Research Methods for Economics and Policy
Data and Methods

Lorenzo Neri[1]

[1] University of Rome “Tor Vergata”, CESifo and IZA

April-May 2025

Useful reference for general practices

- Code and Data for the Social Sciences (Gentzkow and Shapiro)

What's the issue?

- Coding is an essential aspect of applied research: we use coding for cleaning, scraping, analysing data, etc
- Though we all write code for a living, few of us have any formal training in computer science
- Potential issues:
  - Failing to replicate an earlier version of a set of estimates because the files had been moved and even when you recover them, the results you get are different
  - Observations disappearing when making merges
  - Copy-pasting sample selection chunks across different code files - and suddenly realising that one of them does something different
  - Dataset held in a shared space modifid by a member of the research team
- This lecture is about translating insights from experts in code and data into practical terms for empirical social scientists

Automation (1)

- Suppose you have a research project where you have 3 raw data files, all in csv: team.csv, price.csv and players.csv.
- These were sent to you or downloaded from the internet (e.g., via scraping). You want to merge them and run a model where you estimate the effect of player performance on his price. How do you proceed?

Automation (1)

- Suppose you have a research project where you have 3 raw data files, all in csv: team.csv, price.csv and players.csv.
- These were sent to you or downloaded from the internet (e.g., via scraping). You want to merge them and run a model where you estimate the effect of player performance on his price. How do you proceed?

One approach:

- Load data on R, convert into R files, then use them to estimate the regression.

Automation (1)

- Suppose you have a research project where you have 3 raw data files, all in csv: team.csv, price.csv and players.csv.
- These were sent to you or downloaded from the internet (e.g., via scraping). You want to merge them and run a model where you estimate the effect of player performance on his price. How do you proceed?

One approach:

- Load data on R, convert into R files, then use them to estimate the regression.
- Then, open a new MS Word/latex file, copy the output from the results window of the statistical program into a table, write up an exciting discussion of the findings, and save $\rightarrow$ submit to a journal.

Automation (1)

- Suppose you have a research project where you have 3 raw data files, all in csv: team.csv, price.csv and players.csv.
- These were sent to you or downloaded from the internet (e.g., via scraping). You want to merge them and run a model where you estimate the effect of player performance on his price. How do you proceed?

One approach:

- Load data on R, convert into R files, then use them to estimate the regression.
- Then, open a new MS Word/latex file, copy the output from the results window of the statistical program into a table, write up an exciting discussion of the findings, and save $\rightarrow$ submit to a journal.

$\Rightarrow$ What's wrong with that?

Automation (2)

1. Replicability
   - First, innumerable things can go wrong

Automation (2)

1. Replicability
   - First, innumerable things can go wrong
   - There is no record of the precise steps that were taken

Automation (2)

1. Replicability
   - First, innumerable things can go wrong
   - There is no record of the precise steps that were taken
2. Efficiency
   - If we decide to change any aspect of the analysis, we will have to go back and repeat all of the steps of building and cleaning the data

Automation (2)

1. Replicability
   - First, innumerable things can go wrong
   - There is no record of the precise steps that were taken
2. Efficiency
   - If we decide to change any aspect of the analysis, we will have to go back and repeat all of the steps of building and cleaning the data

$\Rightarrow$ In a real project, there might be a thousand steps from raw data to final results. You need to automate.

Automation (2)

1. Replicability
   - First, innumerable things can go wrong
   - There is no record of the precise steps that were taken
2. Efficiency
   - If we decide to change any aspect of the analysis, we will have to go back and repeat all of the steps of building and cleaning the data

$\Rightarrow$ In a real project, there might be a thousand steps from raw data to final results. You need to automate.

$\Rightarrow$ Now, suppose you have coded up the different steps. You have written 2/3 separate code files and saved them in a folder. What's wrong with that?

Automation (2)

1. Replicability
   - First, innumerable things can go wrong
   - There is no record of the precise steps that were taken
2. Efficiency
   - If we decide to change any aspect of the analysis, we will have to go back and repeat all of the steps of building and cleaning the data

$\Rightarrow$ In a real project, there might be a thousand steps from raw data to final results. You need to automate.

$\Rightarrow$ Now, suppose you have coded up the different steps. You have written 2/3 separate code files and saved them in a folder. What's wrong with that?

$\Rightarrow$ The less you manually operate with the data (and the more you automate), the better.

Automation (2)

1. Replicability
   - First, innumerable things can go wrong
   - There is no record of the precise steps that were taken
2. Efficiency
   - If we decide to change any aspect of the analysis, we will have to go back and repeat all of the steps of building and cleaning the data

$\Rightarrow$ In a real project, there might be a thousand steps from raw data to final results. You need to automate.

$\Rightarrow$ Now, suppose you have coded up the different steps. You have written 2/3 separate code files and saved them in a folder. What's wrong with that?

$\Rightarrow$ The less you manually operate with the data (and the more you automate), the better. NEVER modify the data manually - if you need to do that, code it up.

Version Control

- You will find yourself writing several versions of a code - e.g., regressions-footy-2904.R, regressions-footy-2904-LN.R, regressions-footy-2904-EB.R, regressions-footy-3004-LN.R, and so on....

Version Control

- You will find yourself writing several versions of a code - e.g., regressions-footy-2904.R, regressions-footy-2904-LN.R, regressions-footy-2904-EB.R, regressions-footy-3004-LN.R, and so on....
- This generates an endless set of code files that, depending on the tasks they perform, potentially change the underlying data and interact between each other in non-trivial ways...

Version Control

- You will find yourself writing several versions of a code - e.g., regressions-footy-2904.R, regressions-footy-2904-LN.R, regressions-footy-2904-EB.R, regressions-footy-3004-LN.R, and so on....
- This generates an endless set of code files that, depending on the tasks they perform, potentially change the underlying data and interact between each other in non-trivial ways...
- Best solution: version control $\Rightarrow$ You set up a ”repository” on your PC (or, even better, on a remote server).
- You can edit without fear. If you make a mistake, or if you start in a new direction but later change your mind, you can always roll back all or part of your changes with ease.
- You maintain a single version of your file and can go back to previous version - similar to files stored on Dropbos or GoogleDrive

Directories

- Having a single directory that has and does everything has some appeal, but for most real-world research projects this organizational system is not ideal

Directories

- Having a single directory that has and does everything has some appeal, but for most real-world research projects this organizational system is not ideal
- You may want to re-run or change only certain parts of the analysis, or maybe update some commands...
- Also, consider projects making use of large administrative files (either raw or ”modified”)...

Directories

- Having a single directory that has and does everything has some appeal, but for most real-world research projects this organizational system is not ideal
- You may want to re-run or change only certain parts of the analysis, or maybe update some commands...
- Also, consider projects making use of large administrative files (either raw or ”modified”)...
- Depending on the project, you may also want to i) have a consistent subdirectory structure within each high-level directory and ii) a shell file that controls it

Directories

- Having a single directory that has and does everything has some appeal, but for most real-world research projects this organizational system is not ideal
- You may want to re-run or change only certain parts of the analysis, or maybe update some commands...
- Also, consider projects making use of large administrative files (either raw or ”modified”)...
- Depending on the project, you may also want to i) have a consistent subdirectory structure within each high-level directory and ii) a shell file that controls it
- Avoid creating multiple files: work using shared directories

Keys (really, data housekeeping) (1)

county	state	cnty_pop	state_pop	region
36037	NY	3817735	43320903	1
36038	NY	422999	43320903	1
36039	NY	324920	.	1
36040	.	143432	43320903	1
.	NY	.	43320903	1
37001	VA	3228290	7173000	3
37002	VA	449499	7173000	3
37003	VA	383888	7173000	4
37004	VA	483829	7173000	3

$\Rightarrow$ What's wrong with this?

Keys (really, data housekeeping) (2)

- Keep separate datasets with *unique*, *non-missing* identifiers
- In each dataset, you can add information that pertains to that specific unit (e.g., latitude and longitude of the centroid of the geographic unit your are considering)
- You want the final dataset to be a rectangularised version with non-missing keys
- Ideally, compute all variables that relate to a given unit (e.g., county) in the relevant dataset (e.g., county-level), *before* merging this dataset to another one
  - This also avoids trivial mistakes - consider standardising a state-level variable in a county-level dataset. What's wrong with this?
- Only after that, merge the different datasets to create a final dataset for your analysis. (please, never, never, never do m:m merges...)

Abstraction (1)

- Suppose you want to compute a leave-out mean. You write the following code:

```
df <- df %>%
 group_by(state) %>%  mutate(t_pc_x = sum(pc_x), t_obs = n()) %>%
 ungroup %>%
 mutate(leaveout_pc_x = (t_pc_x-pc_x)/(t_obs-1)
```

- But then you want to change the level of aggregation, and you do the following:

```
df <- df %>%
 group_by(metroarea) %>%  mutate(t_pc_x = sum(pc_x)) %>%
 group_by(state) %>%  mutate(t_obs = n()) %>%
 ungroup %>%
 mutate(leaveout_pc_x = (t_pc_x-pc_x)/(t_obs-1)
```

- Finally, maybe you want to change unit of measure...

```
df <- df %>%
 group_by(metroarea) %>%  mutate(t_hh_x = sum(hh_x)) %>%
 group_by(state) %>%  mutate(t_obs = n()) %>%
 ungroup %>%
 mutate(leaveout_hh_x = (t_hh_x-pc_x)/(t_obs-1)
```

- You have copy-pasted your code twice, making mistakes along the way....

Abstraction (2)

- Consider instead writing the following general-purpose function computing the leave-out mean:

```
leaveout_mean <- function(data, invar, byvar, outvar = "leaveout") {
 invar <- rlang::ensym(invar)
 byvar <- rlang::ensym(byvar)
 outvar <- rlang::ensym(outvar)

 data %>%
 group_by(!!byvar) %>%
 mutate(
 total = sum(!!invar),
 count = n(),
 !!outvar := (total - !!invar) / (count - 1)     ) %>%
 select(-total, -count) %>%
 ungroup()}
```

1. Amount of copy-pasting is minimised
2. This can be re-used in different projects
3. A clearly labelled function can be easier to understand for the reader

Abstraction (2)

- Consider instead writing the following general-purpose function computing the leave-out mean:

```
leaveout_mean <- function(data, invar, byvar, outvar = "leaveout") {
 invar <- rlang::ensym(invar)
 byvar <- rlang::ensym(byvar)
 outvar <- rlang::ensym(outvar)

 data %>%
 group_by(!!byvar) %>%
 mutate(
 total = sum(!!invar),
 count = n(),
 !!outvar := (total - !!invar) / (count - 1)     ) %>%
 select(-total, -count) %>%
 ungroup()}
```

1. Amount of copy-pasting is minimised
2. This can be re-used in different projects
3. A clearly labelled function can be easier to understand for the reader

$\Rightarrow$ Also, consider doing “unit testing”

Documentation (1)

- It is important to clearly comment your code, but do not overdo it
- Try to incorporate the content of the comments into the code. Consider this example:

```
# Elasticity = Percent Change in Quantity / Percent Change in Price
# Elasticity = 0.4 / 0.2 = 2
# See Shapiro (2005), The Economics of Potato Chips, Harvard University Mimeo,
# Table 2A.
compute_welfare_loss(2)
```

- and then after a few months you find the followins:

```
# Elasticity = Percent Change in Quantity / Percent Change in Price
# Elasticity = 0.4 / 0.2 = 2
# See Shapiro (2005), The Economics of Potato Chips, Harvard University Mimeo,
# Table 2A.
compute_welfare_loss(3)
```

- **What are the (several and potential) issues here?**

Documentation (2)

- So now consider an improved version. The original version:

```
# Elasticity = Percent Change in Quantity / Percent Change in Price
# Elasticity = 0.4 / 0.2 = 2
# See Shapiro (2005), The Economics of Potato Chips, Harvard University Mimeo,
# Table 2A.
compute_welfare_loss(2)
```

  vs this:

```
percent_change_in_quantity <- -0.4
percent_change_in_price    <- 0.2
elasticity <- percent_change_in_quantity/percent_change_in_price
compute_welfare_loss(elasticity)
```

- This makes clear both the formula for the price elasticity and the quantitative components you are using. But it is far better than the original code, because it has far less scope for internal inconsistency
- When possible, then, you should write your code to be self-documenting

Documentation (3)

- Documentation can be used to make clear that something is right when it at first may seem wrong...

```
# standardise language score
# writing tests were abolished in 2011. See here: XXX.
# we use reading only throughout to be consistent
tscore_kids <- tscore_kids %>%
  mutate(language_score_std = scale(reading_score))
```

Documentation (3)

- Documentation can be used to make clear that something is right when it at first may seem wrong...

```
# standardise language score
# writing tests were abolished in 2011. See here: XXX.
# we use reading only throughout to be consistent
tscore_kids <- tscore_kids %>%
  mutate(language_score_std = scale(reading_score))
```

- ...and can also be used to prevent unintended behavior

Documentation (3)

- Documentation can be used to make clear that something is right when it at first may seem wrong...

```
# standardise language score
# writing tests were abolished in 2011. See here: XXX.
# we use reading only throughout to be consistent
tscore_kids <- tscore_kids %>%
  mutate(language_score_std = scale(reading_score))
```

- ...and can also be used to prevent unintended behavior
- If there are some inputs you really, really want to prevent, comments that say “don’t ever do X” are not the way to go

  → Write your code so it will not let those inputs in the door in the first place

Code style

- Keep it short and purposeful - avoid having scripts longer than a few hundred lines (now, in practice...)
- Use descriptive names
- Pay special attention to coding algebra - it may be useful to break complicated algebraic calculations into pieces
- Be consistent
- Check for errors
- Write tests
- Be conservative - store “too muc” output from slow code