Introduction: Workflow

William Lowe

Hertie School of Governance

7th September 2020

Taming chaos

In data analysis there are two sorts of *surprises* and two sorts of *cognitive stress*

1. Analytical (often good)
2. Infrastructural (almost always bad)

Analytical surprise is when you learn something from or about the data

Infrastructural surprise is when you discover you that

- → you can't find what you did before
- → the analysis code breaks
- → the report doesn't compile
- → the co-author can't run your code

Data analysis can be stressful, workflow is there to ensure the only the *right kind* of stress

Taming chaos

There is a division of labour:

- → Some languages have strong feeling about how work should flow, e.g. Java, c++
- → Some do not, e.g. python and R

For languages that do not impose themselves, we must use *convention*.

So this lecture is very largely about workflow conventions that we have collectively found to work well

- → This is an R-based course, but only some of these conventions are R-specific. Feel free to export the rest to your other projects

Essentials

You should *always* think in terms of projects

A project is a self-contained unit of data science work that can be

- → shared
- → recreated by others
- → packaged
- → mothballed

It contains

- → Content, e.g. raw data, processed data, scripts, functions, documents
- → Metadata: information about tools for running it, e.g. required libraries, compilers

Essentials: Structure

It is easiest to use your *file system* to maintain this discreteness, and plain text files to record the metadata

Plain text ideology

See Healy (2019) for the advantages of working as far as possible with human-readable 'plain text' formats over proprietary and binary ones

For R projects

- → Projects are folders/directories
- → Metadata is the .Rproj files, perhaps augmented with the output of `renv`

Essentials: Structure

The ideal:

- → One folder contains everything inside it
- → All internal paths are relative and point sideways or downwards
- → Can be relocated without problem
- → Put ‘product’-generating files at the top level

Paths and working directories can confuse people (not you of course), so these get their own conventions

Essentials: Good paths

Good paths:

- → `"preprocessing.R"`,
- → `"figures/perpetual-motion.png"`

Invariant to moving or sharing the project folder

Essentials: Bad paths

Bad paths:

- → `"../code/funcs.R"`
- → `"/Users/me/data/thing.sav"`

The first is confusing but not terrible. The second will not work if you share your project

But sometimes keeping data elsewhere *is* unavoidable. Then

```
DATA <- file.path(Sys.getenv("HOME"), "data/thing.sav")
```

can be handy.

- → Hint: do all this at the *very top* of whatever product you are working on

Essentials: Bonus

You'll never have to write

```
setwd("/Users/me/whereever/I/left/the/project")
```

which is good because that won't travel either.

Just let it go…

Essentials: The working directory

Set it manually once per session (Session > Set Working Directory > Choose Directory). Then all your good paths will "just work"

Better yet, have the metadata set it for you

- → Open your session by opening (choosing, clicking on) `myproject.Rproj`
- → Then you'll get the path set for you

Bonus bonus: You won't look like an amateur…

Essentials: Metadata

With external dependencies comes chaos.

Early in a project, you can be fairly loose about libraries and dependencies

As soon as you share it, you need to care…

For regular supporting tools, e.g. latex, use the `.Rproj` file to record your choices

For R libraries:

- → lightweight solution: Put all your `library` calls at the top of the product (not in any files it calls). Rstudio will pick up uninstalled libraries and ask if you want to install them
- → heayweight solution: `renv` and the tools on the Reproducibility Task View

For system dependencies, e.g. C,C++ libraries, compilers, python versions:

- → This is *just hard*. 'Some setup required'

Essentials: Data

Don't store *derived data* or *intermediate state* without the recipe

Essentials: Data

Don't store *derived data* or *intermediate state* without the recipe

What is derived data?

- → Wrangled *raw* data, e.g. the processed survey file with just your questions in it
- → A thousand web pages scraped from a website
- → Graphs and tables

(Actually the web is a partial exception to this rule: but you still need to keep the download code)

What is derived state? The objects in your local environment

- → things you made from the Console, e.g. .RData
- → intermediate computations, e.g. compiled Stan models or Rcpp functions

Why?

Essentials: Entry points

A data science project can, and usually does, have several distinct products

- → The analysis itself
- → Graphs and tables
- → An intermediate report of the analysis
- → An academic paper
- → Overhead slides

These products should also project *entry points*

- → Q: “So, where are we on this project?”
- → A: [Knits report]

Ideally you should be able to generate each of them in one step

Essentials: Analysis entry points

`script.R` should do *everything* when `sourced`. Assert the dependencies, source other scripts, there

Otherwise you have to remember to do things in the right order

Scripting

Have a script, don't be a script

Essentials: Experimentation

Only *one* script (per product), ever other bit of code should contain *functions, not commands*

→ When something goes wrong you want to debug functions, not dig up scripts

Debugger

Make friends with `debug`, `undebug`, and the debugger

Why? *Damage limitation*

→ Functions (when written right) restrict the information being used to their parameters

Note: Sometimes they also hold information about the external environment where they were defined too, a.k.a. 'enclosures'. Try not to depend on that sort of thing…

Essentials: Experimentation

Keep the little experiments in the Console, or in a scratch file.

- → When you like the code, add it to the main script.
- → Maybe don't put that stuff in a huge code block in an `.Rmd` file

Essentials: Figures and tables

Have a folder for figures and tables

→ Be ready to overwrite everything there

One exception: RMarkdown will generate its own transient figure folder, don't bother to mess with that one

Never write data tables by hand. Just like you wouldn't write a bibliography by hand (would you?)

Choose a package that formats tables nicely and learn how to use it

→ I like `texreg` and `xtable` for Latex and `kable` for Rmarkdown

→ There are probably better choices, but I don't care

Essentials: Figures and tables

This is useful even if you *don't* use `knitr` to make your documents

For example, in the R script

```
options(xtable.floating = FALSE, latex.environments = "center",
        xtable.comment = FALSE, xtable.booktabs = TRUE,
        xtable.caption.placement = "top")
...
print(xtable(tbl), file = "tables/tab2.tex)
```

and in my Latex document:

```
\input{tables/tab2.tex}
```

Essentials: Documents

Decisions, decisions:

- → Does the analysis code go in the document? (maybe, if *simple* enough)
- → Does the table and figure code go in the document? (maybe, if it doesn't take too *long* to run)

The RMarkdown cookbook has a lot of good advice, even if you don't write in RMarkdown

- → Compile early, and compile often…

Aside: Got a beautiful slide template? e.g. `myslides.cls` or a pandoc template `myslides.tex`.

- → Put that stuff in a folder if you want your friends to use it

Essentials: Friends and collaborators

It's often tempting to set up a project assuming that you will be the only person working on it, e.g. as homework

That's almost never true.

- → Coauthors happen to the best of us
- → Even if not, there's someone else who you always have to keep happy

Essentials: Friends and collaborators

It's often tempting to set up a project assuming that you will be the only person working on it, e.g. as homework

That's almost never true.

- → Coauthors happen to the best of us
- → Even if not, there's someone else who you always have to keep happy

Future-you

Essentials: Friends and collaborators

Future-you is really the one you do organize your projects for

They are who you use version control for (more on that next week)

Most importantly, they are who will enjoy the fruits of your data science labour, or have to fight back your chaos.

→ So be kind to Future-you. Establish a good workflow. You'll thank yourself later.

References

Healy, K. (2019, October 4). *The plain person's guide to plain text social science*. https://plain-text.co/