**ASSIGNMENT 2.Susette España**

1. Summarize briefly the point of chapters 2- 8 in less than one page.
2. Why do Genztkow and Shapiro think these elements of modern empirical work are so important? What problems does each element solve?
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.
4. How do you plan to incorporate these solutions into your own work?
5. **Gentzkow and Shapiro wrote a handbook for economists on how to organize their coding and data work**. Most economists rarely take formal courses on data science and learn/improvise/implement on the go; they themselves struggled with it and this is akin to the metaphor Scott told us about doing our work as if our present-self was the RA of our future-self. The 8 chapters and appendix summarize practical guidelines on how to establish our coding/data workflow. **In chapter 2, we should begin with automating all possible steps** because if we use the alternative, an “interactive” mode of working we will have difficulty giving replicability to our work. And to gain efficiency we should implement a key script to the directory we will create to avoid human errors of having to type manually what can be automated. **The 3rd chapter covers the version control tools; we want to be able to guarantee replicability and undoing of steps easily**. Using the date and initials methods lends itself for error and confusion, we should instead adopt a version control method for our coding and data. We set up “repositories” as we did in Git Hub where we can easily retrieve previous versions and keep track of what elements were changed and why. **In chapter 4, they discuss a way to manage our data so that we can establish blocks or work (directories).** While you still should have one ‘key directory’ as pointed above, you also want to have lower level directories where you can try out certain functionalities without having to run the whole directory each time. **In Chapter 5, it seems to solve out for the problem of having missing fields or duplicates in the data set**. So, setting up each table with one key solves these problems and makes easier working with the data set generated, for example, by a RA. They also advise with working with normalized data as far in the process as possible, so that our data is far easier to understand while minimizing expensive mistakes. **Then in Chapter 6, they make the case for abstracting, except when it is not necessary!** A common practice software engineers apply, and we should be doing “unit testing”, running a script to make sure the added function does what it is supposed to do. Abstracting means to turn specific elements into a tool for general purpose. This eliminates redundancy and makes it more readable. Abstracting is a far superior mechanism than the “copy-paste” one where we can introduce manual errors and, if we abstract it well, we can use it for other projects as well**! Chapter 7 makes the case for less documenting but better!** A good rule of thumb of when to documenting makes sense is if it is worth maintaining updated and/if we have no practical way of documenting it in the code itself. As they write “nothing documents code quite like code”. **Finally, in Chapter 8,** **they recommend investing time in setting up an efficient task management system** to know who oversees what, for when and how and can be easily checked off from the to-do list. And then also have a system to scribble other areas of the project (figures, ideas, notes) and make them easily available to your collaborators.
6. These protocols are important to have well-organized workflow where we can easily spot and correct errors; and coordinate with large teams. Thinking about the course and one of the stories Scott told un in class, implementing these principles could’ve help him spot the typing error he put into his work before submitting his final work. Also, it could have saved him a lot of time to figure out what it was he did back then and why. Instead of going through piles and piles and piles of data re-learning what it was he did and why!

ELEMENTS:

1**. Automation**: solves for replicability and provides efficiency. You usually end up running every step multiple times and the cost of manually doing it adds up quickly!

**2.Version control:** solves for easily undoing steps and replicability, because you will easily know which files goes with which one

**3.Directories:** solves for accessibility of one data file to use in multiple projects without having to create redundant copies and by separating low level and high level directories can help you identify easier where you made a mistake, reducing the time of debugging

**4. Keys:** solves for the problem of having missing values or duplicated across rows. Having one key for each table guarantees this.

**5. Abstraction:** solvers for redundancy too, minimizing the scope of error and makes your code more readable and to use for other general-purpose projects. If you do good abstraction work, you can implement the functionality in another project.

**6. Documentation**: solves for having mismatched information on the documentation from the coding. The preferred way is to have coding that is self-documenting and only use documenting when you can’t easily put it into the code.

**7. Management:** solves for work lagging and lack of clarity of the desired output.

1. **A problem that might arise if not following the principles can be, for example, with documenting**. You can have the problem that the documenting information does not correspond with the coding information and then you have the problem of now knowing they it was changed in the coding -or if it was changed by mistake. In the handbook the authors speak of how on their document part they the elasticity had (2) but in the coding it was (3). Why and who decided to change the table from where the elasticity was pull from was unclear. So, in this case, it is better to have less in the documenting area, and the coding itself should incorporate the documenting to make sure the applied elasticity is the corresponding one.
2. This handbook was eye-opening for me and very practical hands on. I will work on trying to implement the most principles where human errors are minimized (either by typing, manually updating things, copy-pasting, and where human memory fails after a few weeks!). Beginning with this course, **I’ll work on keeping my GitHub work aligned with these principles. Also, I think Scott’s advice on investing time on learning an automating tool as Latex seems even more compelling after this book!**
3. Briefly explain what git and github are used for, how they are similar and how they are different.

GitHub is an online plataform which provides several services, and it is hosted within the Git system. You can use Git without GitHub, but using GitHub simplifies your work. In Git you do version control and provides great help with replicability requirements. Many popular apps were in fact built using Git tools.

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

Version control: you can know who made what changed when and why.

You can run into issues of replicability, redundancy, changing one part of the code can mess up the whole program

1. 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 have not yet learned coding. I began and will continue using the “how-to-guides” incorporated inside GitHub and the web. I successfully created my repositories and cloned Scott’s class and I think the more I toy around with it, the faster the intuition of it I will develop :)

1. Name the four main Git operations. What does each operation do and how are is each operation different from one another? Copy-paste from the GitHack document:

*1. Stage (or "add"): Tell Git that you want to add changes to the repo history (file edits, additions, deletions, etc.)*

*2. Commit: Tell Git that, yes, you are sure these changes should be part of the repo history.*

*3. Pull: Get any new changes made on the GitHub repo (i.e. the upstream remote), either by your collaborators or you on another machine.*

*4. Push: Push any (commited) local changes to the GitHub repo*

My added commentary: usually the first two are used together (Stage and Commit); and the second two together (pull and push). With stage and commit the idea is your “stage” the change you want to incorporate, and commit Is the key to make sure you want to make the change.

With pull and push, you first pull the repository in case someone is simultaneously working on it and then with push you incorporate your own changes into it.

**9-11 covered and implemented in GitHub and Slack.**

1. The first step in your new empirical workflow is the creation of a Github repository (“repo”). You can either do this independently or do this through R functionality. You need to create a github account, then create your first repository called “Titanic”. Initialize with a Readme and create the separate folders that we discussed in class on Monday.
2. Post a link to your repository
3. Please clone our course github repository on your desktop