Automation

Max Turgeon

SCI 2000–Introduction to Data Science

Lecture Objectives

- List pros and cons of automating a data analysis
- Run an `R` script from the command line
- Write a simple `Makefile`

Motivation

- Complex analyses require structure
    - You want to be able to explain what you did.
- Time- and resource-consuming analyses should only be repeated when absolutely necessary.
    - Especially if using cloud computing

What are we automating?

- We are automating parts of the analysis that generate outputs.
    - Data cleaning
    - Creating figures
    - Modeling results
- We can’t automate *everything*.
    - As the analyst, you still need to make some choices.

Why are we automating?

- The main reason for automating is **reproducibility**.
- Automation requires writing scripts/programs, which serve as documentation.
- Reduces risk of inaccuracies!
    - E.g. The figure you created, did it use the cleaned or uncleaned data?
- Automated data analyses can typically be run by another analyst.

Pros and Cons

- Advantages
  - Increases reproducibility
  - Improves collaboration
  - Easier to maintain/debug
- Disadvantages
  - Requires extra work
  - Slows you down (which can be good!)

R and the command line

- R scripts can be run from the command line using the command `Rscript`.
    - E.g. `Rscript my_script1.R`
- A few things to keep in mind:
    - Need to load packages for each script separately.
    - Load R code from other scripts using `source`.
    - Need to save the output somewhere.
- Look at demo.

Saving outputs

- Saving figures:
  - `ggsave` from `ggplot2`
  - The file extension determines the format.
- Saving data:
  - As a `csv` file using `write_csv`.
  - As binary file using `saveRDS`.

Passing arguments

Makefiles

- **Note**: This works really well on Linux and MacOS. On Windows, it's complicated...
  - You may need to install Cygwin or PowerShell
- `Makefile`s are text files that keep track of which scripts should be run in which order.
  - E.g. Want to clean data before running analysis.
- It does so by keeping track of **dependencies** of certain files (called **targets**).
- As an added bonus: if none of the dependencies have changed, there is no need to update the target.

Example i

```
file_to_create: files.it depends.on like_this.R
    python code_to_run.py
    Rscript like_this.R
```

- `file_to_create`: this is the target, i.e. the file we want to update if its dependencies change
    - Could be a figure, a CSV file, a report, etc.
- `files.it`, `depends.on`, `like_this.R`: these are the dependencies.
    - Could be the raw data, a script with functions, the data cleaning scripts, etc.

Example ii

- `python code_to_run.py` and `Rscript like_this.R` are lines of code that will be run.

Makefiles (cont'd)

- A Makefile should always be named `Makefile`, without extension.
- To run a makefile, use the command `make`.
  - The Makefile must be in the current directory.
- A rule, i.e. a block of code like in the example, will run if:
  - The target is not present already.
  - A dependency is newer than the target.
- **Very important**: Actions must be indented using a tab, not spaces!
- Makefiles often contain a special target called `all`. The other targets are usually dependencies of `all`.

Demo

- We will use the following code repository: `https://github.com/turgeonmaxime/automation-demo`
- **Goal**: Create a Makefile to automate this analysis.

Summary

- Automation improves reproducibility and reduces errors.
- `Makefile`s are a great way to keep track of dependencies within your analysis.
    - But can be a pain to make it work on Windows...
- If you want an `R`-specific solution that works on Windows, have a look at the package `targets`: `https://books.ropensci.org/targets/`