Reproducible R Workflows for Research Papers

How Not To Screw Your Future Self

Dr. Seo-young Silvia Kim

Jan 25, 2021, R-Ladies Dallas Meetup

American University

Reproducible Research with R For Academics

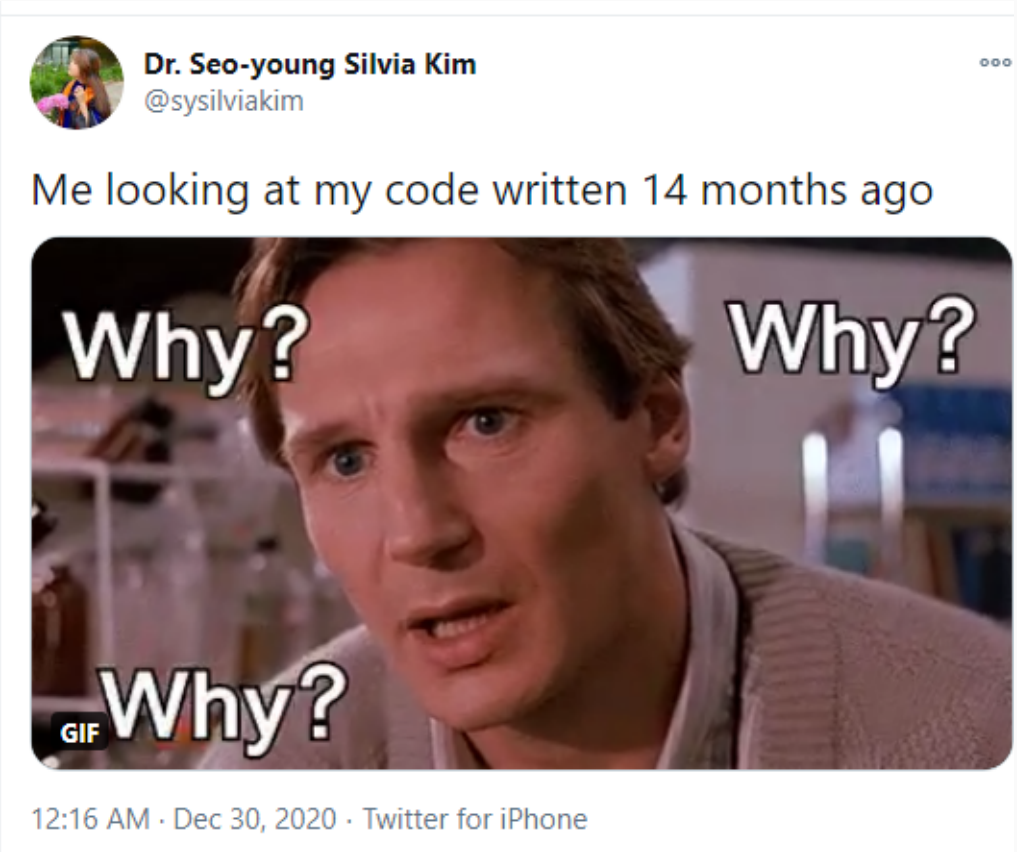
Alternatively,

- If you are a junior scholar and still not working on reproducibility/open science a big no-no
- How not to screw your future self
- Or ten times I got screwed—still being screwed—by my past self

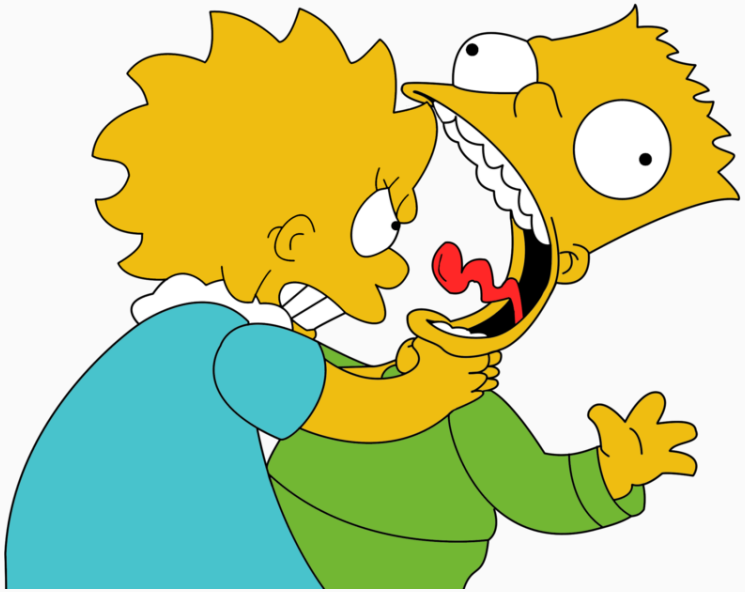
Introduction

- Assistant Professor, Department of Government, American University
- Political behavior, elections, campaign finance
- One of those who writes a tidyverse-approach answer to every Stack Overflow question
- On Twitter @sysilviakim (and same on GitHub)
- Disclaimer: no one size fits all
  The workflow may be completely different in a different discipline... so modify as needed!

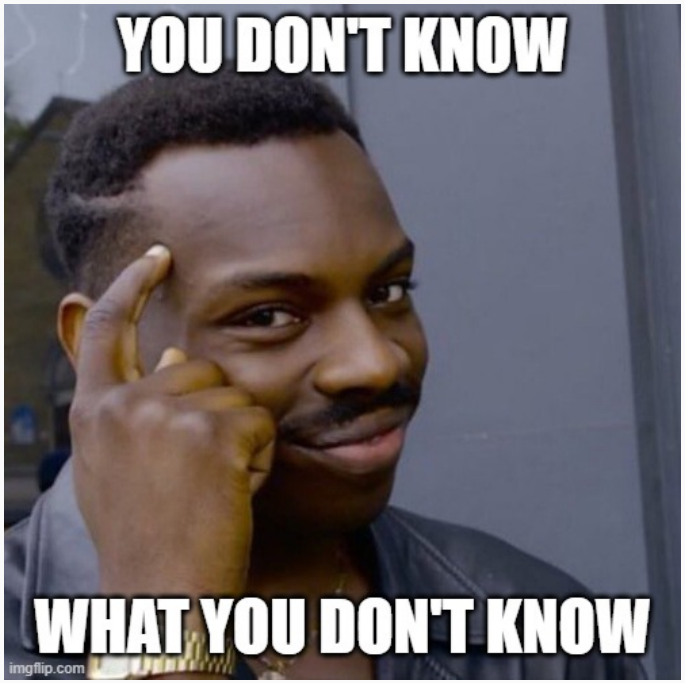
“What Were You Thinking?!”



“What Were You Thinking?!”



Well, I Was Learning



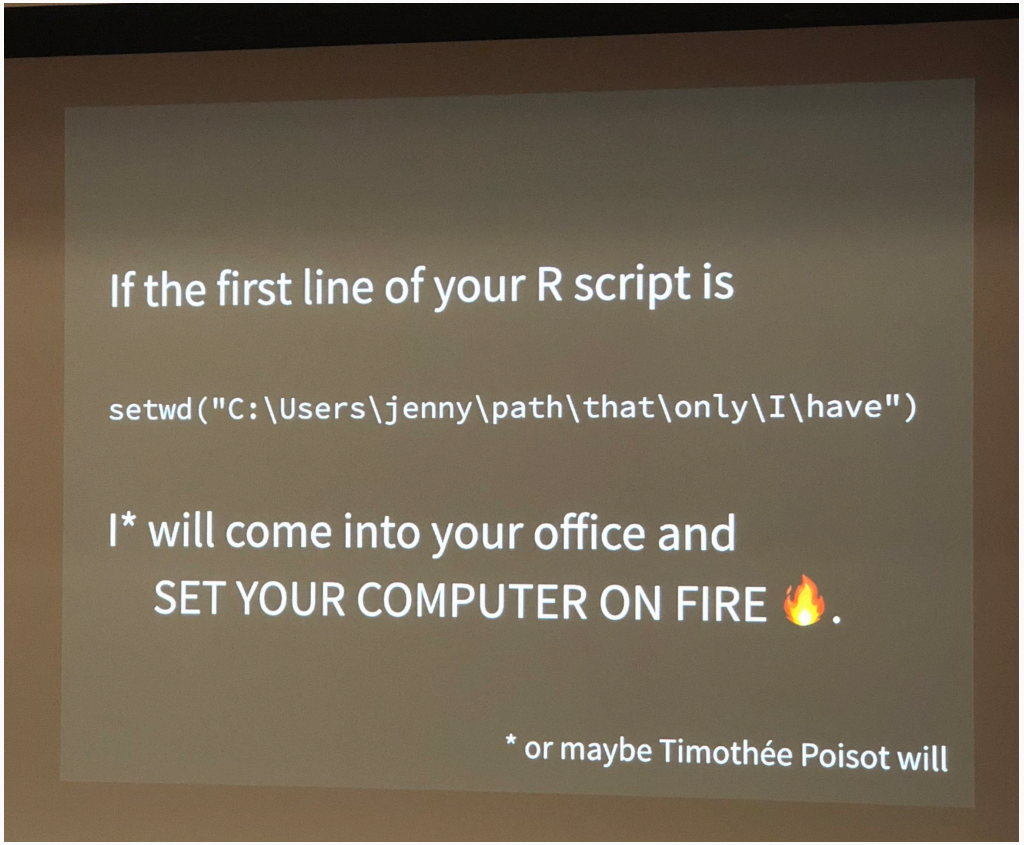
And I’m still learning!

Sin 1: Not Working in an R Project

- A related sin: using `setwd`, which massively screws your coauthors
- My favorite workflow
    1. Open RStudio
    2. File tab ⟶ new project ⟶ new directory ⟶ "New Project"
    3. Check "Use renv with this project" (Already checked: "Create a git repository")
    4. A set of initializing code that I like to run at my GitHub Gist
    5. Tools ⟶ Global Options⟶ Uncheck all that says "Restore" "Always"
    6. Now go to Code tab ⟶ from the Display tab, "show margin: Margin column 80"
- .Rproj remembers the root directory
- If starting from a remote,
    1. (GitHub ⟶ create new repository
    2. Git clone to your local drive
    3. File tab ⟶ new project ⟶ existing directory ⟶ "New Project"
    4. `renv::init`

A Related Sin: Using setwd

From the reputable Jenny Bryan:



See also Hadley Wickham's Twitter thread at
https://twitter.com/hadleywickham/status/940021008764846080?s=20

Sin 2: Not Using Git (Version Control)

- "I use Dropbox so I should be okay!" or
  "This is not software development so I should be okay!" $\rightsquigarrow$ NO.
- On principle: assume your hard drive will die tomorrow morning
  $\longrightarrow$ commit and push to your remote, please
- Easy to see what changed between commits
  $\longrightarrow$ can undo these commits easily, if something goes wrong (free time machine!)
- Use branches to test out new things: if it doesn't work you are still okay!

Sin 3: Not Using renv package

- Pronounced ARE-ENV
- `sessionInfo()` output matters: what packages and their versions are you relying on?
- Massively screwed up when, for example, `tidyr` changed its `.name_repair` convention
- Another time when `rlang` / `dplyr` had major updates and deprecated functions I was using
- Argument for using base R instead of `tidyverse`? Rather, just use `renv`
- Ready to go the extra mile?
  Use Docker (https://rstudio.github.io/renv/articles/docker.html)
- I would also store some of the intermediary wrangled outputs, to compare if something changes and the outputs do not match

Sin 4: Not Having A Consistent Directory Structure

- Develop your own routine: mine is at
  `https://github.com/sysilviakim/Kmisc/blob/master/R/proj_skeleton.R`
- All scripts are under `R` subfolder
  Use `file.path` to refer to datasets
- This is more of an across-projects sin rather than for a single project:
  You'll be confused when you jump around projects

Sin 5: Messing with Your Raw Data

- Automate data import if you can but don't overwrite it
- Beware of URL rots (Wayback Machine?)
- Don't you dare touch anything in `data/raw`! Wrangling data? Put it in `data/tidy`

  If you use functions like `fix` I'll throw your computer into the Potomac river
- Trusting your raw data (and your package) blindly can also lead you to problems
  - Half of my data cleaning is wasted on unclosed quotes and bad delimiters that may or may not work with `read.table` or `readr::read_delim`
  - If you can, open the raw data, count the number of rows, and see if it matches the imported object's `nrow` (I use 010 Editor)

My Own Example

R utils::download.file causes data error for .txt.gz? + Recovery?

Ask Question

Asked 9 months ago Active 9 months ago Viewed 26 times

0

See the following code (Windows 10, R 3.6.3)

```
download.file(
    url = "https://www6.ohiosos.gov/ords/f?p=VOTERFTP:DOWNLOAD::FILE:NO:2:P2_PRODUCT_NUMBER:363"
    destfile = paste0("SWVF_1_22_", format(as.Date(Sys.Date()), "%Y%m%d"), ".txt.gz")
)
```

Once I try to unzip this with 7zip, this simply says "Data error."

- The funniest thing is, this code *used to work before*, going back to late Jan 2019.
- In addition, if you go to the original url, it downloads fine and it is also unzipped properly via either 7zip or WinZip.
- Downloaded files either way have the same file size, which is why I never suspected corruption of files for an entire year and more. I've tried changing `destfile` argument, as well as using `tempfile()`. Neither works.
- This style of `download.file` works fine for `zip` files. Just this `.txt.gz` files.

I have absolutely no idea what is going on, and whether my files can be recovered at all. Any advice?

r download gzip 7zip corrupt

Share Edit Close Delete Flag

asked Apr 25 '20 at 1:56
Kim
2,862 ● 2 ● 21 ● 43

Try adding `mode="wb"` to the `download.file()` call. You'll want to do that for binary data like a gz file. Otherwise "line endings" are translated to the native OS default which is not what you want for binary data. – MrFlick Apr 25 '20 at 2:12

See https://stackoverflow.com/questions/61420050: Windows vs. Unix \r \n vs. \n problem
Urge to open Notepad++ and then CTRL + H replace and save ⇝ nope nope nope

Sin 6: Not Having A Set of Ordered, Modular R Scripts

- Create a makefile of some sort
  Best example I have seen so far: https://github.com/zmjones/eeesr
- If using an HPC (batch jobs) you might need to create some additional files
- Rscripts should be named in order (minor tip: name it 01, 02, 03, ...) and executed as such
- I like to have a `utilities.R` or `fxns.R` code that I source at the start of every script by `source(here::here("R", "utilities.R"))`
- Proceed sequentially but run all scripts fresh in a new R Session
- Re-run scripts that have changed so you are not relying on old outputs/environments, and do a final and full run when things are wrapped up
- Prefer Rmd? Sure! Make sure to use R package `here` such as `here::here()`

Sin 7: Copy-Pasting Code

- Goes squarely against the DRY principle in programming: "Don't Repeat Yourself."
  - If you are doing something repetitive, 95% of the time it can be automated
  - Not only is it stupid it is also dangerous
    prone to mistakes that are extremely hard to detect
- Using `purrr` package's `map`, `imap`, `cross2`, or such functions solves a lot of these problems
- If you have a process that you will repeat over multiple objects, *create a function*!
- Better yet, *create your own package*! Mine is at https://github.com/sysilviakim/Kmisc
  (and maybe, one day, I'll do a CRAN submission.)

- Related sin: likely that you are also hardcoding things:
  - Example: suppose you must use the most recent file with prefix + date:
    `data_file_20210125.txt` and similar files
  - Don't `read.table("data_file_20210125.txt"`

    A combination of `list.files` + parsing dates + `max`
    and you don't ever have to retouch your code again!

Sin 8: Not Commenting Properly

- Human memory is *flawed*
  No, you are not going to remember what you are doing with that clever line of code
  Takes just two weeks to forget what you were doing
- Sparse commenting $\longrightarrow$ wastes time when you are trying to remember what you were doing
- Section your code: can fold (ALT + O) or unfold (ALT + SHIFT + O), can have informative titles

Sin 9: Not Writing/Deploying Tests

- Unit testing but for academic data analysis
- Does this line of code do what I expect it to do?
  Don't check it manually—write it down as a test!
- My favorite: `assertthat` package

  For example, did you replace `NA` values with something? `assert_that(!any(is.na(df$var)))`
- They are not stupid and will save you from disasters

Sin 10: Not Automating Figure/Tables

Figures

- Manual figure creation a big no-no unless this is some theoretical diagram
- See export results and patiently work through the `ggplot2` options
- Export to vectorized figures like a PDF instead of a PNG or JPEG

Tables

- Use `xtable` and `stargazer`
- If you have used `options(digits = XX)`, know that this is *scientific* digits

  For example, 300.123 $\longrightarrow$ 300, 0.123 $\longrightarrow$ 0.12 under `options(digits = 2)`
  "Oh for this group the mean is a pretty 300!" $\longrightarrow$ *wrong*
- If you want numbers after the decimal point, use `formatC` instead

Reproducible Research with R For Academics: Recap

10 Circles of Hell in reproducible research with R

1. Not working in an R project
2. Not using Git (version control)
3. Not using renv package
4. Not having a consistent directory structure
5. Messing with your raw data
6. Not having a set of ordered, modular R scripts
7. Copy-pasting code
8. Not commenting properly
9. Not writing/deploying tests
10. Not automating figures/tables

+ Probably a few more circles that I've already forgotten.

Some Extra Sins You Can Commit in Research (Not Necessarily Reproducibility)

New! rstudio::global conference,
Maintaining the house the tidyverse built by Hadley Wickham

- Hear about code maintenance from the developer!
- Avoiding off-label usage of functions
- CRAN time capsule (pick a day in the past)

Some Extra Sins You Can Commit in Research (Not Necessarily Reproducibility)

- Using `rm(list = ls())` in the beginning of the script
- Reinventing the wheel
- Having thousands and thousands of lines in a single R script or R Markdown is not good
  A script = paragraph in a writing. Should contain only one idea/purpose!
- Not using `set.seed` if relying on random sampling of some sort
- Not using a consistent coding style + not respecting max 80 char per line
  - Google Style Guide or The tidyverse style guide
  - A good package to use: `styler` for the tidyverse style guide
  - For the audience + for the future you who will inevitably forget what you were doing
- Not using debugging feature when something breaks
- Not working with a smaller portion of a gigantic dataset first
  (if it takes too long, convert to `data.table` or use `dtplyr`)
- Waiting for data collection to finish before starting data analysis
- Using number index when you can use names
  e.g., `cf.matrix$overall [[1]]` (X) `cf.matrix$overall[["Accuracy"]]` (O)

Some Extra Sins You Can Commit in Research (Not Necessarily Reproducibility)

- Not consistently naming your objects
- Not using `useNA = "ifany"` code when doing a `table`
- Or not doing sanity checks on your raw data in general, esp. about missing data
- Going outside R frequently to use multiple different languages
  (many can be run within R Markdown (SQL, Python, Stan, Bash),
  or can be used with `system` e.g., curl, 7z)
- Not using a colorblind-friendly palette (accessibility!)
- Not unifying axes scales if working with subfigures laid out side-by-side
- Having too small text in figures
- Having too many figures/tables in your main text
- Not backing up your TeX or Word file
- Not paying attention to `Sys.getlocale()` when using another language than English
- Not signing up on Stack Overflow (those upvotes/bookmarks will be useful!)

“But This Is So Time-Consuming!!”

Hey... they are at least actionable.
And, as another #rstats peep said
“Time spent making replicable analyses is never wasted.”

Conclusion: Embody the Growth Mindset!

Have I been a perfect researcher? No.
But will all my research be perfectly reproducible in the future? Also (probably) no.
But at least you can and *should* try!

Q&A + Any suggestions? Please let me know!