Intro to Project Management

Data Wrangling and Husbandry

03/09/2020

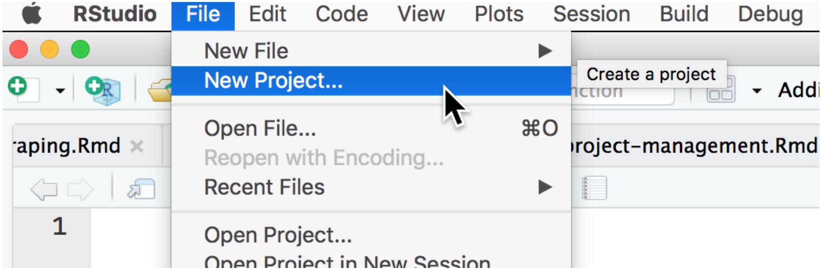
Project Management Issues

As a project becomes more complex, there are a number of issues that need to be addressed to effectively work on the project

- The number of files may increase; you need to keep tabs on them all
- Your data, your code, and your reports may go through multiple versions
- You may be working with other people who will need to share your data or code
- You may be returning to the project months later

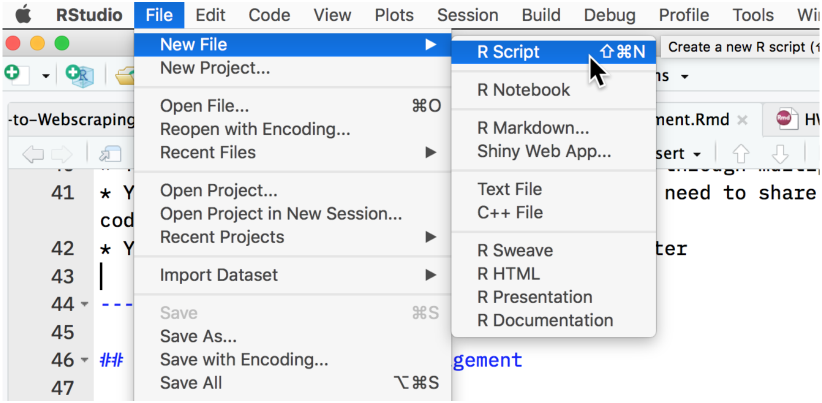
Bare Minimum Project Management

- Use a separate directory (also known as a folder) for each project
  - *Everything* needed goes in the directory
- Use RStudio's "New Project" menu item to either create that directory or to associate an existing directory with a project



Really Minimal Project Management

- Use a separate directory (also known as a folder) for each project
- Rstudio project
- Put the date on every file (including every revision) as a suffix
  - e.g., mouse-study-2018-03-01.R
- Make sure there are no R object that you generated in the console window—everything should come from an R script or an RMarkdown file.
  - I *strongly* recommend running code from an R Script or Notebook file rather than the console.


RStudio
File
Edit
Code
View
Plots
Session
Build
Debug
Profile
Tools
Wi
New File
New Project...
Open File... ⌘O
Reopen with Encoding...
Recent Files
Open Project...
Open Project in New Session...
Recent Projects
Import Dataset
Save ⌘S
Save As...
Save with Encoding...
Save All ⌥⌘S
R Script ⇧⌘N
R Notebook
R Markdown...
Shiny Web App...
Text File
C++ File
R Sweave
R HTML
R Presentation
R Documentation
Create a new R script (

Minimal Project Management

- Use a separate directory (also known as a folder) for each project
- Rstudio project
- Put the date on every file (including every revision) as a suffix
- Have separate directories for:
  - `data`, or sometimes `raw_data` and `clean_data`
  - `code`
  - `figures`
  - `results`
- (`knitr` is a little particular about file locations, so I let it write figures and output to where it wants to—I use the `figures` folder for figures that I explicitly create and save, and similarly for `result`)
- Consider including a `README.txt` or `README.md` file to describe what the files are
- Periodically zip the whole thing into a numbered and dated file?

More Sophisticated Project Management

- Use a Makefile (more later)
- Use git and github (more later)

(Minimal) Project Management and Reproducibility

- Reproducible: another group (including future you) should be able to re-create all of the results from just the code and data
- Requires planning and choice of tools
- Most essential aspects:
  - Don't touch the original data—think of it as read-only
  - Do all of your work via R scripts, notebooks, and RMarkdown files—think of your output as disposable, in the sense that it can be fully recreated from your scripts

My minimal project approach

- Separate directories for data, code, figures, and results
- Series of scripts for the work:
  - file of functions
  - one or two files to read, clean, and tidy data
    - export data at this point so that I'll have a tidy version to share in the future
  - multiple files to do the analysis work, broken up into independent analyses
  - filenames start with the order number and finish with the date if I'm not using git (e.g., `05-regressions-2016-10-31.R`)
- My final report is usually done via a RMarkdown or Rnw (for mixing R and lots of Latex) file that uses `source()` to run the script files

Workflow versus Product

Jenny Bryan distinguishes between workflow and product:

- Part of workflow:
  - The editor you use to write your R code.
  - The name of your home directory.
  - The R code you ran before lunch.
- Part of product:
  - The raw data.
  - The R code someone needs to run on your raw data to get your results, including the explicit library() calls to load necessary packages.
- Your project management approach is a way both to
  - Keep yourself organized
  - Share your work with others

More on Reproducibility through scripts

- Don't change the original data.
  - *No hand-editing*
  - Make changes to data via script, ideally including comments as to why
  - If subjects or variables are to be dropped, do so by name, not by position (e.g. `filter(ID != "A12345")) # withdrew consent`, not `example[-258, ]`
  - Ideally put the provenance of the files in the README or in the scripts that load those files

- Every aspect of cleaning and analysis goes in the scripts. No "repeat this analysis but now using y2 as the outcome"
  - If I've been working in the console, or making a lot of changes to scripts, I will typically delete everything in the global environment
    - `rm(list = ls())`, or
    - RStudio's "Session > Clear Workspace ...") to make sure I'm not using hidden definitions
- Save your seed (e.g., `set.seed(41540130)`) if your analysis involves random number generation.
- Use relative paths like "../data/example.csv", not "~/DropBox/OSC/Smith/data/example.csv"
  - There is a new package `here` that makes finding files in your project directory really easy: `here("example.csv")`

- There are even packages like `packrat` and `checkpoint` to save the versions of the various packages that you used. Recording the results of `sessionInfo()` is sufficient for most purposes, however. Microsoft hosts a CRAN time machine that can be useful for using an older version of a package
- If you do something repeatedly, you almost always want to do it via a function
    - the code is easier to follow and you are less like to make a change in one place but not another

- Refer to rows and variable by name, not position, and don't assume that the rows are in any particular order

```
## example from https://www.stat.ubc.ca/~jenny/STAT545A/blo
## left-hand figure; code contains 44 characters
xyplot(gDat[427:568,5]~log(gDat[427:568,6]))

## right-hand figure; code contains 202 characters
jYear <- 1967
xyplot(lifeExp ~ gdpPercap, gDat,
       subset = year == jYear, main = paste("year =", jYear
       group = continent, auto.key = TRUE,
       scales = list(x = list(log = 10, equispaced.log = FA
```

Recommended order for an R script file

1. Load packages
2. Set data independent constants
   - constants that depend on the data can go in the body of the code, but should be based on functions that invoke the data, e.g. `1:nrow(example)`, not `1:290`
3. Define functions (or `source()` a file with functions)
4. Get to work

Even if I have only a single RMarkdown file I'll put the first 3 steps in the `setup` chunk.