Week 1: Tools

There are two versions of this file: an html and a pdf. If you need to print the document, then the pdf works a little better.

If you find a problem with this assignment, then please let me know, especially if the instructions don't work or you find them confusing. **This is important.** This assignment can be particularly frustrating, so if the instructions are unclear, inaccurate, or incomplete, let me know so that I can update them and save others some frustration.

In this homework assignment, you will install a bunch of software, create a couple of accounts, and make sure things work. You don't need to understand everything that's happening. That's expected and okay. You'll learn more about these tools as we move through the semester.

A Warning

For the conceptual homework, you need to read several things to read complete a few exercises. For this homework, we're trying to work through many simple ideas, so keep your work concise–only dwell on ideas that you find unfamiliar or challenging. Keep your answers brief, especially for ideas that you find easy or those that are already familiar.

The computational assignment is tedious, but not difficult. This is an especially tedious assignment, so **please start it early**. This will give you several opportunities get stuck and seek help.

Conceptual Homework

The readings this week give us enough background in research design to proceed with data analysis (while you continue to learn research design in 5736).

Read the following:

- syllabus

Exercises

1. ~~Complete the exercises at the end of the notes on questions.~~ (We didn't get to this in class, so skip for now.)
2. ~~Complete the exercises at the end of the slide on causality.~~ (We didn't get to this in class, so skip for now.)

What Does a Data Set Look Like?

- Wooldridge. 2016. *Introductory Econometrics: A Modern Approach.* 6th Edition. Read ch. 1, pp. 1-17. "The Nature of Econometrics and Econometric Data."

This chapter gives us some basic terms to help us think about data (e.g., "cross-sectional," "panel," "observational").

Exercises

1. Wooldridge discusses four data structures: cross-sectional, time-series, pooled cross-sectional, and panel. Describe each and contrast.
2. Describe what Wooldridge means by *ceteris paribus*.
3. Problem 2 on p. 15 (end of ch. 1).

- Broman and Woo. 2018. "Data Organization in Spreadsheets." [journal].

This is an **important** article for us. It describes how to organize your data, which helps us in two ways. First, if you collect your own data, it gives you good advice for structuring your spreadsheet. Second, if we're "just" cleaning up raw data into a usable form, this gives us a goal. Given that we want to share the data we use, **we want our data in this form**. Throughout this class, I'm going to insist that we follow their advice–so become familiar with it.

Exercise Broman and Woo give several rules. List the *three* that were most unexpected and briefly summarize why they might be important.

What Does a Research Design Look Like?

- FPP. Read chs. 1-2, pp. 3-28.

Exercises Do the assigned exercises FPP (only those from chs. 1-2). Answer briefly–these questions don't require an in-depth discussion.

Is Simple Description Useful?

- Gerring. 2012. "Mere Description." [journal].

Exercises

1. Gerring gives a taxonomy of descriptive arguments. Reproduce this taxonomy and describe each archetypal form. From your seminar readings, can you find examples of descriptive claims/arguments that fall any of the categories?
2. What is your assessment of the argument that description is valuable as an end to itself? What about as a prerequisite for causal arguments?

How Can Well Make Our Computational Research Credible?

- Bryan. 2018. "Excuse Me, Do You Have a Moment to Talk About Version Control?" [pdf]

Exercise For computational research, we have manually-edited spreadsheet cells and point-and-click analysis on one extreme. On the other extreme, we have computer scripts that manipulate the raw data into the tables and figures. What is the advantage of working near the latter extreme? How does git and GitHub facility this work?

Computational Assignment

Part 1: Getting Your Tools Installed

Most social scientists are *computational* scientists in the sense that (1) they use computers to perform relevant computations and (2) share their code and data with others. Here are the tools I use (with macOS, see the

italics for Windows alternatives where needed):

- **RStudio** for data management and manipulation (e.g., tidyverse), plotting (e.g., ggplot2), and modeling (e.g., rstan).
- LaTeX through **TeXShop** for documents (*TeXstudio for Windows*), **BibDesk** for bibliographies, (*JabRef for Windows*), and **LaTeXiT** for single equations (*KLatexFormula for Windows*).
- **R Markdown** for simple reports (e.g., peer reviews)
- git and GitHub through **GitHub Desktop** for version control.
- **GNU Make** for defining dependencies and reproducing projects.

I feel pretty confident that these are the best tools for macOS, but you might want to consult with more senior graduate students for advice about Windows.

Note: In Part 1, you install a bunch of software and start your first GitHub repository. Keep using this same repository in Parts 2 and 3.

Ask a Question

At some point in this assignment, find something that you don't completely understand. Come to my office and talk to me about it!

You can consider this part of your computational homework for the week.

GitHub

Register a GitHub account. Read the advice from Jenny Bryan about choosing a username and then use your .edu e-mail address to sign up for an account on github.com. You can associate multiple e-mail accounts with GitHub, so feel free to add in the primary account that you have permanent control over (e.g., your gmail account).

Apply for a Student Developer Pack. This gives you private repos for free.

Follow me on GitHub. I'm carlislerainey. Follow harleyroe and qwang1015, our TAs, as well. Once you follow me, I'll add you to our POS 5737 private GitHub Organization.

Browse some repos on GitHub. See an ongoing project of mine of mine, a finished project of mine, and Josh Alley's cool dissertation project (that I made a small contribution to). Remember that most GitHub repos contain software, not research projects. For examples, see the Linux OS, my website, and the ggplot2 R package.

Set Up Your Computer for Computing

To make things a little easier throughout the process, we want to do four things:

1. Show hidden files.
2. Show file extensions.
3. Make it easy to open a terminal at a folder.
4. Make your terminal look nice (on macOS).

Show Hidden Files Typically, files that start with `.`, like `.gitignore`, are hidden from the user's view. Hidden files, like `.DS_Store` and `.git`, are hidden for a reason, but you'll sometimes want to access them, especially `.gitignore`.

On Windows, see the instructions here.

On macOS, simply press `command + shift + .` to show these hidden files. This shows hidden files until you restart finder (e.g., log off, restart your computer, etc). If you don't want to always see hidden files, this offers a convenient toggle.

If you always want to see hidden files, then you can run `defaults write com.apple.finder AppleShowAllFiles YES` in a terminal. You can still use the toggle, but Finder now defaults to showing hidden files.

Show File Extensions For reasons I don't understand, neither macOS nor Windows shows file extensions by default. (That's the `.docx` bit of the Word file `Essay for English 1101.docx`.)

On Windows, see the instructions here.

On macOS, open Finder. Click *Finder* > *Preferences...* Select *Advanced* tab and check the box to *Show all filename extensions.*

Enable the "New Terminal at Folder" on macOS **On Windows** 7/8/10, you don't need to do anything. When you need a command line at a folder, simply hold down the shift key and right-click a folder. The context menu contains an entry, *Open command window here.* (Later, once you've installed Git for Windows, you'll have the option to open a Git Bash terminal as well.)

On macOS, make it easy to open a terminal in a specific directory by enabling *New Terminal at Folder.* Open Finder. Click *Finder* > *Services* > *Services Preferences....* In the right box, navigate past *Pictures*, *Messaging*, and *Development* to *Files and Folders.* Check the box for *New Terminal at Folder.* Now right-click on a folder and you should see the option (near the bottom) to open a new terminal at that folder.

Customize Your Terminal Prompt on macOS This is for **macOS only**.

The command prompt on macOS doesn't look nice by default.

You probably don't have the file `~/.bash_profile`, but you need to check. Open a terminal and run the command `open -a TextEdit ~/.bash_profile`. If you don't have it, you'll get a message saying so. If you already have it, just add the line `export PS1="\W > "` to the file on a new, separate line.

If you don't have it, then you need to create it. Run `echo 'export PS1="\W > "' > ~/.bash_profile`. Open a new terminal. The prompt should look nicer.

R and RStudio

Install (or Update) R. Choose the appropriate OS from CRAN and follow the instructions. On Windows, you get two versions of R: i386 and x64. This is normal. These are 32- and 64-bit versions We only use R indirectly through RStudio, so we never choose between the two. (But both work!)

Install (or Update) RStudio. You may choose either the preview version or the latest stable version. *I use the preview version.* ~~The lab computers have the latest stable version.~~ [edit: The lab is currently under construction.] The preview version has new features; the latest stable version is (I suppose) more robust. Choose a version, select your OS, and follow the instructions.

If you already have R installed, **update your packages**. Just open R or RStudio and run `update.packages(ask = FALSE, checkBuilt = TRUE)`.

Adjust One (Bad) Default. Click *Tools* > *Global Options...*. Select *General*. Under *Workspace*, set *Save workspace to .RData on exit:* to *Never*. Uncheck the box for *Restore .RData into workspace at startup.*

Install the tidyverse package. In the lower-right pane, click the *Packages* tab to show the *Packages* window. You see a list of available packages. Click the *Install* button at the top of the *Packages* window, type "tidyverse" in the middle box, and click *Install*. Make sure that tidyverse installs successfully by entering the command `library(tidyverse)` in the console in the lower-left pane.

Comment: Other instructors might suggest the equivalent approach of entering the command `install.packages("tidyverse")` in the console, but I recommend the point-and-click method. It's easier and packages only need to be installed once per computer.

Comment: R packages allow software developers to distribute additional functionality to users. For example, you'll use the ggplot2 package to create graphs. Hadley Wickham wrote ggplot2 as part of his dissertation in the statistics department at Iowa State. That was version 0.0.7 and we're now on version 3.0.0. I'm excited for the next release because of this annoying bug.

Comment: When you run `library(tidyverse)`, you get the output below, which is both expected and desirable.

```
library(tidyverse)
```

Run an R script. To start a new R script, open RStudio and click *File* > *New File* > *R Script*. This opens a blank R script in the upper-left pane. Copy-and-paste the code below into the R script. Use your mouse to highlight all eight lines (or press command + a) and then click *Run* (or press command + enter) at the upper-right of the script window. Make sure a scatterplot appears in the *Plot* tab of the lower-right pane. If so, you've got R and RStudio working.

```
# simple example script

# load packages
library(tidyverse)

# create two vectors of data
x <- c(2, 3, 5, 4, 1)  # x "gets" a "collection" of 5 numbers
y <- c(4, 2, 1, 5, 6)  # y "gets" a "collection" of 5 numbers

# plot x and y
qplot(x, y)
```

Text Editor

If you already have a favorite, that's fine. Use it.

As RStudio develops, I find that I need another text editor rarely, if ever. In case, here are the one's I recommend.

For **Windows**, I recommend downloading Sublime Text 3. Follow the instructions.

For **macOS**, I recommend downloading Atom. Follow the instructions.

Git

Setup Git **Install git. For macOS**, open a terminal and run `xcode-select --install`. This installs Xcode command line tools, which includes git. (Note that you can also install all of Xcode, but all of Xcode is a much larger installation. You only need Xcode command line tools.) **For Windows**, download Git for Windows and follow the instructions.

Install GitHub Desktop. Download GitHub Desktop and follow the instructions.

Comment: There are three ways you can interact with git: (1) command line, (2) RStudio, and (3) a client like GitHub Desktop. RStudio gives you an easy, but limited, way to interact with git. The command line is complete, but complex. **GitHub Desktop is the best of both worlds.** It's complete and easy-to-use.

Introduce yourself to git. (Only do this on your personal computer.) At some point, you'll want to use git from the terminal/Git Bash. Go ahead and let git know who you are. In a terminal (**macOS**) or Git Bash (**Windows**), run `git config --global user.name 'Jane Doe'` and `git config --global user.email 'jdoe@fsu.edu'`–be sure to use the e-mail associated with your GitHub account. Run `git config --global --list` to verify the settings.

Associate your text editor with git. For **macOS**, run `git config --global core.editor "atom --wait"` in a terminal. For **Windows**, run `git config --global core.editor "'c:/program files/sublime text 3/subl.exe' -w"` in Git Bash.

Comment: When using git from the command line, it sometimes pops open a text editor. By default, git uses vim. Vim is a strange, scary place to wind up. If you find yourself in vim, you might never escape. Avoid that possibility entirely by associating a friendly text editor with git.

Use Git with GitHub There are three similarly-named things here:

1. **git**: that's the underlying version-control software.
2. **GitHub Desktop**: that's the visual interface that you use to interact with git.
3. **GitHub**: that's the website that allows you to easily browse repos' histories and share work.

Start a Repo with GitHub Desktop Open GitHub Desktop. Click *File*, *New Repository...*. Name the repo `test-github` and set the local path to your computer's desktop. Initialize the repo with a README. Click the blue *Create repository* button.

This first repo is just for practice–you'll delete it later.

Click the button *Publish repository* at the top. Supply your GitHub username and password. Again, lick the button *Publish repository*. (Note that the local directory and the GitHub repo do not necessarily share their name, but I recommend you keep them the same.) Click the blue *Publish repository* button.

Comment: In my work, I associate one repo with a single research project (i.e., usually one manuscript). See my repos. Until later in the semester, I'll instruct you to create a separate repo for each homework assignment (i.e., a new repo each week). You'll create another repo for the research project.

Change-Review-Commit-Push When students first start to develop projects with git and GitHub, I recommend a change-review-commit-push model. Choose a manageable change you'd like to make to the project (like "Rewrite the introduction"), then make, review, commit, and push the change. Git and GitHub are extremely complicated, but powerful tools. Don't jump all in on git and GitHub immediately (and maybe not ever). Keep it simple for a while with change-review-commit-push.

First, **edit the file `README.md` locally**. You should find a new folder `test-github` on the Desktop. Open the file `README.md` with Atom (**macOS**), Sublime Text 3 (**Windows**), or RStudio (either OS). Make some change to the file and save it.

Comment: GitHub treats the file `README.md` specially–it displays it on the repo's main page. I have a detailed `README.md` for my latent-dissent project, for example. Because `README.md` is a Markdown file, you can use Markdown syntax to style the document.

Second, **review your change**. Go back to GitHub Desktop, which shows you a list of changed files. This list should include `README.md` because you changed it. This list should *only* include `README.md` because it's the only file you changed (it's also the only file in the repo). If you click the entry for `README.md`, you can see the changes you made to that file.

Third, **commit your change**. (Remember, a commit is like a snapshot of your files.) Once you're happy with the changes, you want to commit those changes. Ideally, a commit is a three step process:

1. *Stage the files.* Check the boxes next to the changed files you want to commit.
2. *Write a commit message.* Type a short description of your change in the *Summary* box and a longer description in the *Description* box. This commit message reminds you what you did and why. I like Chris Beam's discussion of a good commit message.
3. *Commit the change.* Click the blue *Commit to master* button to commit the changes.

Comment: Committing is like taking a snapshot of the directory. You already have one snapshot from when you initially created the repo on GitHub. When you made a change to `README.md`, you took another snapshot. This allows you to return to these exact files later, as needed. For an example, look at the commits for my recent paper in *Political Analysis*.

Fourth, **push these changes up to GitHub**. Click *Push origin* in upper-right corner. This simply passes the update along to GitHub, which safely stores it for backup, browsing, and sharing.

Comment: You've got three things going on now–keep them separate in your mind. You have the *local files* that you see on your computer. You have a *local history* of the project (each commit) stored in the file `.git`. You have a *copy* of the `.git` file hosted on GitHub for safe storage and others to access.

Comment: If multiple users are pushing commits to the same repo on GitHub, then you need a workflow that pulls changes down from GitHub in addition to pushing them up. There are lots of options, but you need a strategy. It should involve pulling down and pushing up changes often.

Lastly, **confirm the local change propagated to the GitHub remote.** In a web browser, navigate to your repo on GitHub. See that the changes you made to the README are there.

Do It Again Make another change. Make a local change and save it, then stage it, commit it, and push it. Maybe try adding a file.

Delete. When you are done, delete your local directory and the GitHub repo. You can do this from GitHub Desktop by clicking *Repository* > *Remove*. Check the box *Also move to trash* and click *Remove*. This stops GitHub Desktop from monitoring the local directory and moves the local directory to the trash.

If you used a community computer, log out of GitHub Desktop when you are done. Be sure to both sign out of your GitHub.com account and remove your *Name* and *Email* from the Git tab. This keeps others from making changes as you (bad) and pushing to GitHub as you (worse).

Avoid typing your username and password every time.

(Only do this on your personal computer!). Follow these instructions.

LaTeX

Install LaTeX For **macOS**, I recommend MacTeX. Download and follow the instructions. It includes the following:

1. TeXShop: editor, used to write LaTeX files.
2. BibDesk: reference manager, used to manage bibliography databases.
3. LaTeXiT: used to create images of equations.

For **Windows**, I recommend MiKTeX, TeXstudio, and JabRef. Download and follow the instructions. Hopefully, the installer asks you if you would like MiKTeX to install missing packages on-the-fly. Choose *Yes* rather than *Ask Me*. (See here for the details). In this setup, you have the following:

1. TeXstudio: editor, used to write LaTeX files.
2. JabRef: reference manager, used to manage bibliography databases.

Check LaTeX from RStudio Open RStudio. Click *File* > *New File* > *Text File*. Save as `latex-test.tex`. Create a new folder on the Desktop and save it there. Copy-and-paste the LaTeX code below into `latex-test.tex`. Click *Compile PDF*. Check that this creates the pdf you expect. Delete the pdf file.

```
% a minimal latex document
\documentclass{article}

\begin{document}
First document. If this compiles into a pdf, then LaTeX seems to work.
\end{document}
```

Check LaTeX from Your Editor Open `latex-test.tex` with your preferred editor. Right-click on `latex-test.tex`, select *Open with*, and choose your editor TeXShop (**macOS**) or TeXstudio (**Windows**). In TeXShop, click the *Typeset* button in the upper-left corner to compile into a pdf. In TeXstudio, click the green play button. Check that this creates the pdf you expect. Delete the pdf file.

At times, you might find it more convenient to develop `.tex` files from a dedicated editor like TeXShop for its features. Other times, you might choose to develop `.tex` files in RStudio to use a single interface.

R Markdown

Start by fixing one bad default. Open RStudio. Click *Tools* > *Global Options...*. Select the *R Markdown* tab on the left. Near the middle, change *Evaluate chunks in directory:* to *Project*. (This makes R Markdown document code chunks behave the same way as R code run in the terminal.)

Now try out R Markdown. Open RStudio. Click *File* > *New File* > *R Markdown*. Install any suggested packages. Fill in the *Title* and *Author* boxes however you like. Make sure the *HTML* bubble is selected (the default). This initiates an R Markdown document that serves as a helpful template.

Notice that the R Markdown file contains a mix of R code and Markdown syntax. Don't sweat the details.

Save the file as `rmarkdown-test.Rmd` to the Desktop.

Click the *Knit* button to knit the `.Rmd` into an `.html` file, which you can view with a web browser.

Check that you can also knit to a pdf (and maybe Word) document by clicking the tiny triangle next to the Knit button and then clicking *Knit to PDF* or *Knit to Word*. (If you don't have Word, this won't work. Don't worry about it. You should be able to knit to pdf, because you have LaTeX working.)

Sometimes, **on Windows**, R Markdown and MiKTeX will not work together properly. If R Markdown fails to generate the pdf, then let MiKTeX install missing LaTeX packages automatically. This might fix the problem.

When you are sure that R Markdown works for you, delete `rmarkdown-test.Rmd` from the Desktop and all the associated files that knitr generated.

Test a Complete Workflow

The steps below make up your graded homework for the week and allow you to see how we use R, LaTeX, GNU Make, and git to form a coherent, reliable, reproducible workflow.

Start New Project Open RStudio. Click *File > New Project*. Then click *New Directory* and *New Project*. Check the boxes for *Create git repository* and *Open in new session*. Name the directory `hw01`. Create the project as a subdirectory of wherever you like on your computer (perhaps as a subdirectory of `pos5737/homework/` for example).

You might find it helpful to review this summary of R projects to understand their role.

Make Your First Commit Open `hw01` in GitHub Desktop by clicking *File > Add local repository* and selecting `hw01`. You've just initialized everything, so type "Initialize project" in the summary. Stage both `.gitignore` and `hw01.Rproj` by checking the boxes next to those files. Commit these changes by clicking the blue *Commit to master* button near the bottom. That's your first commit. Yay!

Create a GitHub Repo Click the *Publish to GitHub* button in the upper-right corner. In the *Name* box, type `hw01-jane-doe`, replacing `jane` with your first name and `doe` with your last name. Check the box to *Keep this code private.* For organization, choose pos5737. (Note that if I haven't added you to the pos5737 organization on GitHub, you won't have this option yet. If that's so, then please wait until I add you–perhaps send me a reminder on Slack.) Click the blue *Publish Repository* button.

You've just created a repo on GitHub and pushed your local files there. You should be able to find your two files on GitHub now. Yay!

Write an R Script In RStudio, with the homework project open, click *File > New File > R Script*. Copy-and-paste the script below into the file and click the save button. Save the file as `create-plot.R` in `hw01`. **File names must be exactly right.** Run the code (in pieces if you want to see how it works).

```
# create-plot.R: minimal R code to make a plot

# load packages
library(tidyverse)

# create variables x and y
x <- c(1, 2, 4, 6, 3)
y <- c(6, 2, 3, 1, 4)

# plot x and y
qplot(x, y)

# save plot in png format
ggsave("plot.png", height = 3, width = 4)
```

Remember that our workflow is change-review-commit-push. You just made a nice **change**, so open GitHub Desktop back up and **review-commit-push**. Include a nice commit message.

Write a LaTeX Document with a Figure In RStudio, with the homework project open, click *File* > *New File* > then *Text File*. Copy-and-paste the LaTeX document below into the file and click the save button. Save the file as `doc.tex` to `hw01`.

```
% doc.tex: a code to include a plot

\documentclass{article}
\usepackage{graphicx} % useful for including graphics

\begin{document}

You should see the figure below.

% add the plot
\includegraphics[width = 4in]{plot.png}

\end{document}
```

You can press the *Compile PDF* button if you like. If you ran the file `create-plot.R` in the previous step, it should compile into a pdf with a scatterplot.

Remember that our workflow is change-review-commit-push. You just made a nice change, so open GitHub Desktop back up and review-commit-push. Include a nice commit message.

Do Something Interesting Do something interesting with the R script. Use Google to find something cool. Find an interesting R package and use a simple example from the help file. It doesn't need to be fancy, just try something. Feel free to borrow heavily from someone else's work you find online, just explain where you borrowed from and explain what's happening using comments in the R code and prose in the LaTeX document (as best you understand it).

I should be clear that I assume you know nothing about R code at this point. You should just search for some examples on Google, try it, and hope it works. You might not understand what's happening. I just want you to experiment a bit.

Hint: Look through some of the geoms on the ggplot2 webpage (e.g., `geom_point()`). Try to run some of the example code there.

Or try this site.

If you break something beyond repair, you can (permanently) go back to your most recent commit. Close any open files in the project, open GitHub Desktop, right-click a file you've changed, and click *Discard All Changes...* (If it all goes to heck, then hop on Slack and I can help you out.)

Once you've done something interesting, clean and rebuild a final time to make sure it works properly. That's the change. Now just review-commit-push.

Find your repo on GitHub and check that your files appear as you expect. Click the *commits* tab. Browse the previous versions of the project.

Part 2: Create a Data Set

Make sure you read Broman and Woo (2018) before staring this portion. **You must follow the advice of Broman and Woo (2018).** Make sure you create both the data set and the data dictionary (perhaps as separate sheets in the same file).

Create a simple data set using a spreadsheet program (like Microsoft Excel or Google Sheets). You may name the file what you like, but short, descriptive names work best.

Meet as many of the following criteria as you can.

1. Collect a data set with about about 5-10 observations (rows) and 4-5 variables (columns). You can have more if you like, but I want you to have data set that allows you to easily inspect every row and column. There's no need to have more data than this 5 rows and 4 columns.
2. Collect several types of variables. Numbers, dates, dichotomous variables, character strings, etc.
3. Collect a data set that falls within your interests. For example, if you're interested in comparative electoral institutions, perhaps collect data on the electoral rules in South American countries, such as `country_name` (character), `number_of_electoral_districts` (numeric), `legislature_size` (numeric), `elected_executive` (Yes/No), and `date_regime_established` (date). If I were doing this exercise, I would use Wikipedia as the source. You could also do US states with `state_name` (character), `percent_uninsured` (numeric), `expanded_medicaid` (Yes/No), and `date_expanded_medicaid` (date).

For this exercise, I want you to learn (by doing) what a **well-formatted** data set looks like. Keep that goal in mind.

If you use Excel, then save the `.xlsx` file to `hw01`. Remember to change-review-commit-push as you work. Unfortunately, GitHub doesn't render `.xlsx` files well, so it's not easy to explore the history, but at least you have it.

- **change**: Update your spreadsheet. Make a little progress, so that you feel like you've completed a discrete "part" (of course this is subjective) of the assignment (e.g., "Add country_name").
- **review**: double-check your progress to make sure that (1) you've done what you think and (2) haven't done anything else. For some files, it's easy to see the changes in GitHub Desktop.
- **commit**: Stage any changed files by checking the boxes next to those files. Write a clear commit message in the summer. Commit these changes by clicking the blue *Commit to master* button near the bottom.
- **push**: Click the button *Push origin* at the top of GitHub Desktop. This sends your latest work up to GitHub for you and others to explore.

If you use Google Sheets, then click the *Share* button on the top-right and then click *Get shareable link*. Copy that link to the Google Sheet and put somewhere easily findable in your project (`README.md` is the best choice, IMO). You can't use git and GitHub for version control with Google Sheets, but you can click *File* > *Version history* > *Name current version* to take snapshots of your sheet at particular points. You can go back to any named version later.

When you've completed the data set and data dictionary, export a `.csv` file of the data set and data dictionary and save them to `hw01`. Again, name the files well. This is an important **change**, so definitely **review-commit-push** at this point.

Part 3: Load a Data Set

Find a raw data set that is common in your field.

- If you are interested in American politics, you can use the ANES or the Correlates of State Policy.
- In international relations, the Correlates of War Project or the International Political Economy Data Resource.
- If comparative politics, then the Manifesto Project, Democratic Electoral Systems, or Democracy and Dictatorship.

The data sets above are only examples. However, it's helpful to a data set that is (1) common in the field, (2) you will use in the future, and (3) other people in the class are using. I'll ask you to apply some of the tools we learn later this semester to this data set.

You might find it helpful to review this chapter from the notes before/as you work.

1. Download the raw data set(s) from the internet. Do not change anything about these raw data sets, including the filename. Write protect the data set(s) and place them in a special folder (I recommend

`hw01/data/raw/`). Add a thorough note to the `README.md` describing *exactly* how and when you downloaded the data set(s).

2. If your data set is enormous, then GitHub Desktop will warn you that it's too big when you try to commit it. If you get this warning, please DO NOT commit this data set. I can help you come up with an alternative procedure.
3. As best you can, write a script to read these data sets into R. `You may adjust the default arguments in the function that reads the data set, but do not alter the data set (yet) after loading it into R. Do not manually "fix" the raw data. It might read into R as total garbage--that's okay. If you want, inspect the resulting data frames using`summary(),tidyverse::glimpse(), `and/or`skimr::skim()‘. Save this script (use good filenames!) and review-commit-push.

Overiew

You have three things due by Sunday night.

1. The conceptual homework. Please turn in at the beginning of class on Monday. Hand-written is fine, but please make it readable.
2. The computational homework. You submit a link to your GH repo on Canvas.
3. The reflection.

I've restructured the course a bit, so feel free to share ways to improve the submission process.