Assignment 02

Causal Inference and Research Design

Scott Cohn

Last compiled on 07 February, 2021

Gentzkow and Shapiro

Ch02

Chapter 02 is about *Automation*. A research project should be able to be replicated and that replication should be as simple as possible. This eliminates all of the possible mishaps that may occur in intermediate processing. Reverse-engineering someone else's project to see how they got their results is not fun, nor should it be expected. It is the responsibility of the researcher to create script or roadmap telling the OS how to run the directory that should output the correct results.

Ch03

Chapter 03 is about *Version Control*. This is super important, especially when working with coauthors. It eliminates the need to tons of draft files and `_v12_Final_DEFUNCT.do` scripts. It is too hard to keep track of multiple versions. Let Git do it for you. Moreover, it is easier to backtrack if things go awry ... which they often do.

Ch04

Chapter 04 is about *Directories*. In addition to keeping project replicable and revision controlled, it is important to make the project organized! We do this with clear directories. We have directories for raw data, input, analysis, etc. Here, it makes it easy to see where the project started, and what tools were used to create outputs, and how those outputs were used. Moreover, it makes is so much easier to find things. This is great when someone emails you 5-10 years after the paper was published and asks for your code.

Ch05

Chapter 05 is about *Keys*. Relational databases are super important. They are fundamental when working with large data structures that can't be directly loaded into memory and have to be queried (using something like SQL).

Ch06

Chapter 06 is about *Abstraction*. It is terrible coding practice to copy and paste code. If you copy and paste large chunks, you are doing something wrong. At that point, it is advisable to write general functions that accomplish the same task. Then, it is a matter of writing a one-line function call rather than the 10-20 lines of manipulation it may take to accomplish the same task over again. This is not always necessary if the operation is performed once, but certainly if done many times. It also makes it clearer to anyone reading your code. If I call some function called `normalize()`, it may not be clear what it means, but a reader can read the following:

```r
normalize <- function(df, x) {
  #' This function normalizes a variable
  #' Input: Dataframe and column vector
  #' Output: Dataframe with normalized variable appended

  x_mean <- mean(df$x)
  x_sd <- sd(df$x)

  df <- df %>%
    mutate(x_norm = (x - x_mean) / x_sd)

  return(df)
}
```

and then understand what a call to `normalize()` does.

Ch07

Chapter 07 is about *Documentation*. Documenting code helps anyone reading the code understand where certain decisions or formulas may come from. Sometimes though, you want to make your code idiot proof. This includes things the `TryExcept` statements that may pop out an error so you don't accidentally "blow up" your computer doing an infinite loop.

Ch08

Chapter 08 is about *Management*. This touches on a more meta-topic. Project management, documentation, version control, etc. is all about minimizing (or ideally removing) any ambiguity. We want to create structures that document changes, highlight which tasks need to be done versus which ones were done already... and by whom! It can be painful to figure out who made a change and where it occurred. To help remedy this problem beyond code, there exists task management software. Use it.

Git

Q5

Git is a version control system. It is a tool that manages source code history. Git**Hub** is a web-based hosting service for Git repositories. There are other hosting services for Git (e.g., BitBucket). Git is the tool whereas GitHub is the service that hosts projects utilizing the tool. GitHub is not necessary to use Git.

Q6

Since Git is a version control tool, any errors can be back-tracked or undone. If you don't use Git, you can accidentally change or delete a file and it may not be recoverable; Git fixes this problem. You can return to any previous "commit" stage in case of an error.

Q7

I do not have problems using Git as they relate to this class. Although, I would like to be able to use it solely from the command line. I'm not quite there yet.

Q8

1. Stage/Add
2. Commit

3. Pull
4. Push

These form the basic workflow for working with Git. I can make local changes (like typing the answer to this question). If I have have a notable contribution, or I want to step away but keep these changes, I can **stage** these changes to the repo history. If I am sure that I want to keep these changes, I can **commit** them to the repo. Suppose I a friend added a file I wanted to include, then I might **pull**. I can also **push** my committed local changes to the upstream remote repo.

Q9

Here is the link to the repo this file is in and here is a link to my clone of this class.

Q10

It's cloned.