Daniel Cabarcas

Assignment 2

SECTION 1

1. *Summary*

Economists are self-taught data scientists who rarely pay attention to the way they write code but often face a variety of difficulties related to the way they code and/or store data, both individually and collaboratively. When writing code, it is desirable to automate all possible processes and write a single script which executes the entirety of the code. This highlights the relevance of scripts as they allow for replicability and efficiency, which are both lost when relying on the "interactive mode". Similarly, when coding, corrections and modifications are usually made. Dealing with several versions that include changes may be challenging when datasets and code become relatively complex. The traditional "date and initial" method is a relevant possibility but it still messes with replicability as changes are made locally and could affect the way subsequent data is treated and compatibility between scripts.

Likewise, the management of data and its versions is facilitated by having a set of directories that enables the researcher to browse through data via the functions that every script and dataset has. It is also recommendable to separate files into inputs and outputs, such as using a subdirectory to convert raw data to usable data and a separate one to analyze this data. It is important to store data in table with keys that help the researcher identify the elements of a table uniquely and does not combine variables defined at different levels of aggregation. Using foreign keys is crucial as these allow to match elements from one table to another, and constitute a relational database. Data stored in this formed is considered to be normalized. Abstraction in coding language can be useful but it can also go too far. Abstracting should have a clear purpose that supports the goal of replicability and what Gentzkow and Shapiro call undo-ability. Documentation can also be useful but it has to be taken with caution as too much documentation may create issues when a command or section of code or script is not coherent with the related documentation. In this case, less can be more. Abstraction in documentation could also avoid mistakes.

2. *Why do Genztkow and Shapiro think these elements of modern empirical work are so important? What problems does each element solve?*

Gentzkow and Shapiro aim to contribute to the researcher's efficiency as ignoring the elements they propose results in lots of time wasted and many potential mistakes made that could ultimately alter a paper's results significantly. They suggest researchers should spend more time on the research problems rather than on messy details of document management and coding.

Automation saves time, simplifies code and allows for replicability. It may solve the problems that arise when new data is introduced or original data modified and code needs to adjust to these changes and most issues that arise when there is no truthful record of the precise steps that were taken.

Version control serves as the undo command for research design. It primarily solves the issues that arise when the researcher wishes to replicate or understand the workings of a code but can't keep track of the changes nor sort out which file goes with which.

Directories allow for the separation of coding into several sections that free the researcher of re running the entire data build and creates consistency between inputs and outputs, so that he doesn't need to figure out which file goes were.

Keys basically allow for data tables to be understandable and consistent, so that complex datasets can be readable and compatible with other tables. Keys solve data inconsistency.

Abstraction and documentation solves redundancy and simplify complexity in codes that might be subject to constant change. Abstraction also prevents coders the need to write new code from scratch everytime, while documentation.

Finally, management solves issues arising from poor coordination or communication in the case of collaborative work. When working alone, management serves mostly for planning but need not involve much elaborate processes.

3. ***Give an example of the sort of problem that could arise in the course of an empirical project if someone were to fail to adopt these principles.***

Some time ago I was processing some data and creating pie charts that I exported as PNG files. Some of the data corresponded to qualitative answers retrieved from an essay so that its qualitative treatment was not straightforward. Either way, pie charts were obtained and a draft of the paper was sent. A couple of months later I needed to check some of the charts and change the phrasing of titles and legends. I did not automate so I had to run the code with caution from the beginning. The way charts would be exported was not abstract in the code so that I couldn't save the files without replacing the previous charts. Finally, documentation was not in order so that I had to compare the code to the data table once again to be sure I was using the right variables and would get a coherent pie chart. It was completely inefficient and prone to mistakes.

4. ***How do you plan to incorporate these solutions into your own work?***

I plan on using version control tools such as git and GitHub and take into consideration Gentzkow and Shapiro's advice. Some previous experiences have led me to be relatively organized in terms of my directories, folders and file names when coding, but I had not given and thought to the role of simplicity and abstraction and I look forward to applying these concepts.

SECTION 2

5. ***Create a new section in the document you used to answer questions 1-4. Briefly explain what git and github are used for, how they are similar and how they are different.***

Git is a version control system which is installed directly in the computer, not the cloud, and allows for a self-contained record of changes made to your files. So git allows you to track and log the changes made to your files over time and lets you go back and review or restore older versions.
GitHub, on the other hand, is a Git repository hosting service, which serves as an online database where you can keep track, review, and share your self-contained Git projects externally (so that they are no longer contained exclusively in your computer, as opposite to Git).

6. ***Name a benefit of using git to organize your empirical research. What types of common problems can occur if you don't use git?***

Git allows you to keep track of changes and modifications made, so that every version of the project (and file) can be reviewed when needed without having to navigate the complex structure of individual files. Not using git can lead to common complications such as damaging the syntax of a code and having a hard time tracking the mistake and getting it to work again. With git you can review every version and easily located where (and when) the change that originated the problem was made. Using branches, git also prevents researchers from trying out new ideas and damaging the previous supposedly functional code by tracing the changes and keeping every version.

7. ***What about using git is challenging for you for right now? What steps can you take to minimize those challenges such that you can adopt git for this class?***

I initially thought it would be harder to understand, but I might be getting a hold of git and GitHub after exploring the interface and watching some tutorial videos. I still get confused at the steps that need to be followed when committing a new change. As of today June 10th I think working collaboratively will be a challenge when it comes to it. In an attempt to improve my understanding of the version control system, I expect to complete a series of tutorials which I found very useful. Later, I expect to incorporate git in my future research projects.

8. ***Name the four main Git operations. What does each operation do and how are is each operation different from one another?***

Stage: staging basically means telling git that you want to add changes to the history in the repository. I understand this step as "uploading" changes to git similar to attaching a document on a mail but not sending it.
Commit: Committing, which is the next step, implies actually adding those changes to the repository history, not merely "having them ready to be added" which is the case of staging.
Pull: Pulling means you get any changes that had not been previously "downloaded"
Push: After you commit and changes are already added, you push changes so that these changes are effective on the files within the repository.