Causal Inference
2 - Good Research Practice

Benjamin Elsner
benjamin.elsner@ucd.ie

Motivation

(Actually) doing **empirical research is hard**! Projects often...

- involve **multiple collaborators**
- stretch over **many months/years**
- require combinations of multiple **datasets**
- ...as well as multiple **programming languages** (R, Stata, RMarkdown, LaTeX, Python)

All these are major **stumbling blocks to good research**

- Difficult to spot errors
- Inconsistent and inefficient processing of information between collaborators
- Research is often not reproducible

Motivation

Most good journals require researchers to post their **data and code online!**

But how to *produce* good research? We will learn about several tools:

- Automation of code
- Version control
- Organization of code
- Project management

This lecture is **based on Gentzkow & Shapiro (2014)**. More comprehensive guides can be found elsewhere on the web.

Automation

Each empirical paper is (partly) the result of a piece of software. Codes are called in sequence. Typical research project:

- Open Stata
- Click on Stata code 1 (*data_clean.do*) reads in the data and cleans them
- Click on Stata code 2 (*analysis.do*) performs the main analysis
- Click on Stata code 3 (*graphs_tables.do*) produces graphs and tables
- Or generate the tables manually
- Tables and Graphs are manually input into MS-Word
- Paper is sent to journal...

Problems: inconsistency, potential for errors, lack of reproducibility!

- In what sequence should we execute the programs?

Automation

Rules from Gentzkow & Shapiro (2014)

1. **Automate everything that can be automated.**
2. **Write a single script that executes all code from beginning to end.**

What this means in practice:

- Anyone should be able to **produce the entire paper in one click!**

Automation

Gentzkow & Shapiro (2014) recommend

- using a **typesetting system** such as LaTeXor RMarkdown
- creating a **single windows shell** that executes all programs in sequence

```
---- rundirectory.bat ----

stattransfer export_to_csv.stc

statase -b mergefiles.do

statase -b cleandata.do

statase -b regressions.do

statase -b figures.do

pdflatex tv_potato.tex
```

Automation

If this is too extreme, ensure the following:

- Your code has a logical structure
- In R/Stata work with a master file and subfiles
- All output has to be created by R/Stata and saved in appropriate formats
- LaTeXor RMarkdown automatically inputs graphs and figures

This would reduce the task to two clicks. But programming a shell is easy, so why not reduce it to one click?!

Do as I say, not as I do...

Automation — Bottom Line

Automation is **relatively easy**

It pays **huge dividends**

Regular review of the process is recommended

Costs:

- Convincing collaborators...
- Self-control

Version Control

The (in)famous "date-initial method"

- New versions of code will get the date and initial stamp

```
cleandata_022113.do      cleandata_022613.do      regressions.log
cleandata_022113a.do     cleandata_022613_jms.do  regressions_022413.do
chips.csv                tvdata.dta               regressions_022713_mg.do
regressions_022413.log
```

Version Control

Problematic about the "date-initial method"

- No track of version history (despite "1.0, 1.1, 2.0 etc")
- Which versions of code produce the final results?!
- Who changed that variable definition and why?!
- Why is this effect no longer significant?
- You overwrote on DropBox what I had written yesterday...

No piece of software on any of your devices has been written with the date-initial method!

USE VERSION CONTROL!

Version control gets you around this problem

Basic idea:

- **Most recent version** *and* **all previous versions** are in a central repository
- Collaborators **work locally** but **check versions in and out of the repository**

The tool for version control is **git**

Version Control

What you need

- a **git repository** available for free with github, gitlab, bitbucket and others
- **connect your computer** with the repository (via a virtual key)
- **collaborators** who live in the 21st century
- Frustration tolerance at the start...

Version Control

Basic workflow with git

- **pull** the latest version from the repository (i.e. check out)
- change the code
- **commit** the changes to the repository (i.e. give a timestamp)
- **push** the changes to the repository (i.e. check out)

git allows authors to **simultaneously work on code**

- it highlights conflicts between versions
- merging two versions is easy

Version Control

Your directory will look much better...

```
rundirectory.bat   tvdata.dta

cleandata.do       regressions.do

chips.csv          regressions.log
```

Main advantages:

- **transparency:** know what was changed, when and by whom
- **consistency:** can go back to any prior version;
- **easier collaboration:** avoid forking conflicts; resolving conflicts is easy
- **replicability**

Version Control

Git syncs all files, even pdf and all the auxiliary LaTeX files (.synctex.gz, .aux, .bbl,...)

But these **cannot be overwritten** on other computers because they don't consist of code

Put **only code under version control**. Add a file "**.gitignore**" to the directory, with the following content:

```
.*
*.aux
*.bbl
*.bcf
*.blg
*.pdf
*.log
*.out
*.run.xml
*latexmk
*synctex.gz
```

Version Control

git works with the **command line**. I recommend using an **interface**:

- Windows: Tortoise git
- Windows: Github Desktop
- Mac: Fork

Setting up git is a pain

- but once it works, it works well
- there are lots of online tutorials

Version Control — Bottom Line

YOU NEED TO USE VERSION CONTROL!

- I want to see your version history in all assingments from now!

Recommendations

- Hang in there! It is tricky to set up and unusual in its usage
- But it will pay off massively
- You will only need basic features of git

Organizing Directories

Motivation: it is not uncommon that all files are put in a single directory

```
                    ---C:/tv_and_potato/---
chips.csv       mergefiles.do          tv_potato_submission.pdf
cleandata.do    regressions_alt.do     tv_potato.tex
extract0B.xls   regressions_alt.log    tv.csv
fig1.eps        regressions.do         tvdata.dta
fig2.eps        regressions.log        rundirectory.bat
figures.do      tables.txt             export_to_csv.stc
```

Problems:

- Number of files expands over time
- Difficult to keep track
- Folders are not easily portable

Organizing Directories

Goal(s)

- Replication should **work on any computer** (given adequate software & power)
- Your code and folders should be geared towards replication
- Ultimately: paper should be **reproducible on another machine in one click!**

What is needed?

- An internally consistent, **logical folder structure**
- Separation of **folders for inputs, outputs and temporary files**

Organizing Directories

Better?

```
---C:/build---
/input
    extract0B.xls

/code
    rundirectory.bat
    export_to_csv.stc
    mergefiles.do

/output
    tvdata.dta

/temp
    chips.csv
    tv.csv
```

```
---C:/analysis---
/input
    tvdata.dta (link to C:/build/output)

/code
    rundirectory.bat
    regressions.do
    regressions_alt.do

/output
    fig1.eps
    fig2.eps
    tables.txt

/temp
    regressions.log
    regressions_alt.log
```

Inputs, outputs, code and tempfiles are clearly separated!

Organizing Directories

Test before submitting a paper:

- **Create a shell** that runs all the code in sequence
- **Copy input and code files** into a new directory
- Does the entire paper get **reproduced**?

Bottom line: Find a directory structure that suits your workflow and allows for reproducibility

Abstraction

Problems with writing code

- Difficult to understand when revisiting after months/years
- Redundancies (same procedures used over again)
- Inconsistencies (e.g. same procedure used differently in different files)

Abstraction can help with these problems

Abstraction

Example: we want to generate a **leave-out-mean at the state-level**

```
egen total_pc_potato = total(pc_potato), by(state)
egen total_obs = count(pc_potato), by(state)
gen leaveout_state_pc_potato = (total_pc_potato - pc_potato) / (total_obs - 1)
```

Now we want to have the leave-out means at different levels of aggregation (county, MSA).

- One solution: copy-paste the same code three times, replacing “state”
- Problem 1: messy code
- Problem 2: change in the formula requires changing three parts of the code; this creates potential for error

Abstraction

Solutions

- **Within a code/project**, write programs (Stata) or functions (R)
- **Across projects**, write ado files (Stata) or functions (R) that can be accessed by multiple sources
- Whatever you do, put it under **version control**!

Abstraction

Example for a Stata program

```
program leaveout_mean

    syntax, invar(varname) outvar(name) byvar(varname)

    tempvar tot_invar count_invar

    egen `tot_invar'= total(`invar'), by(`byvar')

    egen `count_invar'= count(`invar'), by(`byvar')

    gen `outvar' = (`tot_invar' - `invar') / (`count_invar' - 1)

end
```

Afterwards we call the program

```
leaveout_mean, invar(pc_potato) outvar(leaveout_state_pc_potato) byvar(state)
leaveout_mean, invar(pc_potato) outvar(leaveout_metro_pc_potato) byvar(metro)
leaveout_mean, invar(hh_potato) outvar(leaveout_metro_hh_potato) byvar(metro)
```

Abstraction

Rules of Gentzkow & Shapiro (2014)

- Abstract to eliminate redundancy
- Abstract to improve clarity
- Otherwise, don't abstract

Soft-coding

An important source of error is hard-coding. Look at this

Untitled.do*

```
1
2
3     use "C:/project/data/mydatafile.dta", clear
4     gen demand=price^(-2.5)
5     regress demand weather, cluster(state)
6     save "C:/project/data/newfile.dta"
```

The demand elasticity is hard-coded. As is the path. As is the level of clustering standard errors.

Soft-coding

Better:

```
 9
10    global directory "C:/project/"
11    global clustervar "state"
12    local priceelas=2.5
13
14    use "${directory}/data/mydatafile.dta", clear
15    gen demand=price^`priceelas'
16    regress demand weather, cluster(${clustervar})
17    save "${directory}/data/newfile.dta", replace
18
```

- Can easily transfer the analysis to a different directory
- Keep things like the cluster variable consistent
- Ensures we use the same demand elasticity throughout the code

Documentation

Gentzkow & Shapiro (2014) have a whole chapter on code documentation. Read it. Do it!

Project Management

All research projects are complex:

- They involve many tasks to be completed in logical order
- They often involve many collaborators
- (Sequences of) tasks may need to be revisited

This creates two challenges

- Keeping track of tasks
- Communication between collaborators

Email is not a Project Management Tool

Email is messy

Very hard to keep track of to-do-lists

Difficult to assign tasks

Difficult to revisit (discussions of) old tasks

Things slip through the cracks

Use Project Management Tools

Kanban-based platforms:

- Trello
- Asana
- Monday.com

Team communication platform: Slack

All of them have free plans and sync across devices!

Use Project Management Tools

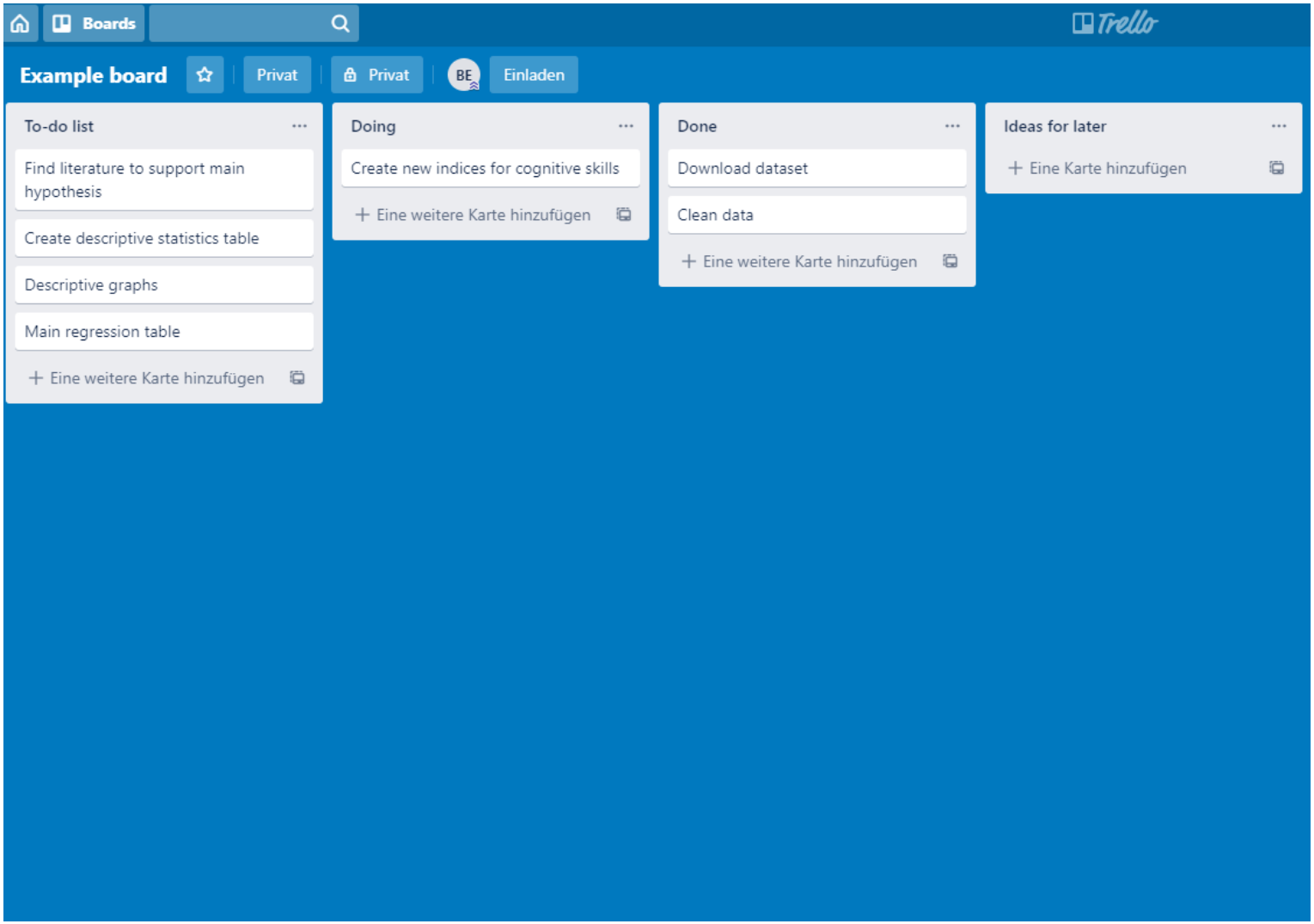
Kanban-based platforms:

- Trello
- Asana
- Monday.com

Team communication platform: Slack

All of them have free plans and sync across devices!

Use Project Management Tools



Use Project Management Tools

Put a bit of thought into how you work

Develop a system you trust and that works for you!

The payoff is almost immediate.

One (of the very few) self-help books I can recommend: Getting Things Done by David Allen

Other Things I Recommend

Use a **Notetaking Software/App**:

- Evernote is great
- Alternatives: Google Keep, OneNote

References I

Gentzkow, Matthew, & Shapiro, Jesse M. 2014. Code and Data for the Social Sciences: A Practitioner's Guide. *University of Chicago*.