How to improve your relationship with your future self*

Cómo mejorar su relación con su futuro yo

JAKE BOWERS
Universidad de Illinois

MAARTEN VOORS
Wageningen University

ABSTRACT

This essay provides practical advice about how to do transparent and reproducible data analysis and writing. We note that doing research in this way today will not only improve the cumulation of knowledge within a discipline, but it will also improve the life of the researcher tomorrow. We organize the argument around a series of homilies that lead to concrete actions. (1) Data analysis is computer programming. (2) No data analyst is an island for long. (3) The territory of data analysis requires maps. (4) Version control prevents clobbering, reconciles history, and helps organize work. (5) Testing minimizes error. (6) Work *can* be reproducible. (7) Research ought to be credible communication.

Inspirational Catchphrase or a short sermon / discourse on a moral theme.

Key words: research transparency, reproducible research, workflow, methodology

RESUMEN

*Este ensayo ofrece consejos prácticos sobre cómo efectuar análisis de datos y escritura científica de forma transparente y reproducible. Argumentamos que organizar la investigación de esta manera en tiempo presente no sólo mejorará la acumulación de conocimientos dentro de una disciplina, sino que también mejorará la vida académica futura del propio investigador. El argumento está organizado en torno a una serie de lecciones que conducen a acciones concretas. (1) El análisis de datos es programación computacional. (2) Ningún analista de datos es una isla por mucho tiempo. (3) El territorio del análisis de datos requiere del uso de mapas. (4) El control de versiones evita la superposición de versiones, la reconciliación del historial y favorece la organización del trabajo. (5) La prueba minimiza el error. (6) El trabajo * puede* ser reproducible. (7) La investigación debe ser una comunicación creíble.*

Palabras clave: *transparencia en la investigación, investigación reproducible, flujo de trabajo, metodología*

[1] Many thanks to the EGAP Learning Days 2016 participants in Santiago de Chile, to the BITSS team, the Department of Economics and Ted Miguel at UC Berkeley, where Maarten was a visiting researcher during spring 2016. Maarten gratefully acknowledges financial support from N.W.O. grant 451-14-001. A previous version of this paper benefited from comments and discussions with Mark Fredrickson, Brian Gaines, Kieran Healy, Kevin Quinn, Cara Wong, Mika LaVaque-Manty and Ben Hansen. The source code for this document may be freely downloaded and modified from https://github.com/jwbowers/workflow. This paper extends a previous version by Jake Bowers (2011b). While most of the text remains the same we have expanded and updated the essay to reflect current developments in technology and thinking about transparency and reproducibility of research.

"If you tell the truth, you don't have to remember anything."
(Twain 1975)

Memory is tricky. Learning requires effort. When we do not practice and repeat something that we want to remember, most people forget quickly, are overconfident in their abilities to recall future information (Koriat and Bjork 2005), and may even recall events that never happened.[1] Moreover, most of us live busy lives. We type text knowing that the laundry needs doing, hearing children play or cry, ignoring the news or the latest journal review, worrying about a friend. Our lives and minds are full and we need to efficiently move from task to task. If we cannot count on memory, then how can we do science?

How long does it take from planning a study to publication and then to the first reproduction of it? Is three years too short? Is ten years too long? We suspect that few of our colleagues in the social and behavioral sciences conceive of, field, write and publish a data driven study faster than about three years. We also suspect that, if some other scholar decides to re-examine the analyses of a published study, it will occur after publication. Moreover, this new scholarly activity of learning from one another's data and analyses can occur at any time, many years past the initial publication of the article.[2]

If we cannot count on our memories about why we made such and such a design or analysis decision, then what should we do? How can we minimize regret with our past decisions? How can we improve our relationship with our future self? This essay is a heavily revised and updated version of Bowers (2011b) and provides some suggestions for practices that will make reproducible data analysis easy and quick. Specifically, this piece aims to amplify some of what we already ought to know and do, and highlight some current practices, platforms and possibilities.[3] We aim to provide practical advice about how to *do work* such that one complies with such recommendations as a matter of course and, in so doing, can focus personal regret on bad past decisions that do not have to do with data analysis and the production of scholarly papers.

[1] See the following site for a nice overview of what we know about memory—including the fact that learning requires practice: http://www.spring.org.uk/2012/10/how-memory-works- 10-things-most-people-get-wrong.php On false memory see Wikipedia and linked studies https://en.wikipedia.org/wiki/False_memory

[2] The process of reproducing past findings can occur when one researcher wants to build on the work of another. It can also occur within the context of classes—some professors assign reproduction tasks to students to aid learning about data analysis and statistics. In addition to those models, reproduction of research has recently been organized to enhance the quality of public policy in the field of economic development by the 3IE Replication Program (Brown, Cameron, and Wood 2014) and to assess the quality of scientific research within social psychology (Open Science Collaboration 2015 and others) and within experimental economics (Camerer et al. 2016). In another study, 29 research teams recently collaborated on a project focusing on applied statistics to see if the same answers would emerge from re-analyses of the same data set (Silberzahn and Uhlmann 2015). They didn't.

Good example for DS or RSS talk to motivate open + reproducible work.

[3] King (1995) and Nagler (1995) were two of the first pieces introducing these kinds of ideas to political scientists. Now, the efforts to encourage transparency and ease of learning from the data and analyses of others have become institutionalized with the DA-RT initiative (http://www.dartstatement.org/; see also Lupia and Elman 2014). These ideas are spreading beyond political science as well (see Freese 2007; Asendorpf et al. 2013; see also http://osf.io and http://www.bitss.org/).

We organize the paper around a series of homilies that lead to certain concrete actions:

- Data analysis is computer programming.
- No data analyst is an island for long.
- The territory of data analysis requires maps.
- Version control prevents clobbering, reconciles history, and helps organize work.
- Testing minimizes error.
- Work can be reproducible.
- Research ought to be credible communication.

I. DATA ANALYSIS IS COMPUTER PROGRAMMING

All your results (numbers, comparisons, tables, plots, figures) should be produced from code, not from a series of mouse clicks or copying and pasting.[4] Imagine you wanted to re-create a figure and include a new variable, you should be able to do so with just a few edits to the code rather than knowledge of how you used a pointing device in your graphical user interface all those years ago.

Let's look at an example. Using an open-source statistics programming language called R (R Development Core Team 2016), you might specify that a file, called `fig1.pdf` is produced by the following set of commands in a file called `makefig1.R`. Let's look at some annotated R code:

```
# This file produces a plot relating the explanatory variable to
the outcome.
## Read the data
thedata <- read.csv("Data/thedata-15-03-2011.csv")
## begin writing to the pdf file
please-open-pdf("fig1.pdf")
please-plot(outcome by explanatory using thedata. red lines
please.)
please-add-a-line(using model1)
## Note to self: a quadratic term does not add to the substance
## model2 <- please-fit(outcome by explanatory+explanatory^2
using thedata
## summary(abs(fitted(model1)-fitted(model2)))
```

[4] To our future selves: both of us are using a computer control device called a track-pad, and haven't used an older device called a mouse in some years. Our current track-pads do not click. We are not sure why we talked about mouse clicks just now.

```
## stop writing to the pdf file
please-close-pdf()
```

Now, in the future if you wonder how "that plot on page 10" was created, you will know: (1) "that plot" is from a file called `fig1.pdf` and (2) `fig1.pdf` was created in `makefig1.R`. Any changes to the file in the future will just require some quick edits of commands already written (provided R still exists).[5] Even if in the future R ceases to exist, you (or someone else) will at least be able to read the plain text commands and use them to write code in a new favorite statistical computing language: R scripts are written in *plain text*, and plain text is a format that will be around as long as computer programmers write computer programs.

Moreover, coding saves time. Often, typing commands into a file saves a lot of time, especially if projects grow in the number of files, collaborators, or complexity of analysis. Manually importing one data file into R may be effortless. Importing 100 files is another issue, and the time costs of each manual action add up quickly (and the probability for mistakes increases too).

Coding Scales better & reproduces better than GUIs.

Also, realize that *file names send messages* to co-authors and to your future self. This means that if you name your files with evocative and descriptive names, your collaborators are less likely to call you at midnight asking for help and you will remove some regret from your future self and protect your friendships and working relationships. For example, if you are studying inequality and protest, you might try naming a file something like `inequality-and-protest-figures.R` instead of `temp9.R` or `supercalifragilisticexpialidocious.R`. By the way, the extension `.R` tells us and the operating system that the file contains R commands. This part of the filename enables us to quickly search our antique hard drives for files containing R scripts.[6]

Human - friendly file names

Coding helps us to avoid making mistakes. For example, in our example above we may be interested in how many people protest. We may use a data file containing all protests for several years. People often create a tabular display of this data by copying the results manually to the working paper document. In copying we can make mistakes. So a better approach is to automatically create the table (in the format you like, with horizontal lines, 3 decimals, a title, etc.) and save this table in whatever file type you need (.tex. .pdf, .rtf, etc.). Now when we obtain new data, a new table can be created quickly, so we make fewer mistakes and save time!

Scripting reduces the opportunity for mistakes & makes them faster to correct.

For example, the following table was created entirely with code using R and the xtable package Dahl (2016).[7] We then read this table into the current file using

[5] We use data from Norris (2015) throughout this paper.

[6] We think that some method of tagging files by the purpose of the file will continue help analysts find and organize their files for long after the idea of a computer mouse ceases to make sense.

[7] @beck2010reg inspired this particular presentation of a linear model.

the LaTeX command `\input{protesttable.tex}` which produces a nice looking table in pdf format.

```
#Run the regression
lm1 <- lm(protest05 ~ gini04 + meanpr, data = good.df)
#make the table file
makebecktable(lmobj = lm1, vars = c("Intercept", "Income
Inequality (lower = more equal)", "Mean Political Rights (lower
= more rights)"),
thecaption = "People living in countries with unequal income
distributions report more protest activity to World Values
Survey interviewers than people living in countries with
relatively more equal income distributions, adjusting for
average political rights as measured by Freedom House 1980--
2010.",
thelabel = "tab:protest", filename = "protesttable.tex")
```

	Coef	Std. Err.	95% CI	
Intercept	43.3	11.5	20.0	66.7
Income Inequality (lower = more equal)	43.0	28.9	–15.6	101.5
Mean Political Rights (lower = more rights)	–8.5	1.7	–11.9	–5.2
		n: 41, resid.sd: 18, R^2: 0.41		

Table 1: People living in countries with unequal income distributions report more protest activity to World Values Survey interviewers than people living in countries with relatively more equal income distributions, adjusting for average political rights as measured by Freedom House 1980–2010.

Notice that we might have made one file called `makefig1.R` and another called `protestincomedisttab.R` which in turn produce a pdf file (`fig1.pdf`) and a LaTeX format table (`protesttable.tex`). This idea of modularity (Nagler 1995) in code enables us to quickly find errors, and, if we want to make changes to improve our work, only one file may be changed at a time. Further, it enhances collaboration—Jake can work to make the figure and Maarten can work to make the table without worry about conflicting files.

Split projects into modular tasks, each gets its own script. This allows parallel working without conflicts.

STEP 1 Code everything that can be coded. If we know the provenance of results, future or current collaborators make fewer mistakes and can quickly and easily reproduce (and thus change and improve) upon the work.

II. NO DATA ANALYST IS AN ISLAND FOR LONG

Data analysis involves a long series of decisions. Each decision requires a justification. Some decisions will be too small and technical for inclusion

in the published article itself. Still, these need to be documented in the code itself (Nagler 1995). Paragraphs and citations in the publication will justify the most important decisions but the code itself documents the smaller but still consequential decisions. So, one must code to communicate with yourself and others. There are two main ways to avoid forgetting the reasons you did something with data: comment your code and tightly link your code with your writing—making your code literate.

Data Sciena is just a series of decisions. ↑ Document your options and your choices.

Code is communication: Comment code

Comments, unexecuted text inside of a script, are a message to collaborators (including your future self) and other consumers of your work. In the above code chunk, we used comments to explain the lines to readers unfamiliar with R and to remember that we had tried a different specification but decided not to use it because adding the squared term did not really change the substantive story arising from the model.[8] Messages left for your future self (or near-future others) help retrace and justify your decisions as the work moves from seminar paper to conference paper to poster back to paper to dissertation and onwards maybe even to publication.

Notice one other benefit of coding for an audience: we learn by teaching. By assuming that others will look at your code, you will be more likely to write clearer code, or perhaps even to think more deeply about what you are doing as you do it since you are explaining even as you write.

Comment liberally. Comments are discarded when R or STATA runs analysis, so only those who dig into the source code of your work will see them.

Writing your code for other people holds you accountable for writing clearly. ↑

Code to communicate: Literate programming.

Steal quote for slide.

> *"Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do."*
> (Knuth 1984, 97)

Imagine you discover something new (or confirm something old). You produce a nice little report on your work for use in discussions of your working group or as a memo for a website or reviewer appendix. The report itself is a pdf or html file or some other format which displays page images to ease reading rather than to encourage reanalysis and rewriting. Eventually pieces of that report (tables, graphs, paragraphs) ought to show up in, or at least inform, the publishable paper. Re-creating those analyses by pointing, clicking, copying, or

[8] R considers text marked with # as a comment. For Stata simply add a * before the text.

pasting would invite typing error and waste time. Re-creating your arguments justifying your analysis decisions would also waste time.

More importantly, we and others want to know why we did what we did. Such explanations may not be very clear if we have some pages of printed code in one hand and a manuscript in the other. Keep in mind the distinction between the "source code" of a document (i.e. what computation was required to produce it) and the visible, type-set page image. Page images are great for reading, but not great for reproducing or collaborating. The source code of any document exchanged by the group must be available and executable.[9]

How might one avoid these problems? *Literate programming* is the practice of weaving code into a document such that paragraphs, equations, and diagrams can explain the code, and the code can produce numbers, figures, and tables (and diagrams and even equations and paragraphs). Literate programming is not merely fancy commenting but is about enabling the practice of programming itself to facilitate easy reproduction and communication.

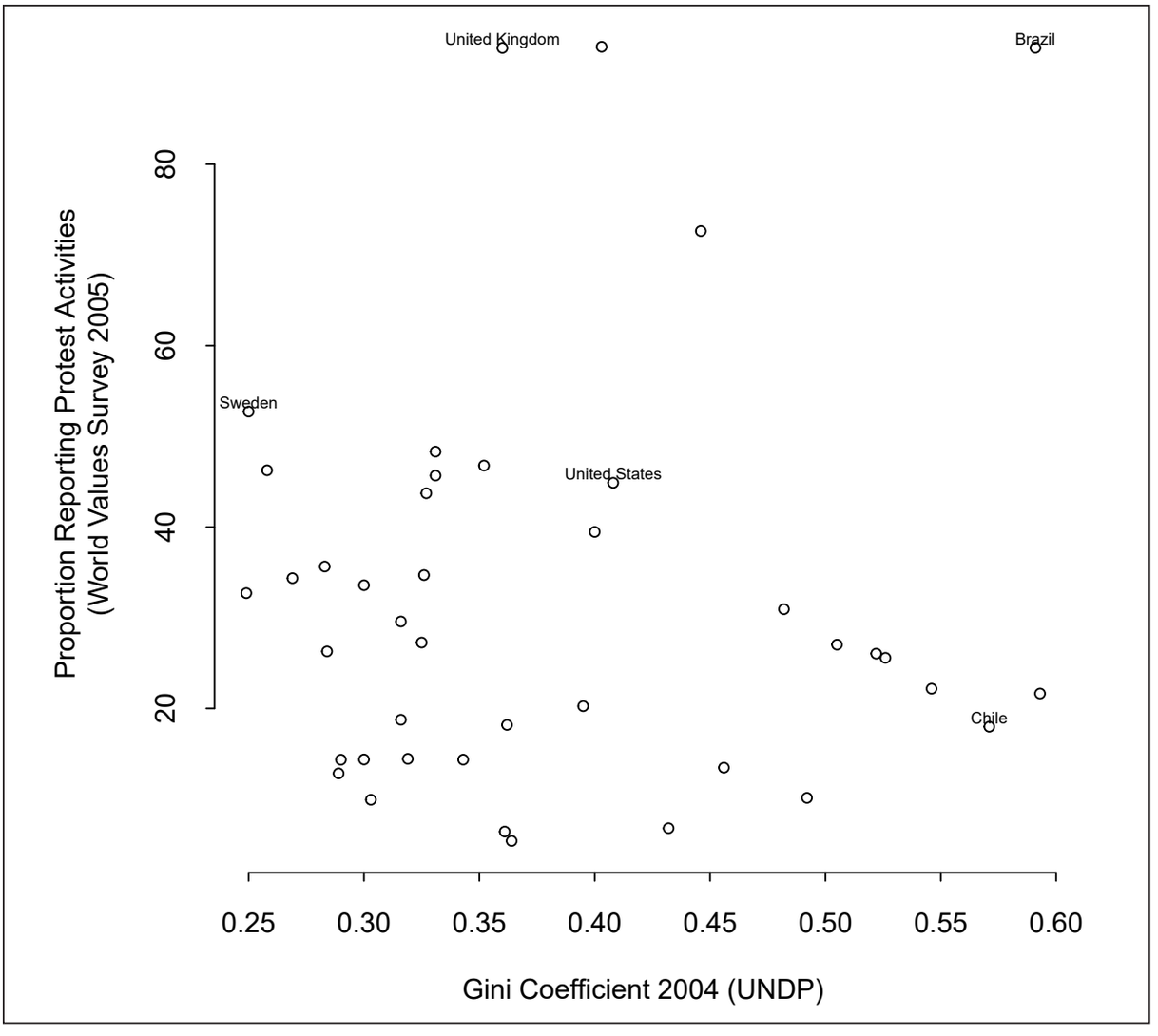
For example, in § Section 1, we suggested that we knew where "that plot on page 10" comes from by making sure we had a `fig1.pdf` file produced from a clearly commented plain text file called something like `makefig1.R`. An even easier solution would be to directly include a chunk of code to produce the figure inside of the paper itself. This paper, for example, was written in plain text using markdown markup with R code chunks to make things like Figure 1.[10] Here is an example of embedding a plot inside of a document along with the source of this paragraph. In a document sent to a journal one would tend to hide the code chunks and thus one would only see the plot.

```
## Make a scatterplot of Protest by Inequality
par(bty = "n", xpd = TRUE, pty = "s", tcl = -0.25)
with(good.df, plot(gini04, protest05, xlab = "Gini Coefficient
2004 (UNDP)", ylab = "Proportion Reporting Protest Activities\
n(World Values Survey 2005)", cex = 0.8))
## Label a few interesting points
with(good.df[c("Brazil", "United Kingdom", "United States",
"Sweden","Chile"), ], text(gini04, protest05, labels = Nation,
srt = 0, cex = 0.6, pos = 3, offset = 0.1))
```

[9] As of the 2016 version of this paper, this idea is now widespread and made much easier than before via online services for code sharing and collaboration such as GitHub, Open Science Framework and BitBucket. As more data analysis moves online, it will become easier for cross-platform and geographically distant collaboration to occur. For just one set of examples, see Docker or other services that make cloud computing easier and more accessible.

[10] This combination of Markdown and R is called R Markdown.

Figure 1: Average number of protest activities by income inequality across countries in 2004–2005.



Markdown (and LaTeX and HTML) all have ways to cross-reference within a document. For example, by using `\label{fig:giniprot}` in LaTeX or `{#fig:giniprot}` in the *pandoc* version of markdown built into RStudio, we do not need to keep track of the figure number, nor do extra work when we reorganize the document in response to reviewer suggestions. Nor do we need a separate `makefig1.R` file or `fig1.pdf` file. Tables and other numerical results are also possible to generate within the source code of a scholarly paper. For example, if we had omitted the filename in `makebecktable` above, the LaTeX formatted table would have appeared within this document itself. For Stata users, there now is Markdoc, which is very similar to R markdown.

The R project has a task view devoted to reproducible research listing many of the different approaches to literate programming for R. If your workflow does not involve R, you can still implement some of the principles here. Imagine creating a style in Microsoft Word called "code" which hides your code when you print your document, but which allows you to at least run each code chunk piece by piece (or perhaps there are ways to extract all text of style "code" from

a Microsoft Word document for use in some other program). Or one could just use some other kind of indication linking paragraphs to specific places in code files. There are many ways that creative people can program in a literate way.

Literate programming need not go against the principle of modular data analysis (Nagler 1995). Jake routinely uses several different files that fulfill different functions, some of them create LaTeX code that he can `\input` into his `main.tex` file, others setup the data, run simulations, or allow him to record his journeys down blind alleys. Of course, when we have flying cars running on autopilot, perhaps something other than combining R and Markdown or LaTeX will make our lives even easier. Then we'll change.

Slightly patronising description of literate programming.

Misses drawbacks wrt scaling and automation

STEP 2 We analyze data in order to explain something about the world to other scholars and policy makers. If we focus on explaining how we got our computers to do data analysis, we will do a better job with the data analysis itself: we will learn as we focus on teaching others about why we did what we did, and we will avoid errors and save time as we ensure that others (including our future selves) can retrace our steps. A paper than can be "run" to reproduce all of analyses also instills confidence in readers and can more effectively spur discussion and learning and cumulation of research.

III. THE TERRITORY OF A DATA ANALYSIS REQUIRES MAPS

Data analysis tends to involve multiple people working with multiple files. Future collaborators will need a map to understand the flow of inputs (like raw data) and outputs (like figures, tables, and individual numbers).

Meaningful code requires data

All files containing commands operating on data must refer to a data file. A reference to a data file is a line of code that the analysis program will use to operate on `("load" / "open" / "get"/ "use")` the data file. One should not have to edit this line on different computers or platforms in order to execute this command. Using R, for example, all analysis files should have `load("thedata.rda")` or `read.csv("thedata.csv")` or some equivalent line in them, and `thedata.csv` should be stored in some easy to find place (like in the same directory as the file or perhaps in "`Data/thedata.rda`"). Of course, its even better to include a comment pointing to the data file in addition to the line loading the file itself. This means one should not see lines like `setwd(C:\JakesFiles\theproject)` or `setwd(/Users/maartenvoors/theproject)` at the beginning of documents that are meant to be shared with a future self or present or future others.

Write portable file paths from root directory of project.

File organization can be a map itself

Where should one store data files? An obvious solution is to make sure that the data file used by a command file is in the same directory as the command file. More elegant solutions require all co-authors to have the same directory structure so that `load("Data/thedata.rda")` means the same thing on all computers used to work on the project. This kind of solution is one of the things that Dropbox and more formal version control systems do well (as discussed a bit more below in § Section 4).

The principle of modularity suggests that you separate data cleaning, processing, recoding and merging from analysis in different files (Nagler 1995). So, perhaps your analysis files will load `("cleandata.rda")` and a comment in the code will alert the future you (among others) that `cleandata.rda` was created from `create-cleandata.R` which in turn begins with `read.csv(\ url("http://data.gov/dirtydata.csv"))`. Such a data processing file will typically end with something like `save("cleandata.rda")` so that we are doubly certain about the provenance of the data.[11]

Now, if in the future we wonder where `cleandata.rda` came from, we might search for occurrences of `cleandata` in the files on our system. Of course, if it is difficult to find the right version of cleandata on the system, it might help to know where files like cleandata tend to be: that is, to have a system for project file organization. Best practice is to create a file system where you separate folders by function and separate input from output files. For example, below we show a folder structure for a paper where we look at inequality. The paper is written in .tex, data analysis in both Stata (that Maarten prefers, so we see .do and .dta file extensions) and in R (the program of choice for Jake, see the .R extensions). Maarten likes to make sure that the directory structure in his projects is the same across projects (so, for example, the numbers on the directory names allow for easy default sorting). The file naming with full dates also allows for easy sorting:

```
00_archive/
01_paper/
  20160622_inequality.tex
02_data/
  00_archive/
  01_rawdata/
  02_cleandata/
  20160622_analysis.dta
  20160622_analysis.rda
```

[11] Of course, if you need math or paragraphs to explain what is happening in these files, you might prefer to make them into R+Markdown or R+LaTeX files, for which the conventional extension is .Rmd or .Rnw respectively. So you'd have create-cleandata.Rmd written as a mixture of Markdown and R which might explain and explore the different coding decisions you made, perhaps involving some diagnostic plots.

```
03_analysis/
  00_archive/
  01_temp/
  02_output/
    01_tables/
    02_figures/
      20160622_models.do
      20160612_simulations.R
      20160502_prep_data.do
```

There are a couple of things to note: See that files start with a number to give them an ordering. Note also we create an `00_archive folder`, here you can store older files without having to delete them permanently and can reduce the number of files in your main folders. Also there is a clear separation between raw data (data that came straight from the field, downloaded data, someone else's replication files, etc.) and clean data (data ready to use). Note that in the analysis folder we created two files, one for cleaning and merging data called `20160622_prep_data` and another for running the analysis called `20160622_models`. See also the placeholder output folders, with subfolders for figures and tables.

directory structure & file names ↑ chosen for human readability

The input-output map can also be in the form of files

Another solution to the problem of finding files and knowing how files relate is to maintain a file for each project called `MANIFEST.txt` or `INDEX.txt` or `README.txt`, which lists the data and command files with brief descriptions of their functions and relations. However, this file may be a burden to maintain.

Jake tends to be less organized than Maarten and relies on `README.md` files to tell his future self what is going on in a given directory, a version control system to keep track of versions of files (so that he doesn't need to add dates to filenames), and a Makefile to keep track of relationships between files. Notice that the analysis subdirectory has its own `README.md file.`

```
manuscript\
  paper.pdf
  paper.Rmd
figures\
  boxplots.R
  boxplots.pdf
tables\
analysis\
README.md
models.R
data\
  workingdata.R
```

```
  rawdata.csv
  workingdata.rda
build\
libraries\
README.md
Makefile
```

Imagine that the `paper.Rmd` shows the `boxplots.pdf` file and also reports on the total size of the data. So, the introduction to the paper requires the `workingdata.rda` file (to know the sample size) and `boxplots.pdf`. In turn, those files depend on `workingdata.R` which takes `rawdata.csv` and produces the `workingdata.rda` file, and `boxplots.R` which uses `workingdata.rda` to make the plots. A Makefile records this web of relationships in a structured way such that one can say make `manuscript/paper.pdf` and all of the files that are required to produce `paper.pdf` will be executed if they have not recently been used or if they have recently been changed.[12] Here is an example that records the relationships just described:

```
manuscript/paper.pdf: manuscript/paper.Rmd figures/boxplot.pdf
    cd manuscript && Rscript -e "library(rmarkdown);
render('paper.Rmd')"

figures/boxplot.pdf: figures/boxplot.R data/workingdata.rda
    R CMD BATCH figures/boxplot.R

data/workingdata.rda: rawdata.csv workingdata.R
    R CMD BATCH data/workingdata.R
```

To produce `paper.pdf` having only downloaded `rawdata.csv` and the command files, one would type make `paper.pdf` and the GNU make system would first run `R CMD BATCH data/workingdata.R` then, if that is successful, it would run `R CMD BATCH figures/boxplot.R` and finally run the line required for creation of `paper.pdf`. Later, if only `paper.Rmd` is edited, make `paper.pdf` would only run the line for `paper.pdf` because it would know that `boxplot.pdf` and `workingdata.rda` are relatively recent and the `.csv` and `.R` files required for them have not changed.

Document + automate with README and make files.

STEP 3 We should know where the data came from and what operations were performed on which set of data. The authors of this paper have two different versions of this workflow. Both fulfill their purpose as a map to the territory of a data analysis.

[12] See http://kbroman.org/minimal_make/ for more about Makefiles.

IV. VERSION CONTROL PREVENTS CLOBBERING, RECONCILES HISTORY, AND HELPS ORGANIZE WORK

Group work requires knowing which versions of files are new, which are old, and what changed in between. Many people are familiar with the "track changes" feature in modern WYSIWYG (what you see is what you get) word processors or the fact that Dropbox or Google Docs allow one to recover previous versions of files. These are both kinds of version control. More generally, when we collaborate, we'd like to do a variety of actions with our shared files. Collaboration on data analytic projects is more productive and better when: (1) it is easy to see what has changed between versions of files; (2) members of the team feel free to experiment and then to dump parts of the experimentation in favor of previous work while merging the successful parts into the main body of the paper; (3) the team can produce "releases" of the same document (one to MPSA, one to APSR, one to their parents) without spawning many possibly conflicting copies of the same document; and (4) people can work on the same files at the same time without conflicting with one another, and can reconcile their changes without too much confusion and clobbering. Clobbering is what happens when your future self or your current collaborator saves an old version of a file over a new version, erasing good work by accident (or creating a conflicted copy resulting in an arduous back-and-forth between versions).

Good, concise description of why we want Version Control. ↑

Using Dropbox, "track changes" in Word or Google Docs is one way to manage your collaborative efforts. They are popular tools, but have some downsides as they require you to communicate with other folks in your group before you can edit existing files. On Dropbox, you cannot edit and save a given file at the same time. Tracking changes with Word has similar limitations (merging the changes between two documents is not simple or built-in). Google Docs allows for two people to edit the same document at the same time, but you must be online with a fairly fast internet connection.[13] Communication and trading-off while editing a file prevents your work (or your colleagues' work) from getting lost when you both try to save the same file on top of each other. One may also use Dropbox as a kind of server for version control. See the example above. This system is fine until your collaborations or project grows large (in terms of MBs) and you may run out of storage space on your computer or you may need to upgrade your Dropbox's storage space. You will, however, need to agree on an iron rule for file and directory naming with your collaborators to ensure that the files and directories are always named the same across machines and versions, plus a clear communication plan for trading-off.

Make table of these properties vs software.

An excellent, simple, and robust version control system is to rename your files with the date and time of saving them: `yyyymmdd_project.docx` (for changes within the same day, just before a deadline for example, you may even add a time, so it becomes `20160623_4PM_inequalitypaper.docx`).

[13] Note to future selves: change this section every five years or so. Probably stop saying 'internet' in 2026.

This communicates version and prevents clobbering unless the old file has the same name or you don't update the new file name. Remember the days when you received a file called `inequalityMV4_APSRsubmit_reallyfinal2.tex`? That should quickly become something of the past. We find ourselves preaching this gospel to our collaborators frequently and will keep renaming files until a collaborator has been successfully pacified. Be sure to include year in the file names—remember, the life of an idea is measured in years. If you are wise enough to have saved your documents as plain text then you can easily compare versions of the same document using the many utilities available for comparing text files.[14] When you reach certain milestones you can rename the file accordingly: `thedocAPSA2009.tex`—for the one sent to discussants at APSA—or `thedocAPSR2015.tex`—for the version eventually sent to the APSR six years after you presented it at APSA. The formal version control systems mentioned above all allow this kind of thing and are much more elegant and capable, but you can do it by hand as long as you don't mind taking up disk space and having many "thedoc..." files around. Indeed, the number of files can add up quickly (especially around submission time!) and you may want to create an "0_Archive" folder to store older versions. If you do version control by hand, spend a little extra time to ensure that you do not clobber files when you make mistakes typing in the file-names. And, if you find yourself spending extra time reconciling changes made by different collaborators by hand, remember this is a task that modern version control systems take care of quickly and easily.

We feel the best practice is to use formal version control approaches to organizing work version control. As of 2016, the standard for managing large collaborations with many files is the Git system. User friendly free front-ends for Git currently make the process of learning and using Git very convenient and integrated into the kind of workflow that your future self will be proud of: GitHub and BitBucket make it easier to use Git, and the Open Science Framework integrates with GitHub to further ease this task. Jake uses github for all of his collaborations with others as well as with his future self: his collaborators appreciate some of the nice extra features of GitHub, such as the ability to keep a shared task list. Learning about version control systems takes a bit of time. We suggest, however, it is well worth the time investment as it will save lots of time later on.

Step 4 Writing is rewriting. Thus, all writing involves versions. When we collaborate with ourselves and others we want to avoid clobbering and we want to enable graceful reconciliation of rewriting. One can do these things with formal systems of software (like Git) or with formal systems of file naming, file comparing and communication—or, even better, with both. In either case, plain

[14] Adobe Acrobat allows one to compare differences in pdf files. OpenOffice supports a "Compare Documents" option. Word now does the same. And Google Docs will report on the version history of a document. On a Mac, the FileMerge utility works for plain text files.

text files will make such tasks easier, take up less disk space and be easier to read for the future you.

V. TESTING MINIMIZES ERROR

Anyone writing custom code should worry about getting it right. The more code one writes, the more time one has to appreciate problems arising from bugs, errors, and typos in data analysis and code. One thing we can do to minimize and catch the inevitable mistakes is to include testing in our coding.

This idea is not new. The desire to avoid error looms large when large groups of programmers write code for multi-million dollar programs. The idea of test driven development and the idea that one ought to create tests of small parts of one's code arose to address such concerns.[15]

For the social scientist collaborating with her future self and/or a small group of collaborators, here is an example of this idea in a very simple form: Say you want to write a function to multiply a number by 2. If the function works, when you give it the number 4, you should see it return the number 8 and when you give it -4, you should get -8.

```
##  The test function:
test.times.2.fn <- function(){
    ##  This function tests times.2.fn
    if (times.2.fn(thenumber = 4) == 8 &
        times.2.fn(thenumber = -4) == -8) {
        print("It works!")
        } else { print("It does not work!")
}
}
##  The actual function is:
times.2.fn <- function(thenumber){
    ##  This function multiplies a scalar number by 2
    ##  thenumber is a scalar number
    thenumber+2
}
```

Here we use the test function to make sure it works.

```
test.times.2.fn()

[1] "It does not work!"
```

[15] There are now R packages to help R package writers do this, see for example https://github.com/hadley/testthat and the article on it https://journal.r-project.org/archive/2011- 1/RJournal_2011- 1_Wickham.pdf

Ack! we mistyped + instead of *. Good thing we wrote the test![16]

That approach works well when one is paying close attention to messages within the code. Another approach that we like better is to make the code stop with an error whenever it doesn't pass a test. In R we use the stopifnot function for this. For example, say we wanted to re-run our analyses excluding the countries with extreme values of protest. The following code makes a new dataset excluding places where more than 90% of the respondents to the World Values survey reported some protest activity. If all of the values of `protest05` are not less than 90, the code will stop with an error.

```
smalldat <- subset(good.df, subset = protest05 < 90)
stopifnot(all(smalldat$protest05 < 90))
```

STEP 5 No one can foresee all of the ways that a computer program can fail. One can, however, at least make sure that it succeeds in doing the task of motivating the writing of the code in the first place or produces an error in the right place and the right time.

Write unit tests ↑

VI. WORK CAN BE REPRODUCIBLE

Over the past decades more and more attention is being paid to "reproducible research", and creating a "reproducible workflow". As always, the devil is in the details: Here we list a few of our own attempts at enabling reproducible research. You'll find many other inspiring examples on the web. Luckily, the open source ethos aligns nicely with academic incentives, so we are beginning to find more and more people offering their files and ideas about workflow online for copying and improvement. By the way, if you do copy and improve, it is polite to alert the person from whom you made the copy and to credit them in some way if not also to offer your ideas for improvement.[17]

We have experimented with several systems so far: (1) We wrote this paper in the R Markdown literate programming format using two different text editors (Jake used vim) and (Maarten used atom). We organized our files and collaboration on GitHub. The source code for this document can be accessed, downloaded and modified from https://github.com/jwbowers/workflow. We also used GitHub Issues to send notes to each other and maintain a task list. The nice thing about GitHub is that is enables you to make "releases" of a project, which enable you to smooth the reproduction of all of the products of your

Nice! Suggest as DS repo to investigate.

[16] A more common example of this kind of testing occurs everyday when we recode variables into new forms but look at a crosstab of the old versus new variable before proceeding.

[17] For a formal example of copying and improving code, see the GitHub-based fork and pull-request workflow https://gist.github.com/Chaser324/ce0505fbed06b947d962

research (simulations, data analyses, tables, figures, etc.). Similar features are offered by the Open Science Framework (OSF). Maarten recently participated in a workshop on research transparency organized by the Berkeley Institute for Transparency in the Social Sciences in which all course material was accessible through OSF. OSF integrates with GitHub nicely and a key benefit is that both systems have been created to remain for a long time (unlike perhaps your university drive). (2) For one paper Jake simply mixed R and LaTeX code in one document (called the "Sweave" format of literate programming) and then added that document and data into a compressed archive (Bowers and Drake 2005); (3) For another, more computing intensive paper, Jake's collaborator, Mark Fredrickson, assembled a set of files that enabled reproduction of results using the make system (Bowers, Hansen, and Fredrickson 2008); (4) Jake also tried the "compendium" approach (Gentleman 2005; Gentleman and Temple Lang 2007), which embeds an academic paper within the R package system (Bowers 2011a); The benefit of the compendium approach is that one is not required to have access to a command line for make: the compendium is downloadable from within R using `install.packages()` and is viewable using the `vignette()` function in any operating system than runs R.[18] The idea that one ought to be able to install and run and use an academic paper just as one installs and uses statistical software packages is very attractive, and we anticipate that it will become ever easier to turn papers into R packages as creative and energetic folks turn their attention to the question of reproducible research.[19] Finally, (5) Maarten has been using Dropbox for most of his collaborative projects. See comments above. As a tool for making files publicly available, Dropbox is less useful. You can, of course, make replication files available through creating an public folder, but you will still need to refer to this folder on a particular project website (an example is Maarten's project on conflict and football).

bit dated

There are some noteworthy new developments also. See, for example, a new web app, Jupyter Notebook (http://jupyter.org/), that enables you to make and share documents that include text, code, equations and visualizations in one literate programming environment. For researchers working across different platforms, there is Docker, which promises to enable all of the collaborators on one project to use the same computing environment even if they are using different laptops running different operating systems.

There are also are great notes programs that help you communicate with collaborators. Remember, while email is fantastic for fast communication, it is not a tool designed for project management. It may work for projects with few co-authors, but when the number of collaborators increase it becomes very difficult to keep track. Most online systems for task management have mobile

[18] Notice that Jake's reproduction archives and/or instructions for using them are hosted on the Dataverse, which is another system designed to enhance academic collaboration across time and space.

[19] See, for example, http://r-pkgs.had.co.nz/

apps, including, amongst others, Flow, Asana, Wrike, Basecamp, Simplenote, Evernote and for Windows users there is OneNote.

nice quote to motivate practice.

STEP 7 We all learn by doing. When we create a reproducible workflow and share reproduction materials we improve both cumulation of knowledge and our methods for doing social science (Freese 2007; King 1995).

VII. RESEARCH OUGHT TO BE CREDIBLE COMMUNICATION

> *[I]f the empirical basis for an article or book cannot be reproduced, of what use to the discipline are its conclusions? What purpose does an article like this serve?*
> (King 1995, 445)

We all always collaborate. Many of us collaborate with groups of people at one moment in time as we race against a deadline. All of us collaborate with ourselves over time. The time-frames over which collaboration is required—whether among a group of people working together or within a single scholar's productive life, or probably both—are much longer than any given version of any given software will easily exist. Plain text is the exception.

But what if no one ever hears of your work, or, by some cruel fate, your article does not spawn debate? Why then would you spend time to communicate with your future self and others? Our own answer to this question is that it is efficient and that out of principle we want our work to be credible and useful to ourselves and other scholars. What we report in our data analyses should have two main characteristics: (1) the findings of the work should not be a matter of opinion; and (2) other people should be able to reproduce the findings. That is, the work represents a shared experience—and an experience shared without respect to the identities of others (although requiring some common technical training and research resources).

Assume we want others to believe us when we say something. More narrowly, assume we want other people to believe us when we say something about data: "data" here can be words, numbers, musical notes, images, ideas, etc. The point is that we are making some claims about patterns in some collection of stuff. For example, when Jake was invited into the homes and offices of ordinary people in Chile in 1991, the stuff was recordings of long and semi-structured conversations about life during the first year of democracy. Now, it might be easy to convince others that "this collection of stuff" is different from "that collection of stuff" if those others were looking over our shoulders the whole time that we made decisions about collecting the stuff and broke it up into understandable parts and reorganized and summarized it. Unfortunately, we can't assume that people are willing to shadow a researcher throughout her career. Rather, we do our work alone or in small groups and want to convince other distant and future people to take our analyses and findings seriously.

Now, say your collections of stuff are large or complex and your chosen tools of analyses are computer programs. How can we convince people that what we did with some data with some code is credible, not a matter of whim or opinion, and reproducible by others who didn't shadow us as we wrote our papers? This essay has suggested a few concrete ways to enhance the believability of such scholarly work. In addition, these actions (as summarized in the section headings of this essay) make collaboration within research groups more effective. Believability comes in part from reproducibility and researchers often need to be able to reproduce in part or in whole what different people in the group have done or what they, themselves, did in the past.

In the end, following these practices and those recommended by Fredrickson, Testa, and Weidmann (2011) and Healy (2011) among others working on these topics allows your computerized analyses of your collections of stuff to be credible. If someone then quibbles with your analyses, your future self can shoot them the archive required to reproduce your work.[20] You can say, "Here is everything you need to reproduce my work." To be extra helpful you can add "Read the README file for further instructions." And then you can get on with enjoying your life full of friends, children, laundry, news, or even journal reviews.

REFERENCES

Asendorpf, Jens B., Mark Conner, Filip De Fruyt, Jan De Houwer, Jaap J. A. Denissen, Klaus Fiedler, Susann Fiedler, David C. Funder, Reinhold Kliegl, Brian A. Nosek, Marco Perugini, Brent W. Roberts, Manfred Schmitt, Marcel A. G. Van Aken, Hannelore Weber, Jelte M. Wicherts. 2013. "Recommendations for Increasing Replicability in Psychology". *European Journal of Personality* 27(2): 108-19.

Bowers, Jake. 2011a. "Reproduction Compendium for: 'Making Effects Manifest in Randomized Experiments'". Retrieved on 26 Sep 2008 from http://hdl.handle.net/1902.V15499.

Bowers, Jake. 2011b. "Six Steps to a Better Relationship with Your Future Self". *The Political Methodologist* 18(2): 2-8.

Bowers, Jake and Katherine W. Drake. 2005. "Reproduction Archive for: 'EDA for HLM:Visualization when Probabilistic Inference Fails'". Retrieved from http://hdl.handle.net/1902.V13376.

Bowers, Jake, Ben B. Hansen and Mark M. Fredrickson. 2008. "Reproduction Archive for: 'Attributing Effects to A Cluster Randomized Get-Out-The-Vote Campaign'". Retrieved on 26 Sep 2008 from http://hdl.handle.net/1902.1/12174.

Brown, Annette N., Drew B. Cameron and Benjamin DK Wood. 2014. "Quality Evidence for Policymaking: I'll Believe It When I See the Replication". *Journal of Development Effectiveness* 6(3): 215-35.

[20] Since you used plain text, the files will still be intelligible, analyzed using commented code so that folks can translate to whatever system succeeds R, or since you used R, you can include a copy of R and all of the R packages you used in your final analyses in the archive itself. You can even throw in a copy of whatever version of Linux you used and an open source virtual machine running the whole environment using, say, Docker.

Camerer, Colin F., Anna Dreber, Eskil Forsell, Teck-Hua Ho, Jürgen Huber, Magnus Johannesson, Michael Kirchler, Johan Almenberg, Adam Altmejd, Taizan Chan, Emma Heikensten, Felix Holzmeister, Taisuke Imai, Siri Isaksson, Gideon Nave, Thomas Pfeiffer, Michael Razen and Hang Wu. 2016. "Evaluating Replicability of Laboratory Experiments in Economics". *Science* 351(6280): 1433-1436.

Dahl, David B. 2016. "Xtable: Export Tables to Latex or Html". Retrieved from https://cran.r-project.org/web/packages/xtable.

Fredrickson, Mark M., Paul F. Testa, and Nils B. Weidmann. 2011. "Collaboration for Social Scientists, or Software Is the Easy Part". *The Political Methodologist,* 18(2): 19-23.

Freese, Jeremy. 2007. "Replication Standards for Quantitative Social Science Why Not Sociology?". *Sociological Methods & Research* 36(2): 153-72.

Gentleman, Robert. 2005. "Reproducible Research: A Bioinformatics Case Study". *Statistical Applications in Genetics and Molecular Biology* 4(1): 1-23.

Gentleman, Robert and Duncan Temple Lang. 2007. "Statistical Analyses and Reproducible Research". *Journal of Computational and Graphical Statistics* 16(1): 1-23.

Healy, Kieran. 2011. "Choosing Your Workflow Applications". *The Political Methodologist* 18(2): 9-18

King, Gary. 1995. "Replication, Replication". *PS: Political Science and Politics* 28(3): 444-452.

Knuth, Donald E. 1984. "Literate Programming". *The Computer Journal* 27(2): 97-111.

Koriat, Asher and Robert A. Bjork. 2005. "Illusions of Competence in Monitoring One's Knowledge During Study". *Journal of Experimental Psychology: Learning, Memory, and Cognition* 31(2): 187-194.

Lupia, Arthur and Colin Elman. 2014. "Openness in Political Science: Data Access and Research Transparency". *PS: Political Science & Politics* 47(1): 19-42.

Nagler, Jonathan. 1995. "Coding Style and Good Computing Practices". *PS: Political Science and Politics* 28(3): 488-92.

Norris, Pippa. 2015. "Democracy Crossnational Data, Release4.0". Accessed from https://sites.google.com/site/pippanorris3/research/data#TOC-Democracy-Cross-national-Data-Release-4.0-Fall-2015-New-.

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

R Development Core Team. 2016. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from http://www.R-project.org.

Silberzahn, Raphael and Eric L. Uhlmann. 2015. "Crowdsourced Research: Many Hands Make Tight Work". *Nature* 526(7572): 189.

Twain, Mark. 1975. *Mark Twain's Notebooks & Journals, Volume I: (1855-1873)*. Berkley and Los Angeles: University of California Press

Jake Bowers is an Associate Professor in the Departments of Political Science & Statistics at the University of Illinois at Urbana-Champaign, and a Fellow of the White House Social and Behavioral Sciences Team. Contact email: jwbowers@illinois.edu

Maarten Voors is an Assistant Professor at the Development Economics Group at Wageningen University, the Netherlands. Contact email: maarten.voors@wur.nl