Reproducibilidad en las ciencias sociales

Denise Laroze

6 de mayo de 2019

CESS - Universidad de Santiago
denise.laroze@cess.cl

Resumen de contenidos

¿Qué es una ‘Reproducibilidad’?

Definición

Reproducibility “refers to the ability of a researcher to duplicate the results of a prior study using the same materials as were used by the original investigator. That is, a second researcher might use the same raw data to build the same analysis files and implement the same statistical analysis in an attempt to yield the same results . . . Reproducibility is a minimum necessary condition for a finding to be believable and informative”

NSF 2015, citado en Goodman, Fanelli, and Ioannidis 2016, Science Translational Medicine Vol. 8, Issue 341, pp. 341ps12

Según esta definición una reproducibilidad es:

- es la capacidad de recrear o producir de nuevo los resultados de una investigación usando **los mismos datos** que el autor original
- requiere buenas prácticas de organización y distribución del material de investigación

Se **diferencia de Replicabilidad**, que es la capacidad de un investigador de duplicar los resultados de un estudio si se siguen los mismos procedimientos pero se recaban nuevos datos - Ver trabajos de Camerer 2016 y 2018.

Relevancia

¿Por qué tener buenas prácticas de reproducibilidad?

- Es requisito mínimo para que los resultados sean entendibles y confiables
- facilita el trabajo con coautores (algo cada vez más común)
- Aumenta la eficiencia y escalabilidad del trabajo propio, considerando que hoy los proyectos son cada ves más complejos y largos. Hay que poder volver años después al mismo código y poder introducir las correcciones requeridas.

Journals

Muchos de los journals más destacados hoy tiene políticas de reproducibilidad.

- Econometrica
- AER
- Science
- AJPS

Un repositorio muy relevante es Dataverse

Buenas prácticas

Fuentes de buenas recomendaciones para realizar estudios que sean reproducibles.

- Nagler (1995) Coding Style and Good Computing Practices texto
- Gentzkow y Shapiro: A Practitioner's Guide video, documento
- Research Replication: Practical Considerations texto

Algunas reglas clásicas de Nagler I

1. Rule: Maintain a lab book from the beginning of a project to the end.
2. **Use command files** that include identifying information, date, purpose.
3. Rule: Code each variable so that it corresponds as closely as possible to a verbal description of the substantive hypothesis the variable will be used to test.
4. Rule: **Errors in code should be corrected where they occur** and the code rerun.
5. Rule: Design each program to perform only one task.
6. Rule: Separate tasks related to **data manipulation vs. data analysis** into separate files.

Algunas reglas clásicas de Nagler II

7. **KISS Rule**: Do not try to be as clever as possible when coding. Try to write code that is as simple as possible.
8. Rule: Use a consistent style regard- ing lower- and upper-case letters.
9. Rule: Use variable **names that have substantive meaning**.
10. Rule: Use variable names that indicate direction where possible (eg. women, not sex).
11. Rule: Use appropriate white space in your programs, and do so in a consistent fashion to make the programs easy to read.
12. Rule: **Include comments** before each block of code describing the purpose of the code.
13. Rule: Include comments for any line of code if the meaning of the line will not be unambiguous to someone other than yourself.
14. Rule: Rewrite any code that is not clear.

Algunas reglas clásicas de Nagler III

15. Rule: **Verify that missing data are handled correctly** on any recode or creation of a new variable.
16. Rule: **After creating each new variable** or recoding any variable, produce f**requencies or descriptive statistics** of the new variable and examine them to be sure that you achieved what you intended.
17. Rule: When possible, **automate things** and avoid placing hard-wired values (those computed "by hand") in code.

“If your program is not worth documenting, it probably is not worth running.”

“The time you save by writing clean code and com- menting it carefully may be your own.''

Gentzkow y Shapiro: Experiencia práctica I

This manual began with a growing sense that our own version of this self-taught seat-of-the-pants approach to computing was hitting its limits. Again and again, we encountered situations like:

- In trying to replicate the estimates from an early draft of a paper, we discover that the code that produced the estimates no longer works because it calls files that have since been moved. When we finally track down the files and get the code running, the results are different. . .
- In the middle of a project we realize that the number of observations in one of our regressions is surprisingly low. After much sleuthing, we find that many observations were dropped in a merge . . . When we correct the mistake . . . the results change dramatically.

Gentzkow y Shapiro: Experiencia práctica II

- A referee suggests changing our sample definition. The code that defines the sample has been copied and pasted throughout our project directory, and making the change requires updating dozens of files. In doing this, we realize that we were actually using different definitions in different places, so some of our results are based on inconsistent samples.
- We are keen to build on work a research assistant did over the summer. We open her directory and discover hundreds of code and data files. Despite the fact that the code is full of long, detailed comments, just figuring out which files to run in which order to reproduce the data and results takes days of work. Updating the code to extend the analysis proves all but impossible. In the end, we give up and rewrite all of the code from scratch.

Gentzkow y Shapiro: Experiencia práctica III

- We and our two research assistants all write code that refers to a common set of data files stored on a shared drive. Our work is constantly interrupted because changes one of us makes to the data files causes the others' code to break.

Here is a good rule of thumb: If you are trying to solve a problem, and there are multi-billion dollar firms whose entire business model depends on solving the same problem, and there are whole courses at your university devoted to how to solve that problem, you might want to figure out what the experts do and see if you can't learn something from it.

Reglas Gentzkow y Shapiro I

1. Automate everything that can be automated.
2. Write a single script that executes all code from beginning to end.
3. Store code and data under version control.
4. Run the whole directory before checking it back in.
5. Separate directories by function.
6. Separate files into inputs and outputs.
7. Make directories portable.
8. Store cleaned data in tables with unique, non-missing keys.
9. Keep data normalized as far into your code pipeline as you can.

Reglas Gentzkow y Shapiro II

10. Abstract to eliminate redundancy. Abstract to improve clarity. Otherwise, don't abstract.
11. Don't write documentation you will not maintain. Code should be self-documenting.
12. Manage tasks with a task management system. E-mail is not a task management system.

Github

Mecanismos de control de versión

Github.com

- Es una herramienta para tomar “fotografías” del proyecto en los momentos que queramos
- Registra los cambios hechos entre una fotografía y otra, quién los hizo y cuándo
- Permite recuperar el proyecto a su estado en el momento de cualquier fotografía
- Permite tener distintas ramas del mismo proyecto

Existen muchos otros sistemas de control de versión, este es sólo uno.

Demostración en clases