -s AEAREP-xxxx`
3. Launch your Matlab, Stata, etc job

4. Disconnect from tmux: `ctrl-b d`. You don't need to press this both Keyboard shortcut at a time. First press "Ctrl+b" and then press "d".
5. Log out of SSH

Next time:

1. Login via SSH
2. Reconnect to your tmux session: `tmux a -t AEAREP-xxxx`
3. If you forgot what session, `tmux ls`

To save the output of a `tmux` session to a file, see https://unix.stackexchange.com/questions/26548/write-all-tmux-scrollback-to-a-file.

Unzipping large ZIP files fails

In some cases, unzipping large ZIP files (larger than 2GB) may fail:

```
error: invalid zip file with overlapped components (possible zip bomb)
```

If this is detected, you need to use a 64-bit version of a decompression program. This may vary by Linux host.

BioHPC | **Others**

Instead of `zip`, use `7z` as follows:

```
ICPSRNUM=123456
/programs/bin/util/7z x -O${ICPSRNUM} ${ICPSRNUM}.zip
```

(which is the equivalent to the zip command `zip -n ${ICPSRNUM} -d ${ICPSRNUM}`). The first option (`-O`) is an upper-case letter `O`, not zero.

Setting up SSH Agent for password-less login

The SSH protocol allows you to create a public-private key pair. You deposit the public key on the Linux server, and keep the (passphrase-protected) private key on your laptop. You can then configure a "ssh-agent" running on your laptop to store (temporarily) your passphrase in memory, and provide it every time you log in to the Linux server. This works very easily on Unix-like systems (macOS, Linux laptops), but is a bit trickier on Windows laptops.

macOS | **Linux (laptop)** | **Windows (laptop only)**

To be completed.

The next steps are common to all operating systems.

You will want to create a key, then transfer it to the Linux server. The following commands should do this (should work on both Powershell and Bash):

```
ssh-keygen -t ed25519 -C "For BioHPC"
```

`ed25519` is the encryption type, “For BioHPC” is an arbitrary comment for your own tracking of keys. When prompted, you should add a passphrase.

This should have created two files in your `.ssh` directory:

```
ls $HOME/.ssh
```

Add the new key to your SSH-agent (optional, but recommended):

```
ssh-add $HOME/.ssh/id_ed25519.pub
# you should be prompted for your passphrase.
```

You will want to transfer the `.pub` file to the Linux server. You can do this with the command `ssh-copy-id` (if it exists), or manually. We show you how to do this manually:

> **Note**
>
> You should open two terminals: one locally on your laptop, one remotely on the Windows server!

This creates a directory used by SSH:

```
# this is on the remote server
mkdir $HOME/.ssh
# stay logged in!
```

This transfer the public key to the remote Linux server

```
# this is on your laptop
netid=lv39  # adjust to be your own netid!
scp $HOME/.ssh/*.pub ${netid}@cbsulogin.biohpc.cornell.edu:.ssh/
```

You should now see the `.pub` key in your `.ssh` directory on the remote Linux server:

```
# this is on the remote server
# this should show the *.pub file
ls $HOME/.ssh
# this authorizes you to use the SSH key to log in:
cat $HOME/.ssh/*.pub >> $HOME/.ssh/authorized_keys
```

Now test it:

```
# This is run from your laptop
ssh ${netid}@cbsulogin.biohpc.cornell.edu
```

You should now be prompted for your SSH passphrase:

```
$ ssh netid@cbsulogin.biohpc.cornell.edu
Enter passphrase for key `C:\Users\netid\.ssh\id_ed15559.pub`:
```

Utilizing Aliases

Workflows on Linux Systems can be simplified by using bash aliases. The example below can exist on your personal machine

```
alias biohpc='ssh <netid>@cbsulogin.biohpc.cornell.edu'
```

Adding this to your `~/.bashrc` enables you to type `bihopc` in your terminal, and an ssh connection is opened.

Using `linux-system-info.sh`

Part of replicating packages is capturing as much information as we can. You will see in the replication report that we must specify the amount of compute our replication used, under `Computing Environment of the Replicator`. This can be captured by adding the following line to your SBATCH script.

```
echo "Linux System Info:"
<package>/tools/linux-system-info.sh
```

This will output the computing resources in the SLURM output file.

> **Tip**
>
> Additional tips-and-tricks can be found on the LDIlab wiki. These are focused on the BioHPC cluster, but may work on other servers as well.

Reproducibility Checks in Codeocean

The workflow should be the same up until the verification stage, at which point you will use Codeocean to run the reproducibility check. You can access Codeocean from your personal computer or on CISER.

Create a Capsule

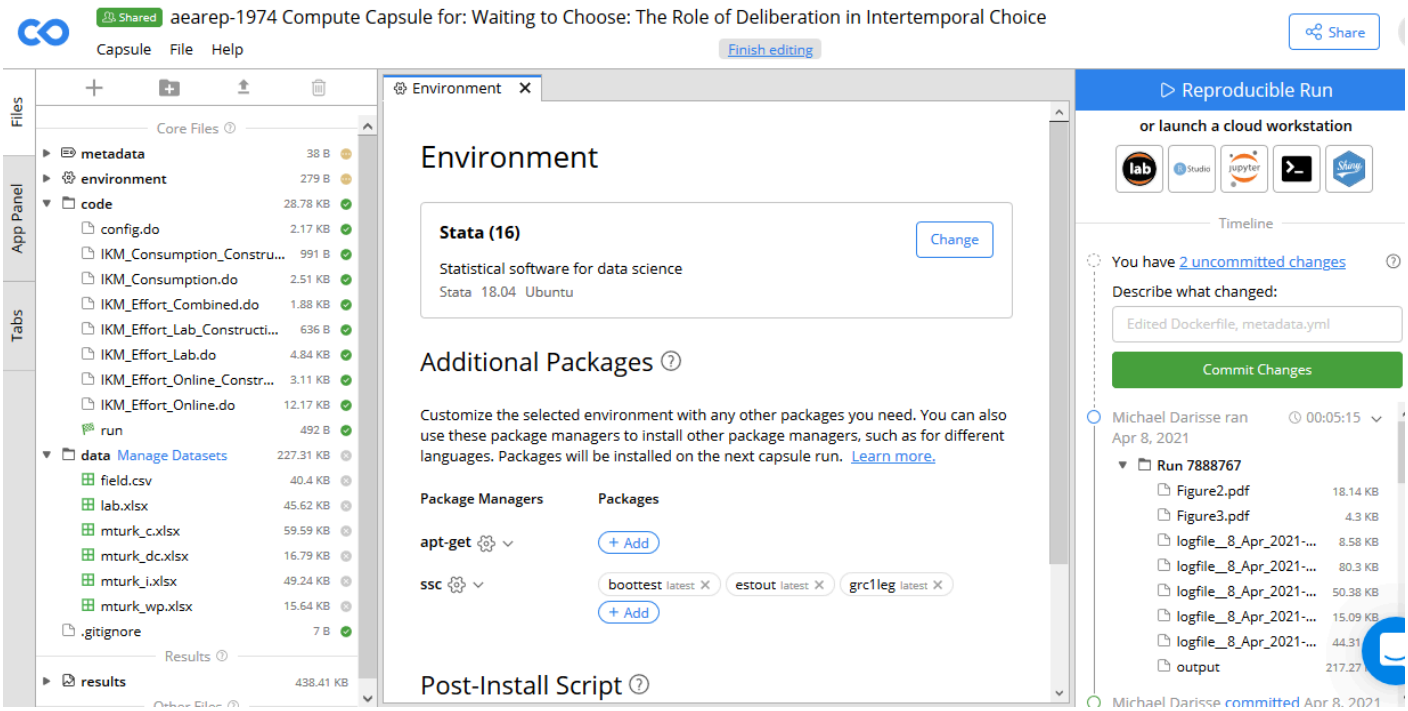
- First, create your own account on Codeocean using your cornell.edu email address (it's free).
- In Codeocean, create a new capsule. Name the compute capsule as "`AEAREP-xxxx Compute Capsule for: TITLE`" Example:

> AEAREP-1974 Compute Capsule for: Waiting to Choose: The Role of Deliberation in Intertemporal Choice

- Share your CodeOcean capsule with `dataeditor@aeapubs.org` (make sure to give `ownership` permissions).

Environment

- Set up the environment specified by the authors. This includes software, version of that software, and any dependencies (packages). Example: `Stata(16) with ssc packages estout and boottest`



Metadata

- Add a “Tag” in the metadata with the AEAREP number, and an additional tag with the manuscript number. These tags persist, even when we rename the deposit for publication, and thus allows us to keep track of things.

Recording the environment in Jira

In Jira,

- record the URL for the CodeOcean capsule (e.g., `https://codeocean.com/capsule/1665863/tree`) in the field `Computing URL` (`Data Info` tab)
- record the use of CodeOcean in the `Computing Environment` field (`Data Info` tab) (use the existing word `codeocean`)
- record the `Working location of the data` as “Codeocean”

Files and Directories

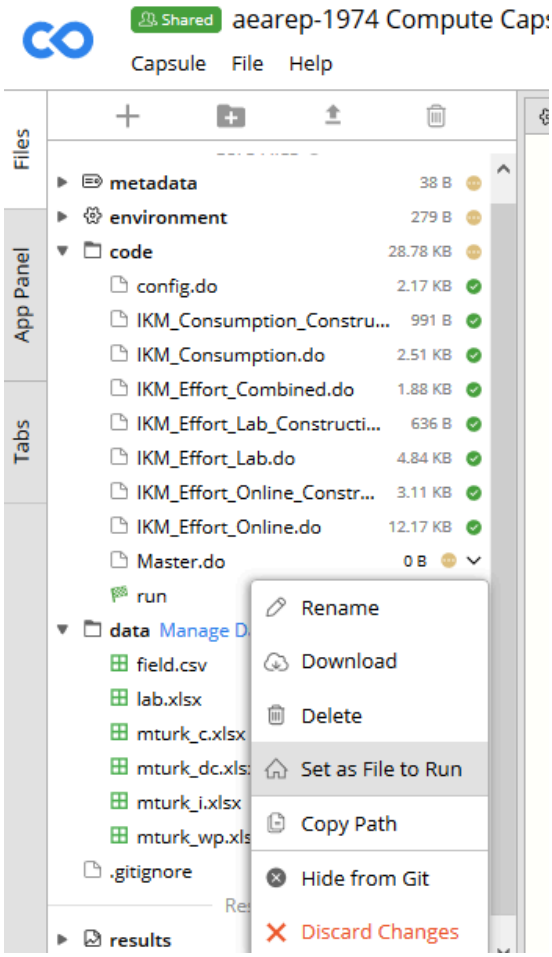
- There are only three directories in the Codeocean environment: `/data`; `/code`; `/results`.
  - All paths in the code must refer to one of these directories.
  - If the authors set their own globals, you change the globals to reflect these directories.
  - If the authors do not use globals, you should use the `config.do` (see instructions) to make the code more reproducible by setting globals in the `config.do` and amending the paths in the authors’ code. Example: `use "D:\Dropbox\Data for paper\x.dta"` becomes `use "$data/x.dta"`.

- Modify the standard `config.do` as follows:

```
local scenario "A"                      // line 35
global logdir "${rootdir}/results/logs" // line 59
// .... go to the very end
global data    "${rootdir}/data
global results "${rootdir}/results
```

Occasionally, other changes are also necessary, for instance, creating new directories.

- Upload the code to the `/code` and the data files to the `/data` directories in Codeocean.
- Define the default `run` file using the right-click menu for the main do file. This will create a new file (called `run`) which is a Bash script that CodeOcean uses to run all of the other code.



- Which file to use for the automatic creation of the `run` file depends on your manuscript:
  - If the manuscript has a single master file (e.g., `master.do`), use that. The `run` file will then look like

```
#!/usr/bin/env bash
set -ex

# This is the master script for the capsule. When you click "Reproducible Run", the code in this file
stata-mp -q do master.do "$@"
```

  - If the authors provide instructions on a specific order in which to individually execute programs, each program will need to be added to the “run” file in that order. In this case, you should be including the `config.do` at the top of each

program instead of only the `Master.do`.

```
#!/usr/bin/env bash
set -ex

# This is the master script for the capsule. When you click "Reproducible Run", the code in this file
stata-mp -q do 01_step1.do "$@"
stata-mp -q do 02_step2.do "$@"
stata-mp -q do 03_step3.do "$@"
stata-mp -q do 04_step4.do "$@"
```

Log files

- Any code that generates logfiles (this includes the `config.do`) needs to be amended to write to the `results` directory (i.e. `global logdir "${rootdir}/results"`).
- Note that CodeOcean automatically generates a capture of all screen output in a file called "`output`" in the `results` directory. In many cases, this will be sufficient.

Output

- Nothing is captured unless it is *explicitly* exported to the `/results` directory.
  - For example, you will not see any graphics files saved by the code unless they are saved as something such as `${results}/figure1.gph`.
  - You may not need to do this for intermediate data files. The program(s) should still run as long as the files are successfully created.
  - All tables and figures need to be adjusted to write to "`${results}`". The best way is to use a global (yet again defined in the `config.do`), e.g. `graph export "${results}/figure1.pdf"`.

Notes

- Note that some features will not work on CodeOcean if they rely on graphical windows (see LINK TO CODEOCEAN SITE)
  - A particular feature missing from Stata 16.0 on CodeOcean is the ability to write PNG graphics. Write PDF instead.

Recording edits

- Once you have finished editing and running the code in Codeocean, you should download the edited code and commit it to the repository. The best way to do this is similar to the steps taken during a revision:
  - Delete, by hand (**NOT** git rm), the code files as you downloaded them onto your workspace.
  - Then, place the amended codes into that same directory.
  - Git `add, commit, push`.
  - We should then be able to identify the changes you made to the code in order to run it on Codeocean. See further guidance here.

Expert tip

Instead of manually downloading CodeOcean results, in a (git) bash shell on your laptop or CISER, cd to the base directory of the reproducibility check (e.g., “aearep-xxxxx”), and run

```
tools/sync-codeocean.sh nnnnn
```

where `nnnnn` is the numerical capsule identifier from the CodeOcean URL: `https://codeocean.com/capsule/` **nnnnnn** `/tree`. You will be asked to authenticate using your CodeOcean (not Bitbucket!) login and password!

This will create two directories in your workspace: `codeocean-nnnnn-live` and `codeocean-nnnn`. The former will be “ignored”, but the latter will be refreshed each time you run the `sync-codeocean.sh` command, and will be committed to the Bitbucket repository. This is a convenient way to synchronize the two.

Don’t forget to `git add; git commit; git push` afterwards. This command can be run multiple times, as changes are made on CodeOcean.

Recording results

- Download the `/results` directory as a ZIP file. Remember the “run” number from the CodeOcean interface (e.g. “Run 1234567”).
- If not already present, create a “results” directory in the `codeocean-nnnnn` directory.
- Create a directory for each run with the “run” number.
- Unzip the ZIP file you just downloaded into the numerical directory you just created.

This should look like this:

```
aearep-1234/
    codeocean-nnnnn/
        code/
        metadata/
        results/
            2931206/
                output
                logfile_20220201-1201-root.log
                figure1.pdf
```

- Don’t forget to add all results from the “results” directory, `git add; git commit; git push`.

Conducting reproducibility checks on WholeTale

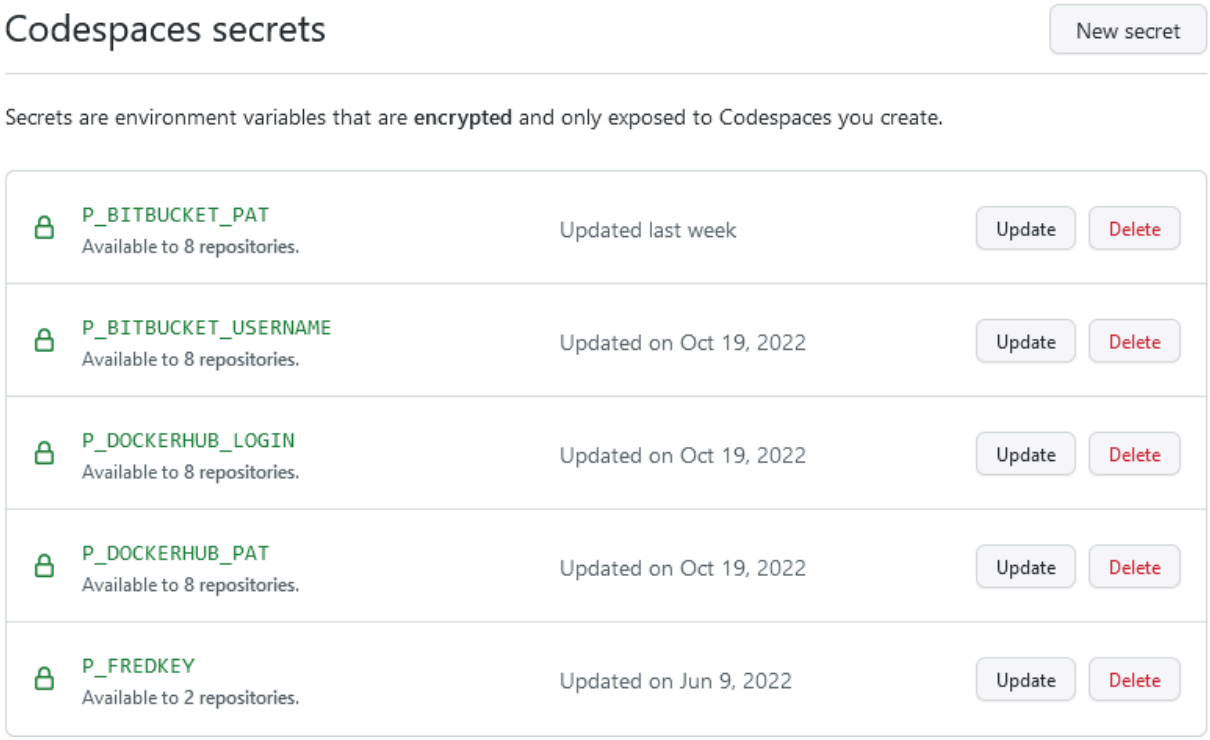
See instructions (experimental)

Computing using Github Codespaces

Setup

- Go to your personal Codespace settings on Github

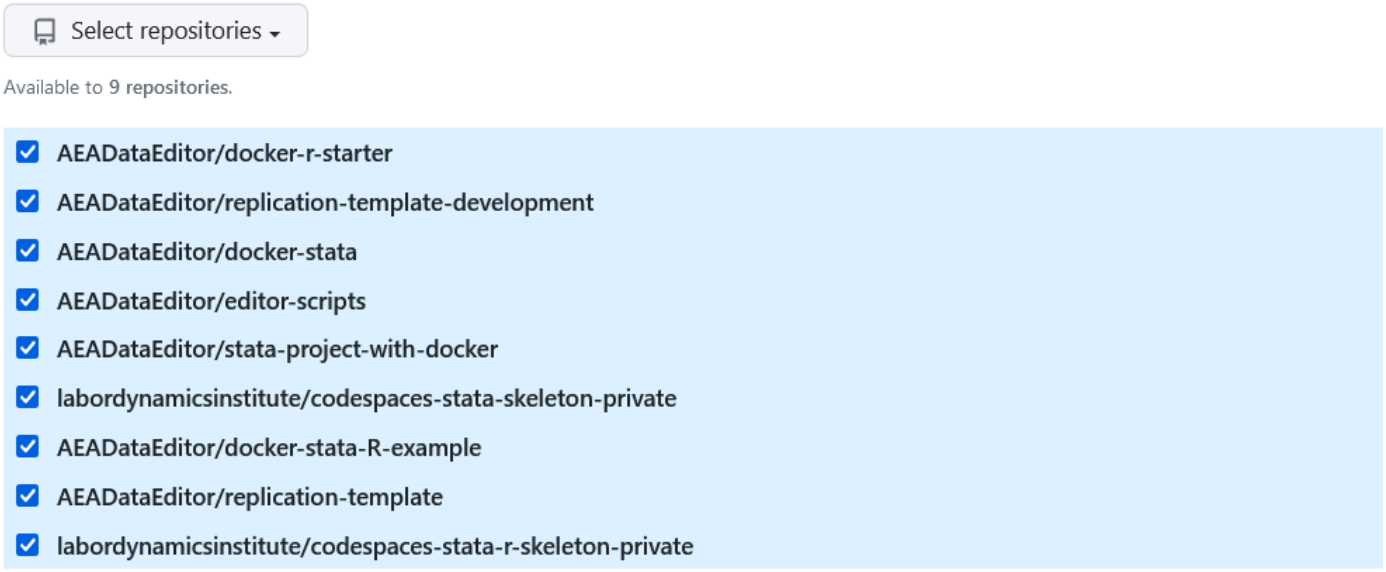
- Add the following `Codespaces secrets` by choosing `New secret`:



- `P_BITBUCKET_PAT`: Your personal access token from Bitbucket (see here)
- `P_BITBUCKET_USERNAME`: Your login on Bitbucket

The others are only needed in specific circumstances.

Then choose to apply them to specific repositories:



Start an environment on Github Codespaces

- Go to labordynamicsinstitute/codespaces-stata-skeleton-private and select “Code -> Codespaces -> Create Codespace on main”
- When you have one running, you can re-use the previous one. It will show in the popup, or on codespaces.

Alternate:

- Go to codespaces and select `labordynamicsinstitute/codespaces-stata-skeleton-private` or `labordynamicsinstitute/codespaces-stata-r-skeleton-private` from the dropdown menu (you may need to search).

If neither of those options appear, contact the LDI Lab Administrator.

Connect to local VS Code (optional but useful)

- Click on the green `Codespaces` button on the bottom left, choose “Open in VS Code” from menu that appears at top center.
- This will open a local VS Code instance with the same content. Your main window in the browser may or may not stay open.

Processing cases

- Open a terminal (top menu, Terminal, New Terminal, or shortcut `shift-ctrl-``.
- `cd ..` to be in `/workspaces`
- You can clone a Bitbucket case as usual.
  - special command available: `aeagit [NNNN] http` where `[NNN]` is the `AEAREP-NNNN` number. The command should open a new VS Code instance, with the cloned repository in the file pane.
- All command line git functions should work, as should command line Stata.

Administrator instructions

In order to enable a replicator to use this repository for Codespaces, they must be “collaborators”. Apparently, that requires write access (to be verified).

Older instructions

(experimental instructions)

Setting up your Bash environment

The following setup applies to any computer you may be using to run Bash commands (except for CodeOcean).

> **Note**
>
> On Windows, this will be “Git bash”. On Linux, you are usually using `bash`, but check with your sysadmin if you are unsure. On MacOS, newer versions have changed the default shell to `zsh`, which should work mostly the same. It is possible to use the `bash` shell there as well.

Preparing the BASHRC

Open up VS Code

Open up a VS Code window as follows:

```
code $HOME/.bashrc
```

- copy the above exactly as shown. There is a “dot” before the word “`bashrc`”.

> **Warning**
>
> - If this code shows an error, stop here, and debug!
> - If no VS Code windown shows, stop here, and debug.

You should now have a (new) VS Code window, either empty or with some pre-written script. If there is content, place your cursor at the very end of the edit window (you may need to scroll down).

Now, copy-paste the following code into the VS Code window. We will edit the values with the appropriate replacements. Keep all the line breaks, quotes, and spaces (or absence thereof) as shown!

```
# env for ICPSR
export ICPSR_EMAIL=mylogin@cornell.edu
export ICPSR_PASS='supersecretpwd'
# env for Bitbucket
export P_BITBUCKET_PAT='supersecretPAT'
export P_BITBUCKET_USERNAME=bitbucketusername
```

> **Note**
>
> The use of single-quotes for the password ensures that special characters are correctly preserved.

Get the Bitbucket PAT

First, create a Bitbucket PAT. Once you have created it,

- paste the PAT into the line with `P_BITBUCKET_PAT` (remember to keep the single-quotes!)
- also put your `bitbutcketusername` (if you can’t find it, see in the Bitbucket Profile (top-right corner, gear icon, etc.)).

Get the openICPSR info

Next, find your openICPSR login (should be your NetID + `@cornell.edu`) and your **openICPSR** password (not your Cornell or Google password)

> **The ICPSR password is *not* your NetID or your CMail/Google password!** Click to hide
>
> It must be set separately, by invoking the "Forgot Password" functionality. See openICPSR authentication for more details.

- paste the openICPSR login (probably `netid@cornell.edu`) into the line with `ICPSR_EMAIL`
- paste the openICPSR password into the line with `ICPSR_PASS`

Why?

This will

- allow you to use the `aeagit` shortcut to download a Bitbucket repository to your workspace directly from the Bash command line
- allow you to use the `tools/download_openicpsr_private.py` script to download replication packages from openICPSR from the Bash command line (see Helpful scripts for details on this and other scripts).

Verifying it works

> **Warning**
>
> On Windows, close the Git Bash window and open a new one to make the changes take effect. On Linux and MacOS, you can just run the following command in the terminal.
>
> ```
> source $HOME/.bashrc
> ```

You should now be able to verify that the configuration setup worked, by typing the following at the terminal prompt:"

```
export | grep BIT
```

should show your Bitbucket username and PAT.

```
export | grep ICPSR
```

should show your openICPSR login and password.

Now clear the confidential information from your screen!

Just to be sure, now type

```
clear
```

Configure some convenience scripts

We have a bunch of scripts, some of which can make your life easier. See AEADataEditor/editor-scripts. You can make these available to your Bash shell by running the following command:

If you do not yet have a `$HOME/bin` directory

Check first if you already have a `$HOME/bin` directory:

```
ls -l $HOME/bin
```

If that yields an error, then you don't have one. So run the next part:

```
git clone https://github.com/AEADataEditor/editor-scripts $HOME/bin
cd $HOME/bin
./install.sh # installs some Python scripts
```

You should now have access to the various scripts, such as `aeagit` (a bash script) and `aea-parse-tags` (a Python script).

Note

The scripts are actively updated frequently. To update the scripts, run

```
cd $HOME/bin
git pull
./install.sh # will re-install and update Python scripts
```

If you already have a `$HOME/bin` directory

If you *do* have a `$HOME/bin` directory, you will need to manually adjust a few more things. Contact your supervisor.

Other software dependencies

CCSS Cloud | **Other Windows** | **MacOS** | **Linux**

You should have all software that is needed. If you need additional software, remember that you **cannot** install software yourself. Reach out to your supervisor before contacting the CCSS Cloud support team.

Configuring Python defaults

The last step you do once you have cloned your first repository (this can be run from within **any** recently cloned repository).

If you are on a machine that has Python installed, run the following command (if it fails with `python3`, replace with `python`). You should do this once, from any recently cloned Bitbucket repository (which will contain a `requirements.txt` file). You do NOT have to do it every time!

Windows | **Linux/macOS**

When running in Bash, this should work:

- Change the working directory to one of your recent cases, say `xxxx`:

```
cd /l/workspace/aearep-xxxx
```

then

```
python -m pip install -r requirements.txt
```

Problems

Windows | **Linux**

Permission errors on CCSS Cloud

If you get an error about permissions on CCSS Cloud, such as the following or similar:

```
WARNING: Failed to write executable - trying to use .deleteme logic
ERROR: Could not install packages due to an OSError: [WinError 2] The system cannot find the file spec
```

then you will need to use Anaconda Python instead of the default Python installation. To do so, run the following commands in your Bash shell:

```
/c/ProgramData/Anaconda3/Scripts/conda init bash
```

You will be prompted for an admin password, which you should say “No” to. The command will report that it failed, but should have modified your `.bashrc` file to use Anaconda Python. Close that Bash window, and open a new one. It should display the word `(base)` at the beginning of the prompt, indicating that Anaconda Python is now active.

```
(base)
lv39@RS-CCSSlv39-16 MINGW64 ~
```

Now, go back to the repository directory (e.g., `cd /l/Workspace/aearep-xxxx`) and re-run the `python -m pip install -r requirements.txt` command from above.

Updating

Sometimes, updates are made.

Scripts

If you installed them in `$HOME/bin`, then simply run

```
cd $HOME/bin
git pull
```

Adjust if you installed them somewhere else.

Python dependencies

It is safe to re-run

```
python -m pip install -r requirements.txt
```

at any time.

Optional customizations

Windows: Add Git Bash to the Windows Terminal

It can be convenient to add the Git Bash to the Windows Terminal application that is present in Windows 10 and higher (better fonts, etc.).

Follow instructions at https://www.commandlinewizardry.com/post/how-to-add-git-bash-to-windows-terminal to do so.

Additional setup instructions

CCSS | **Github CS**

- You should synchronize your VSCode settings with your *normal* computer. This will bring over any settings and extensions! Very useful. See the VS Code Settings Sync page.
- You should do additional setup for your Git bash shell: this includes making downloads from openICPSR simpler. See appendix.

Authentication-related issues

For CISER-related authentication or logon questions, see CCSS-RS pages.

Bitbucket Authentication

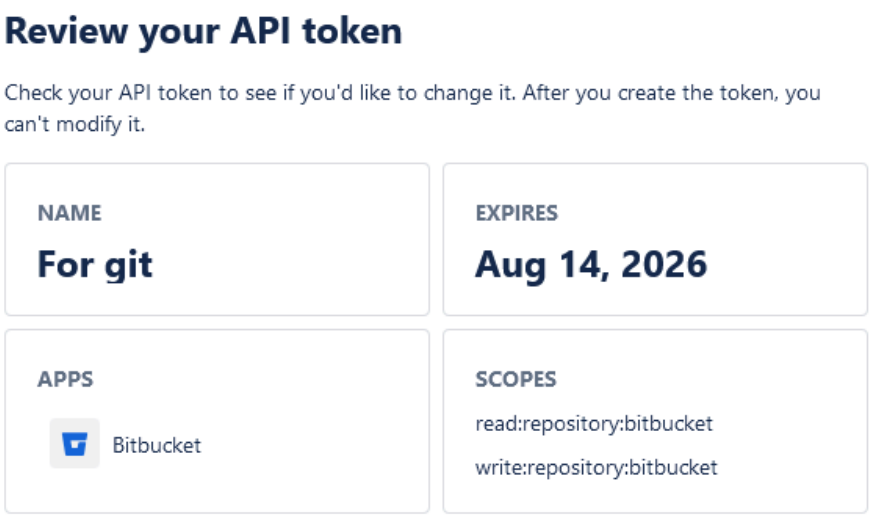
Creating an API Token (or PAT)

The long-term solution is to create an API Token[1]

1. Follow the steps to create an API Token. You can also try going to this page directly.
2. Click on **Create API token with scopes**.

- Name it something like “LDI Replication Lab Git”.
- Choose an expiration date that makes sense (minimum end of semester, maximum 1 year)
- You want to give it ONLY permissions to `read:repository:bitbucket` and `write:repository:bitbucket`.

Your confirmation screen should look like this:



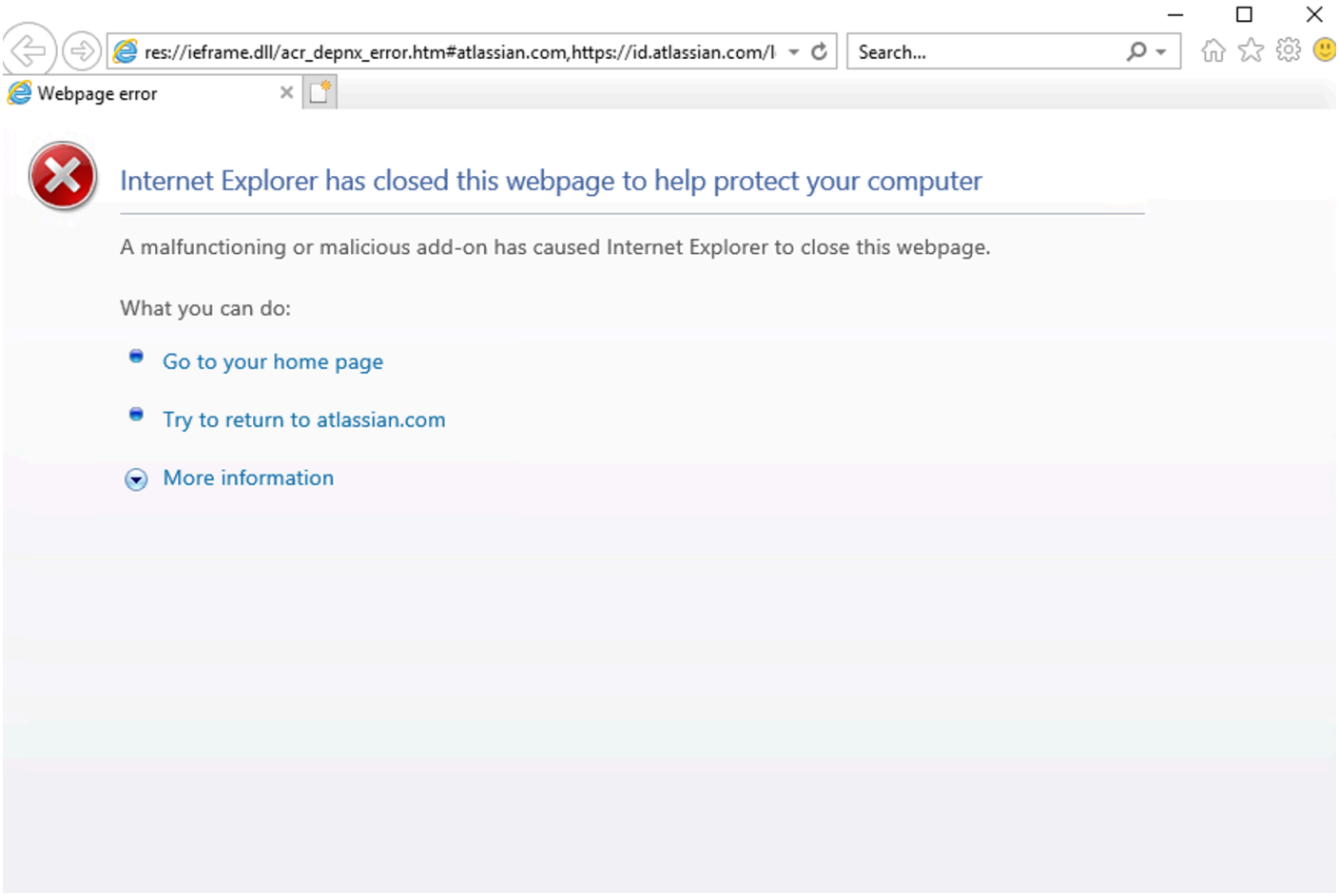
3. Click **Create**, and copy the long string of characters that appears to your password manager. Use it whereever it says to use a PAT or Bitbucket API token in this manual.

Some important points about API tokens:

- You **cannot** view an API token or adjust permissions after you create it, but you can create a new one (and delete the old one).
- You **cannot** use them to log in to your Bitbucket account at bitbucket.org.

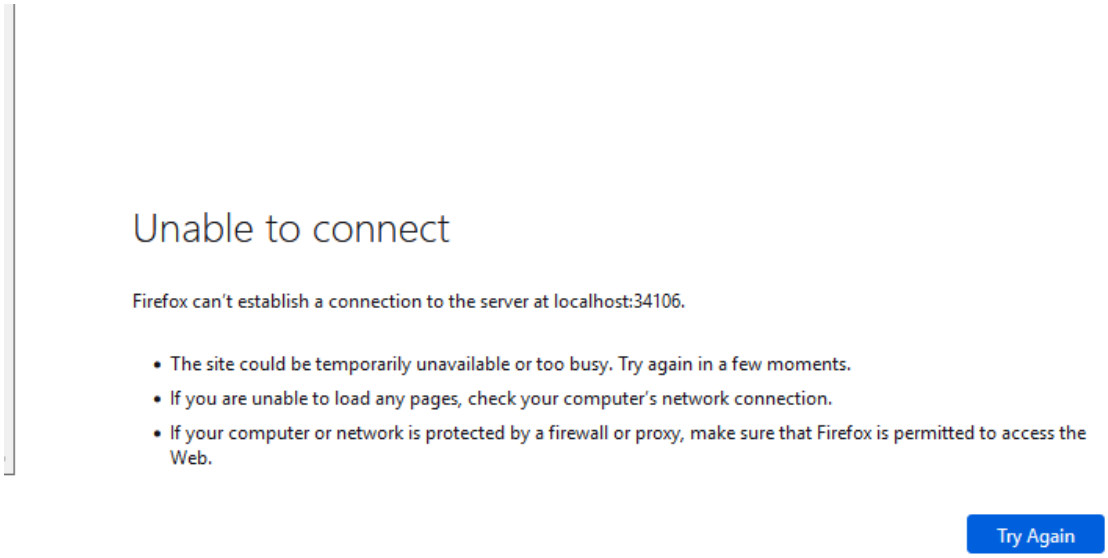
Possible errors

Any user who created a Bitbucket account after Semptember 13, 2021 may encounter the following error when attempting to git clone (or push) using Internet Explorer.



Solution

1. Make Firefox (or Edge) your default browswer.
2. Sign into Bitbucket on your browswer and then attempt the git action once again. You may then receive an error like below:



3. Attempt the git action again.

---

[1] This used to be an App Password, but they are being phased out as of 2025.

openICPSR Authentication

First-time

Go to https://www.openicpsr.org/openicpsr/ and in the top-right corner, choose `Login/Create an account`. You will be redirected to the ICPSR account web page.

- You should create an account with your `netid@cornell.edu`
- You can link your account with your Netid via Google student authentication BUT:
- You must still create a separate password for use by the automation scripts.

Warning

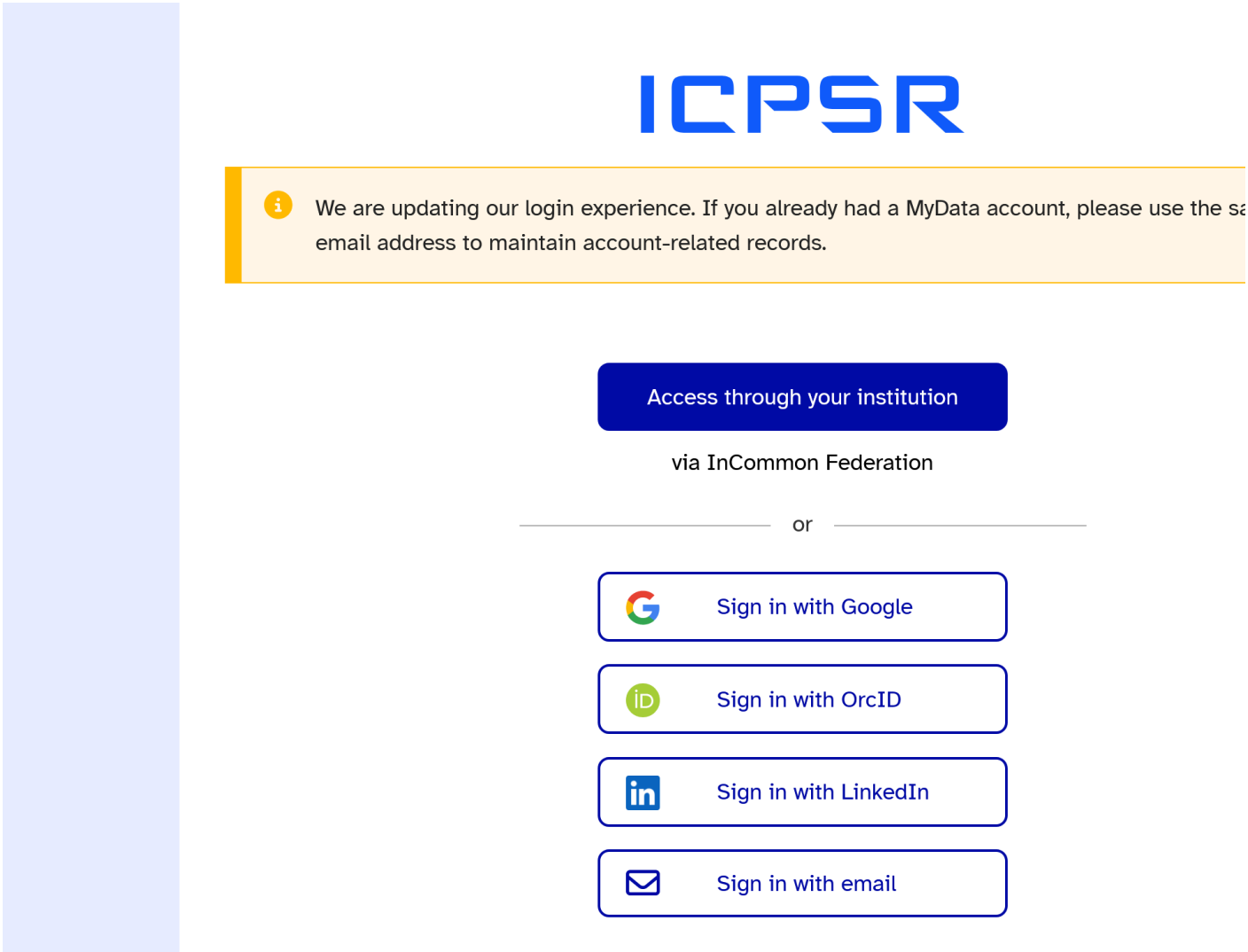
If you run the script and get an error message like the following:

```
Downloading file: icpsr-123456.zip
Traceback (most recent call last):
  File "/workspaces/codespaces-stata-r-skeleton-private/aearep-nnnn/./tools/download_openicpsr-pr
    with zipfile.ZipFile(outfile) as z:
  File "/usr/lib/python3.10/zipfile.py", line 1269, in __init__
    self._RealGetContents()
  File "/usr/lib/python3.10/zipfile.py", line 1336, in _RealGetContents
    raise BadZipFile("File is not a zip file")
zipfile.BadZipFile: File is not a zip file
```

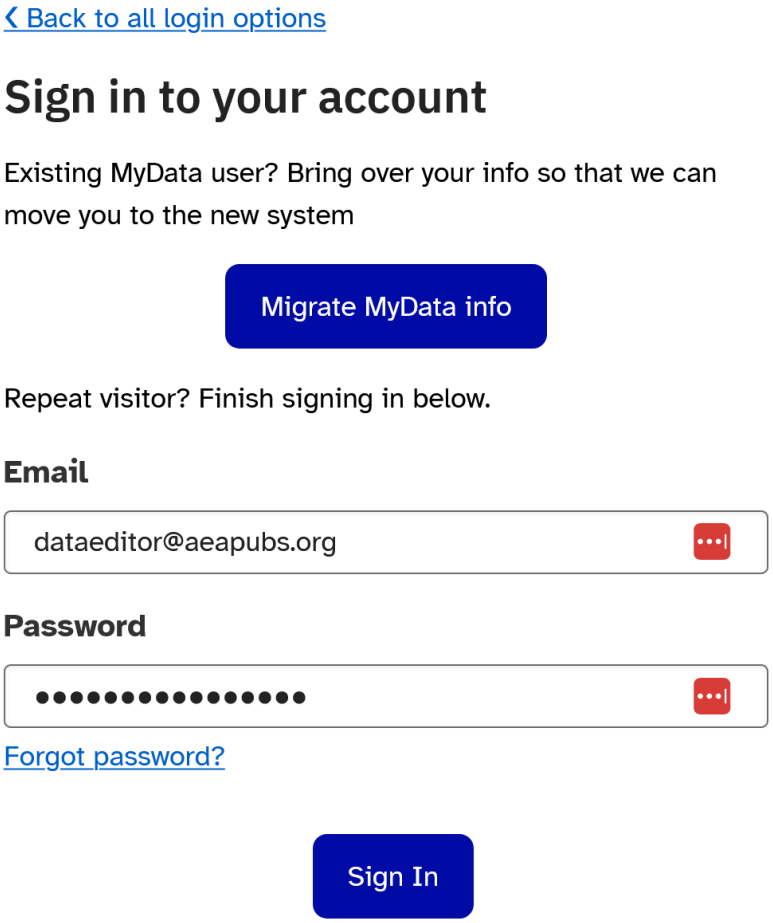
it means that you have not set up authentication as described below.

Creating an openICPSR-specific password when you linked your accounts

- Go to https://www.openicpsr.org/openicpsr/.
- If you are logged in (top-right corner shows your name), log out, or open the page in a different browser.
- On the login page, choose the “Sign in with email” option.



- Use the **same** email you used to create your account (i.e., `netid@cornell.edu`), but choose the “Forgot password” link



- Choose a password that is **NOT** the same as your Cornell NetID password!
- Now go to the Bash setup instructions, and change your password.

Various helpful scripts

The replication template provides a variety of scripts to assist with common tasks. These scripts are available in the `tools/` folder in each repository.

> Detailed documentation can be found at https://aeadataeditor.github.io/replication-template/tools/

Note

If you do not see a script referenced there, or the script does not behave as intended,

1. `git commit` all changes, and `git push` them
2. Run the `Refresh repository` script in Bitbucket.

Automation Explainer

We rely on a lot of automation to speed up and standardize tasks. This is usually handled by Bitbucket Pipelines, which is a continuous integration (CI) service built into Bitbucket. For more information on the available pipelines, see the Pipeline Overview. The actual pipeline configuration can be found on Github.

Bitbucket Pipeline Overview

> **Note**
>
> This page has been moved and can now be found at https://aeadataeditor.github.io/replication-template/automations/.

Bitbucket Pipelines Configuration

> **Note**
>
> This page has been moved and can be found at https://aeadataeditor.github.io/replication-template/automations/bitbucket-pipelines/.

Various templates for correspondence by AEA Data Editor team members

Preparing a Legacy Deposit for Updates by Authors

Post-publication, authors may wish to make changes to their deposit. In cases where the original deposit was created prior to July 2019, the deposit ownership should be assigned to the corresponding author.

When we receive such an email, the first step is to forward the email to dataeditor-queue@aeapubs.org to create a Jira issue. A record of all email correspondence should be kept on the issue by forward the emails to dataeditor-queue@aeapubs.org with the Jira issue in the subject line.

The next step is to respond to the author(s) with some form of the email below.

```
Let me outline the process and send the link to the AEA policy on Data and Code revisions (https://www.ae


In order for you to make the updates to the deposit, we'll do the following:

- Request a transfer of deposit ownership from the AEA to you (at which point you should receive a notifi
- Share the deposit with your co-authors, giving them full access rights (including the right to re-submi


Once we've completed the steps above, you will be able to update the deposit.

- The currently published V1 will remain available indefinitely, and will remain the version of record; t
- Do not create a second folder with "updated data" or similar; rather, simply delete any files that are
- One suggestion: once you have updated the README, move it to the root folder for increased visibility.
- Once you are done, we strongly suggest you also update the remaining metadata in the deposit (see https
- Once that is done, submit the deposit to us. We will check for completeness, and republish.

Please don't hesitate to ask if you have any questions.
```

Original Jira Issue

One of the most efficient ways to search for the original Jira issue would be to take the Manuscript Number from the ICPSR deposit and search for it in Jira. If an original Jira issue does exist, link the two issues.

Since in these cases the original deposit was created prior to July 2019, it is very possible that no original Jira issue exists (the paper pre-dates the current Data Editor process). In this case, post a comment to this effect on the new Jira issue and post another comment with the paper DOI.

Jira Fields

- Journal
- Manuscript Central Identifier
- openICPSR Project Number
- MCStatus should be “Update” (“Revision” is also acceptable)

Author Access

Necessary steps from our end:

1. Request a transfer of deposit ownership from AEA to the author. They should receive a notification from OpenICPSR.
2. Share the deposit with all authors of the manuscript, giving them full rights (including the right to re-submit).

Transition

There is a specific transition in the Jira workflow called “`Author submitted post-publication modifications`” which enables the user to move the issue from “`In progress`” straight to “`Pending openICPSR changes`.”

> Not sure what permissions are needed.

Checks

Ensure that authors have “submitted to AEA”, and that an updated README as well as any updated files are present.

In these cases we are only checking to verify what specific changes were made and that these changes are reflected in an updated README. The email correspondence should be checked, as should the logs. There should be no additional directories within the deposit called “update” or similar. See the revision policy.

The resolution should always be “`Evaluation only`.”

Moving to Publication

From here on, follow the usual workflow to publication of the deposit. In general, the AEA Data Editor, not the Editorial Office will publish the deposit.

Communicating to AEA Editorial Office

See Revision policy (link above). If the version of record is changed, this needs to be notified to the editorial office, so they can update the records. If a new version is created, no further action is needed.

IPUMS Beta API request

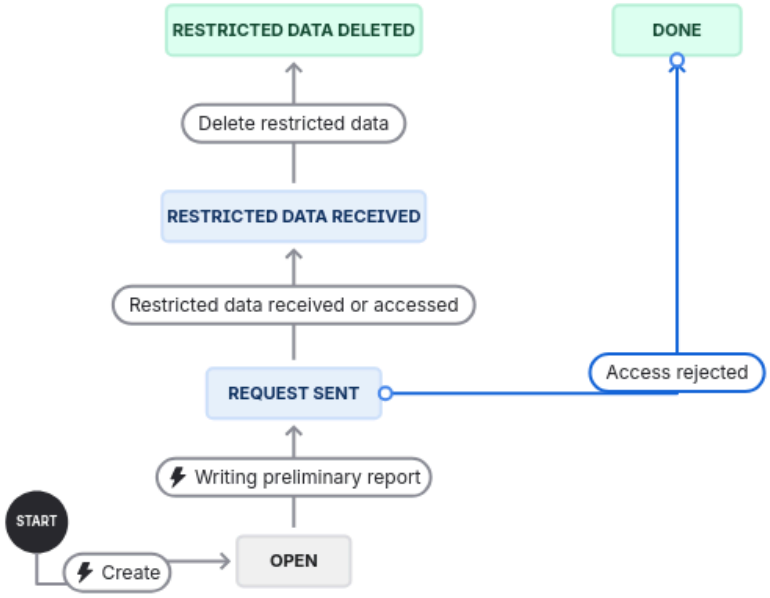
If the author uses IPUMS data, request access to the IPUMS API extract.

```
Your replication package uses IPUMS data. Provision of extracts is required for sample
(not full-count) files, and is compliant with IPUMS Terms of Use.

For the extracts you generated, you should provide

- the dofile for read-in "usa_0028.do" and dct files
- for sample files, the actual extract (dat or dta files)

However, IPUMS is trialling an API for extracts, which allows a user to export the extract
specification, and for a replicator to re-use the extract specification programmatically.
We are able to convert your stored extract specification into a downloadable file.
There are two ways forward for this:

-  you sign up for the beta (write to ipums+api@umn.edu and tell them you want to participate
in the beta, and that the AEA Data Editor sent you)
- alternatively, you can give us permission to access your data extracts. We will then handle
converting the extract and adding the files to your replication package.

Please contact the Data Editor (dataeditor@aeapubs.org) to let us know your choice of action.
```

Requesting Restricted-Access Data from Authors

Restricted-access data must be requested by the LDI Research Aide; it should not be done by the replicator.

The generic workflow (2026) is as follows:



For more details, see the AEA Research Aide manual.

Request for External Reproducibility Check (general)

See AEA Research Aide manual for details.

Clarification regarding privately-provided restricted access files

See AEA Research Aide manual for details.

Technical issues

What are EPS files?

Some authors have the code write out `.eps` files. These are "Encapsulated PostScript" files. In principle, Adobe Acrobat should be able to conver them, but they cannot be included in the `REPLICATION.md` file, nor can they easily be viewed.

Since October 2024, we have a pipeline script that can help you.

- First, be sure to `git add` all EPS files, then `git commit` them, followed by `git push`.
- Go to the `Pipeline` tab in Bitbucket for your repository, and
  - choose the `6-convert-eps-pdf` pipeline.
  - enter the `openICPSRID` that is being used (this field can also handle a deeper directory, such as `openICPSRID/replication_package`, but you will rarely need to use this).
  - by default, the pipeline will only convert `.eps` to `.png`, but you can also select `yes` in the `ProcessPDF` field if you need PDF files.

This will convert all `.eps` files, and create `.png` files in the same directory. You should then `git pull` in your local workspace to get the files.

File too big - rejection by Bitbucket

Bitbucket imposes a 100MB limit for files being pushed to the Git repo. This is similar to other Git services.

> ⚠ **Do not try to override this!**
>
> While in some cases, the right solution may be to use something like Git LFS, we do not need to use that here, except in very rare cases.

If the commit has not been pushed yet

This works if you have only a single commit where this is an issue, i.e., your `git status` shows something like:

```
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
```

First, find the file:

```
find /path/to/search -type f -size +100M
```

Because this lists all the files in the directory that are larger, but not necessarily the files that have been `git committed`, you may need to inspect the output from the `git show` command as well:

```
commit=<commit-hash>
git show --pretty=format: --name-only $commit
```

or

```
git show --pretty=format: --name-only $commit | while read file; do        ls -l "$file" | sort -nk 5;
```

where the largest files should be at the bottom of that list.

Then, remove the file from the repository, and then “amend” the previous commit. This assumes that the immediately preceding commit is the offending one. Adjust if not.

```
git rm --cached path/to/VERYLARGEFILE
git commit --amend
```

If there are more than 1 commits

If your `git status` shows

```
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
```

(or more), you will need to rewrite the history. This is a bit more complicated, but it can be done. The following steps will rewrite the history of the last 3 commits.

```
git rm --cached path/to/VERYLARGEFILE
git rebase -i HEAD~3
```

This will open an editor with the last 3 commits. Change the word `pick` to `squash` for the commits between when the large file was added, and when it was removed. Save and close the editor. You will then be prompted to edit the commit message. Save and close the editor. This will solve it on your local machine. You can now push.

If the commits have already been pushed

Sometimes, this happens after the commits have already been pushed, but Bitbucket has implemented a new policy (does not happen very often). You may then need to clean the historical commits. If not already done, first `git pull` so you get the complete history.

Then proceed as in the previous step. Once done, you will then need to force push to the remote repository.

```
git push origin master --force
```

File too big in history of repository

If there are various files scattered throughout the history of the repository, you will need to use a tool called `bfg-repo-cleaner`. This is a more advanced tool, and you should be careful when using it. It is not recommended to use this tool unless you are very comfortable with Git.

- Instructions to migrate to LFS

However, we do not want to migrate to LFS, rather, we want to delete the big files. See the command line option `-D` to `bfg`.

> **Warning**
>
> Danger! Do not proceed unless you **really** understand what you are doing!

Sample session:

```
# In the original cloned repository for aearep-xxxx
xxxx=4406
git rm data/path/to/VERYLARGEFILE
git commit -m 'Removing big files' -a
git push
cd ..
# The "--bare" is very important here!
git clone --bare git@bitbucket.org:aeaverification/aearep-${xxxx}.git
# remove all rds files
java -jar bfg-1.14.0.jar -D "*.rds" aearep-${xxxx}.git/
# remove files based on size
java -jar bfg-1.14.0.jar --strip-blobs-bigger-than 50M aearep-${xxxx}.git/
# The report will detail what was removed
cd aearep-${xxxx}.git
git reflog expire --expire=now --all && git gc --prune=now --aggressive
git push origin master --force
```

After cleaning, if the repository size is still too big, then it may be necessary to

- delete the repository completely on Bitbucket
- create a new (empty) repository on Bitbucket
- redo `git push origin master --force`

Sample report:

```
Cleaning
--------

Found 13 commits
Cleaning commits:       100% (13/13)
Cleaning commits completed in 90 ms.

Updating 3 Refs
---------------

        Ref                 Before     After
        ---------------------------------------
        refs/heads/master | bb7f0f55 | 6d879f82
        refs/tags/update1 | 2957723c | 644805e8
        refs/tags/update2 | dfe21158 | 95801e58

Updating references:    100% (3/3)
...Ref update completed in 17 ms.
...
                               Before     After
        -------------------------------------------
        First modified commit | 86b4bcc3 | 1098d6cb
        Last dirty commit     | f769cec6 | cfcaead8

Deleted files
-------------

        Filename                  Git id
        ---------------------------------------------
        aa_fc_m.rds             | f1c98be6 (56 B )
        aa_fc_nr.rds            | 7b82b717 (63 B )
        altmech.rds             | 9e02cdbc (2.9 KB)
        arrests-aa-outcomes.rds | bb6b87ba (17.0 MB)
        arrests_by_NUID.rds     | 12f4b2f2 (2.0 MB)
        arrests_by_NUID180.rds  | 1d65d2c9 (2.5 MB)
        base_cohorts.rds        | 03ac41c8 (80.0 KB)
        cncomp.rds              | b1691139 (6.0 KB)
        compare_gr.rds          | ed3a3380 (554 B )
        error_df.rds            | 0874a70c (600.8 KB)
        extra_comps.rds         | 1bcd3f3d (299.5 KB)
        fe_list.rds             | 018ba52c (349 B )
        fes_by_NUID.rds         | a7bd7f27 (3.7 MB)
        fes_by_NUID180.rds      | 4cfb0bae (453.2 KB)
        full_cohorts.rds        | e44c10c3 (1.5 MB)
        ...


In total, 27 object ids were changed. Full details are logged here:
```

Code of Conduct

Summary View

Below is a summary of LDI Replication Lab Code of Conduct. Continue reading for a more detailed description of the CoC.

> All team members of the LDI Replication Lab are subject to Cornell University Code of Academic Integrity and Policy 6.4 on “Prohibited Bias, Discrimination, Harassment, and Sexual and Related Misconduct.” For all reporting procedures, please consult those websites.

We are dedicated to providing a welcoming and supportive environment for all people, regardless of background or identity. By participating in this team, participants accept to abide by LDI ReplicationLab's Code of Conduct and accept the procedures by which any Code of Conduct incidents are resolved. Any form of behaviour to exclude, intimidate, or cause discomfort is a violation of the Code of Conduct. In order to foster a positive and professional learning environment we encourage the following kinds of behaviours in all platforms and events:

- Use welcoming and inclusive language
- Be respectful of different viewpoints and experiences
- Gracefully accept constructive criticism
- Focus on what is best for the community
- Show courtesy and respect towards other community members

If you believe someone is violating the Code of Conduct, we ask that you report it to Cornell University, who will take the appropriate action to address the situation.

Details

Part 1. Introduction

The LDI Replication Lab emphasizes participation of all team members in an equitable fashion. We value the involvement of everyone in the community. We are committed to creating a friendly and respectful place for learning and contributing. All participants in our work and communications are expected to show respect and courtesy to others.

To make clear what is expected, everyone participating in LDI Replication Lab's activities is required to conform to the Code of Conduct. This Code of Conduct applies to all spaces managed by LDI Replication Lab including, but not limited to, workshops, email lists, and online forums such as GitHub, Slack and Twitter.

The LDI Executive Director is responsible for enforcing the Code of Conduct. They can be contacted by emailing ldi-leadership@cornell.edu. Should a complaint concern the Executive Director, Cornell University should be contacted, as outlined by the Code and Policies mentioned above. All reports will be reviewed and will be kept confidential.

Part 2. LDI Replication Lab Code of Conduct

The LDI Replication Lab is dedicated to providing a welcoming and supportive environment for all people, regardless of background or identity. As such, we do not tolerate behaviour that is disrespectful to our teachers or learners or that excludes, intimidates, or causes discomfort to others. We do not tolerate discrimination or harassment based on characteristics that include, but are not limited to, gender identity and expression, sexual orientation, disability, physical appearance, body size, citizenship, nationality, ethnic or social origin, pregnancy, familial status, veteran status, genetic information, religion or belief (or lack thereof), membership of a national minority, property, age, education, socio-economic status, technical choices, and experience level.

Everyone who participates in LDI Replication Lab activities is required to conform to this Code of Conduct. It applies to all spaces managed by The Carpentries including, but not limited to, workshops, email lists, and online forums such as GitHub, Slack and Twitter. Workshop hosts are expected to assist with the enforcement of the Code of Conduct. By participating, participants indicate their acceptance of the procedures by which the LDI Replication Lab resolves any Code of Conduct incidents, which may include storage and processing of their personal information.

Part 2.1 Expected behaviour

All participants in our events and communications are expected to show respect and courtesy to others. All interactions should be professional regardless of platform: either online or in-person. In order to foster a positive and professional learning environment we encourage the following kinds of behaviours in all LDI Replication Lab activities and platforms:

- Use welcoming and inclusive language
- Be respectful of different viewpoints and experiences
- Gracefully accept constructive criticism
- Focus on what is best for the community
- Show courtesy and respect towards other community members

Note: See the four social rules for further recommendations.

Part 2.2 Unacceptable behaviour

Examples of unacceptable behaviour by participants at any LDI Replication Lab activity/platform include:

- written or verbal comments which have the effect of excluding people on the basis of membership of any specific group
- causing someone to fear for their safety, such as through stalking, following, or intimidation
- violent threats or language directed against another person
- the display of sexual or violent images
- unwelcome sexual attention
- nonconsensual or unwelcome physical contact
- sustained disruption of talks, events or communications
- insults or put downs
- sexist, racist, homophobic, transphobic, ableist, or exclusionary jokes
- excessive swearing
- incitement to violence, suicide, or self-harm
- continuing to initiate interaction (including photography or recording) with someone after being asked to stop
- publication of private communication without consent

Part 2.3 Consequences of Unacceptable behaviour

Participants who are asked to stop any inappropriate behaviour are expected to comply immediately. This applies to any LDI Replication Lab activity and platforms, either online or in-person. If a participant engages in behaviour that violates this code of conduct, the organisers may warn the offender, ask them to stop any activity they may be involved in, leave the platform, or engage university mechanisms to investigate the Code of Conduct violation and impose appropriate sanctions.

Update Logs

- 2021-08-19 Adapted from the Carpentries' Code of Conduct

About this Document

This document is adapted from guidelines written by the Django Project, which was itself based on the Ada Initiative template and the PyCon 2013 Procedure for Handling Harassment Incidents. Contributors to the the initial document are Adam Obeng,

Aleksandra Pawlik, Bill Mills, Carol Willing, Erin Becker, Hilmar Lapp, Kara Woo, Karin Lagesen, Pauline Barmby, Sheila Miguez, Simon Waldman, and Tracy Teal. Additional language was added by Otter Tech from the PyCon U.S. 2018 Code of Conduct (licensed CC BY 3.0) In 2018, the Code of Conduct was revised to add a summary, straightforward examples of both beneficial and unwanted behaviors, and evaluating intent. Reporting guidelines were also revised to include alternate contact points and a reporting form with the procedure was added. In 2019, an appeal process, the procedure for following up with a reportee, terminology, CoC incident response procedure, termed suspension checklist, expanded clauses for conflicts of interest, and committee membership agreement were included. Contributors of these revised documents are Ethan White, Kari L. Jordan, Karin Lagesen, Malvika Sharan, Samantha Ahern, and Simon Waldman.

Videoconferencing Rules and Etiquette

What you will need

- Your laptop
  - While a cell phone can work in a pinch, it is not ideal on a number of dimensions
- A web camera (if not already on your laptop)
- Headphones with a microphone
- A quiet place

Before entering the meeting

- Ensure your technology works correctly **BEFORE** the meeting.
  - If you've switched devices, find out if they are working. There is a feature in Zoom to test your microphone and speakers.
  - If you know your internet is slow, use phone dial-in for voice (and understand how that works)
  - Understand how the transmission speed works in your location - just because you can see others does not mean they can see you. Choose a location with stable Wifi, plugin an ethernet cable (if possible), use the phone dial-in so your voice is guaranteed to be heard.
- Be **on time** (not really a videoconferencing rule - it's always a rule!).
  - Being late on a videoconferencing meeting may be as disruptive as walking past the presenter in a in-person meeting!
- Be in a place where **you are not disturbed**
  - Don't videoconference from the coffee shop!
  - You need to be able to turn on camera (no cluttered bedroom background) and microphone (no chatter, no washing machine running).
- Wear work-**appropriate clothing**.
  - Assume that you will be seen head-to-toe.
- Frame the camera correctly, and look into the camera
  - Be able to look into the camera, not to the side.
  - Assume you are having your wedding photo/ graduation photo/ photo for your grandma taken.
- Have the right light.

    - Not too dark, not light from the background - we want to see you
- We may ask you to share your screen to help troubleshoot issues. Please familiarize yourself with this feature.

While in the meeting

- General rule: if you wouldn't do it in a face-to-face meeting, then you shouldn't do it in a virtual one.
- Speak clearly. Don't shout.
- **Mute your microphone if not speaking**.
    - But also know when to unmute your microphone!
    - Be mindful of your own desk noise in proximity to the microphone (keyboard, paper shuffling)
- Look into the camera. Keep the camera on!
- Leave the "self view" on to remind what you look like to others (no making faces)
- Pay attention. (That's why you keep the camera on!)
- When hiccups occur, handle them silently.
    - if your call drops, dial-in again - and when you are back, don't shout "I'm back".
    - if your voice or video connection breaks up - switch to using the phone dial-in option (silently).
- It is OK to
    - politely interrupt somebody if that person is particularly faint and you cannot hear them.
    - if in a big room, introduce yourself when speaking (the far end may not always be able to clearly see your face). This is not necessary when in a single-person location.
- Don't carry on side conversations.
    - Especially not with the microphone on!

Sources

- My own 22 years of doing this
- OWLLabs
- Inc.

References

[Mar14] M. (ed.) Martone. Data citation synthesis group: joint declaration of data citation principles. 2014. URL: https://doi.org/10.25490/a97f-egyk, doi:10.25490/a97f-egyk.

[AmericanEAssociation19] American Economic Association. AEA Member Announcements: Updated AEA Data and Code Availability Policy. July 2019. URL: https://web.archive.org/web/20191208160745/https://www.aeaweb.org/news/member-announcements-july-16-2019 (visited on 2019-09-21).

[AmericanEAssociation20] American Economic Association. Data and code availability policy. *AEA Papers and Proceedings*, 110:776–78, May 2020. doi:10.1257/pandp.110.776.

[FORCE1116] FORCE11. The FAIR data principles. Technical Report, FORCE11, 2016. URL: https://www.force11.org/group/fairgroup/fairprinciples (visited on 2017-05-26).

[ICPSRd.] ICPSR. Citing Data. n.d. URL: https://www.icpsr.umich.edu/web/pages/datamanagement/citations.html (visited on 2020-08-21).

Older versions

- Last version of 2023 (with Workflow v2)