

# Sociospatial Data Science

*Christopher Prener, Ph.D.*

*2018-01-05*



# Contents

<b>Preface</b>	<b>5</b>
License . . . . .	6
<b>1 Introduction</b>	<b>7</b>
1.1 Opinionated Processes . . . . .	7
1.2 What is data science? . . . . .	8
1.3 How is this book organized? . . . . .	9
 <b>I First Steps</b>	 <b>13</b>
<b>2 Approaching These Courses</b>	<b>15</b>
2.1 Zen and the Art of Data Analysis . . . . .	15
2.2 Getting and Staying Organized . . . . .	15
2.3 Course Flow . . . . .	17
2.4 Staying Current . . . . .	18
<b>3 “Good Enough” Research Practices</b>	<b>23</b>
3.1 Reproducibility . . . . .	23
3.2 Thinking in Workflows . . . . .	24
3.3 Data . . . . .	25
3.4 Plain Text . . . . .	26
3.5 Version Control . . . . .	26
<b>4 Protecting Your Work</b>	<b>27</b>
4.1 Threats To Our Data . . . . .	27
4.2 Creating a Sustainable File System . . . . .	28
4.3 Organizing Projects . . . . .	31
4.4 Backing Up Your Data . . . . .	33
<b>5 Getting Help</b>	<b>37</b>
5.1 Helping Yourself . . . . .	38
5.2 How to Seek Help . . . . .	40
5.3 Where to Seek Help . . . . .	44
 <b>II Data Science Toolkit</b>	 <b>47</b>
<b>6 Opinionated Tools</b>	<b>49</b>
<b>7 Markdown</b>	<b>51</b>
7.1 Document Organization . . . . .	51
7.2 Working with Text . . . . .	53

7.3	GitHub Markdown . . . . .	54
<b>8</b>	<b>Basic Git</b>	<b>57</b>
8.1	Git Basics . . . . .	57
8.2	The Workflow of Git and GitHub . . . . .	58
8.3	GitHub Issues . . . . .	60
8.4	GitHub Desktop Application . . . . .	61
8.5	Learning More . . . . .	61
<b>9</b>	<b>Advanced Git</b>	<b>63</b>
9.1	Even More Git-lingo . . . . .	63
9.2	Creating Branches . . . . .	63
9.3	Git Terminal . . . . .	68

# Preface



This text is a companion text for both of my research methods courses at Saint Louis University:

- SOC 4650/5650 - Introduction to Geographic Information Science
- SOC 4930/5050 - Quantitative Analysis: Applied Inferential Statistics

The goal of the text is to create a reference for the intangible, subtle or disparate skills and ideas that contribute to being a successful *computational* social scientist. In writing this text, I draw inspiration from the work of Donald Knuth.<sup>1</sup> Knuth has discussed his experiences in designing new software languages, nothing that the developer of a new language

...must not only be the implementer and the first large-scale user; the designer should also write the first user manual... If I had not participated fully in all these activities, literally hundreds of improvements would never have been made, because I would never have thought of them or perceived why they were important...

While there is nothing particularly new about what I am writing here, and I am certainly not developing a new language for computing, the goal of this text remains similar to Knuth's experience. By distilling some of key elements for making a successful transition to being a *professional developer* of knowledge rather than a *casual consumer*, I hope to both improve the course experience itself and also create an environment that fosters a successful learning experience for you.

In both classes, the course names are deceptive. We are not only concerned with statistical work or mapping. Rather, we are more fundamentally concerned with research methods. In particular, we are concerned with *high quality* research methods and the *process* of conducting research. We therefore focus on a combination of mental habits and technical practices that make you a successful researcher. Some of the skills and techniques

---

<sup>1</sup>Donald Knuth is the developer of TeX, a computer typesetting system that is widely used today for scientific publishing in the form of LaTeX. He also established the concept of literate programming, which forms the basis of some of the practices we follow with R.

that we will discuss this semester are not taught as often in graduate programs, let alone undergraduate programs. Instead, they are often the products of “learning the hard way”. These “habits of mind and habits of method” are broadly applicable across methodologies and disciplines.

## License

Copyright © 2016-2018 Christopher G. Prener

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

# Chapter 1

## Introduction

The first part of this text is designed to help get you oriented to coursework in computational social science. Both Introduction to Geographic Information Science (SOC 4650/5650) and Quantitative Analysis: Applied Inferential Statistics (SOC 4930/5050) are focused on building students' capacities to address social science research questions using tools that have been the traditional domain of computer and information scientists. The growth and application of these tools in a variety of disciplines both inside and outside of the social sciences has come to be known as data science. Both courses are taught from this perspective, so that while we focus on social science data, the tools and techniques are broadly applicable across disciplines.

### 1.1 Opinionated Processes

This text, and both of the courses it is written with an eye towards, is meant to be “opinionated”. We'll introduce a high level definition of opinionated process here, and discuss how these ideas might fit into previous coursework you've taken.

#### 1.1.1 Opinionated Analysis Development

The idea of research being opinionated is something we borrow from Hilary Parker, who has been developing the idea of “opinionated analysis development” over the last several years. Parker (2017) argues that opinionated analysis development is a guard against human errors in data analysis that are common but preventable. She emphasizes process repeatedly because, in her view, errors in data analysis are often the result of poor process (and not the individual failures of an analyst).

Following Hilary's lead, we will also make process a cornerstone of our coursework. This emphasis on process is not routine in research methods courses (Long 2009), which often treat methods and techniques in isolation and leave much about how they fit together to be learned “the hard way”. A common experience with statistics coursework is that students will learn the assumptions and requirements of various tests, but not how to fit them together to produce a well-conceptualized analysis. Thus, students working on their first research projects often feel personal responsibility for errors “despite not being taught processes that protect against them” (Parker 2017:2).

Parker (2017) lays out three core areas for analysis development. Analyses should be (1) reproducible and auditable, (2) accurate, and (3) collaborative. She also provides a set of opinions and questions about each of these categories. Projects that are both reproducible and auditable, for example, should be driven by executable analysis scripts with clearly defined dependencies. Projects that are accurate should use modular, tested code and should assertively test data, assumptions, and results. Finally, projects that are

collaborative should use version control software for project management, should have the ability to track issues, and should allow for communications that can be archived easily.

This text introduces strategies and techniques for implementing these opinions in data science research, with special emphasis on doing so in social science and geospatial projects. While these ideas appear in nearly every chapter, the chapter on “Good Enough” Research Practices pays particular attention to these concepts in greater detail.

### 1.1.2 Contextualizing These Ideas

Each time I teach a research methods course, I have students with a mix of experiences. For some students, all of these concepts and techniques are new and they therefore do not have a strong conception of what it means to do research. For other students, particularly those with research experience, these ideas can directly conflict with how they have been taught in the past and how they work. For everyone, these ideas require a greater mindfulness and attention to detail than they are used to.

Occasionally, students react strongly to these ideas. For example, these opinions could be interpreted as an indictment of students’ own processes or seen as a desire to micromanage. Others could see the focus on process as misplaced. After all, as long as the final product looks good, what does it matter how organized the analyst’s hard drive is (or isn’t)? If you feel like this at any point in the semester, I can sympathize greatly. I remember the frustration I felt the first time I was told that I didn’t name variables well and that my code was written in a way that was hard to read.

What I didn’t understand at the time, and what I strive to emphasize now, is that process cannot be separated from product. If the process is unclear, undefined, or disorganized, our faith in the final product should be diminished because these flaws limit our ability to judge its accuracy. Thus being “right” is not solely a judgement of the final results but also how they were obtained. This emphasis on process is inherently more time consuming, slower, and more deliberate than a slapdash approach to data analysis. Our goal is not to conduct data science work in the easiest way, however, it is to conduct it in the most robust way possible.

## 1.2 What is data science?

Given data science’s new emergence, its definition remains both contested and often unclear. For me, there are four key aspects to focus on when considering what constitutes data science:

1. Statistics
2. Programming
3. Visualization and Communication
4. Substantive Knowledge

I think of this as a “full stack” approach to computational research. You want to be well versed not only the techniques for generating and analyzing data, but also substantively in the academic literature about your area of interest as well as ways to communicate your findings with other researchers and the wider public.

### 1.2.1 Statistics

Statistics covers the mathematical techniques that we use to draw inference from our data. It is the main subject of one my courses, Quantitative Analysis. In Introduction to GIS, we do not explicitly cover much in the way of inferential statistics. However, the course is designed to prepare you for a next level, Intermediate GIS course that covers spatial statistics.



### 1.2.2 Programming

Computer programming is an essential part of data science broadly and computational social science more specifically. Using a programming language means that our work can be easily reproduced. This emphasis on **reproducibility** is a response to a growing fear in many disciplines that results are replicable from study to study, which raises questions about the validity of much of the research work that we do. Our goal in both courses is to produce research that is as reproducible as possible.

This book introduces two programming languages - R and Python. In Quantitative Analysis, we will be focused exclusively on learning R and using it to produce statistical analyses. In Introduction to GIS, we will spend a lot of time in R, but we will also use some Python to help pass data from R to ArcGIS, the mapping application we will use. I will also provide some additional Python lessons to folks who are interested, but these will not be required for the course.

### 1.2.3 Visualization & Communication

Visualization is a fundamental aspect of data science work. It is how we make our results easily digestible and accessible to a wider audience, many of whom may not be able to interpret statistical output but can learn from a well-designed scatter plot. In Quantitative Analysis, we will focus on building plots to communicate information about statistical distributions and the relationships between our variables. In Introduction to GIS, we will use the same fundamental skills to build simple maps in R. We will extend our emphasis on visualization to ArcGIS, where we will focus on producing cartographically rich depictions of our data.

Separately, we will discuss the presentation of statistical data in tables using LaTeX in Quantitative Analysis as well as the producing to conference-style presentations to communicate research findings. In Introduction to GIS, we will focus on producing conference-style posters instead. These are different mediums, but they rely on the same design fundamentals that are covered in this text.

### 1.2.4 Substantive Knowledge

Substantive knowledge covers two tangentially related topics: the ability to work well in groups and the ability to digest and integrate an academic literature into your own research. Each of these topics receive some focus in both Quantitative Analysis and Introduction to GIS. Each class has some group work associated with the completion with weekly lab assignments. Introduction to GIS also has a group work component associated with the final project. In each class, students' final projects are focused on a specific content area that requires at least some background research and knowledge. Synthesizing this knowledge and integrating it into your final projects is a key piece of addressing this facet of data science.

## 1.3 How is this book organized?

Like many books about data science, this text follows a number of standard conventions related to how it is organized and how examples are given. This section introduces those conventions.

### 1.3.1 Typefaces and Fonts

Technical publications that describe scientific computing processes use a `monospaced typewriter style typeface` to refer to commands (inputs) and results (outputs). In some documents, like lecture slides and cheat-sheets, I may highlight a command by using a particular color to increase the visibility of the command name itself.

The `typewriter` typeface is also used to refer to functions (e.g. `library()`), file names (e.g. `mpg.csv`) or file paths (e.g. `C:\Users\JSmith\Desktop`). Finally, we will use the `typewriter` typeface to refer to GitHub repositories (e.g. `Core-Documents`, the repository that contains this file).

Technical publications use *italicized text* to refer to text that is meant to be replaced. These references will typically appear in a `typewriter` typeface since they are often part of commands. For example, `str(dataFrame)` (with `dataFrame` italicized) indicates that you should replace the text `dataFrame` with the appropriate variable name from your data set.

These publications also use a sans serif typeface to refer to areas of the user interface, menu items, and buttons. I cannot replicate that here because of the publishing software that I use, but you'll notice this text in course documents. We will therefore use the `typewriter` typeface in the User Guide to identify these same features.

Technical documents also use a sans serif or `typewriter` typeface to refer to keyboard keys (e.g. `Ctrl+C`) where the plus sign (+) indicates that you should press multiple keys at the same time. A sans serif typeface combined with a right facing triangle-style arrow (>) is used to refer to actions that require clicking through a hierarchy of menus or windows (e.g. `File > Save`).

### 1.3.2 Data

There are two sets of data that are used in this text, and they are also used as part of Quantitative Analysis and Introduction to GIS. Both are available as R packages. The `testDriveR` package contains some generic data that are particularly well suited for exploring statistical topics. The `stlData` package contains data that can be mapped at various levels. Both of these packages are currently available on GitHub, and can be installed using the `devtools` package:

```
install.packages("devtools")
library(devtools)
devtools::install_github("chris-prener/testDriveR")
devtools::install_github("chris-prener/stlData")
```

### 1.3.3 Examples

Throughout the semester, I will give you examples both in lecture slides and in an example do-file. Examples in lectures and course documents can be easily identified by their use of the `typewriter` typeface:

```
> library(stlData)
> str(stlLead)
'data.frame': 106 obs. of 15 variables:
 $ geoID      : num  2.95e+10 2.95e+10 2.95e+10 2.95e+10 2.95e+10 ...
 $ tractCE    : int  118100 117400 126700 119102 126800 126900 108100 127000 127400 103700 ...
 $ nameLSAD   : chr  "Census Tract 1181" "Census Tract 1174" "Census Tract 1267" "Census Tract 1191." ...
 $ countTested : int  345 871 458 182 486 1296 903 585 2116 417 ...
 $ pctElevated : num  9.57 12.06 18.12 2.2 4.73 ...
 $ totalPop    : int  1161 4307 1089 3237 3490 4590 3144 2052 5486 2408 ...
 $ totalPop_MOE : int  192 447 199 309 231 826 464 273 516 274 ...
 $ white       : int  414 2604 432 2008 3026 148 108 304 1777 2149 ...
 $ white_MOE   : int  100 303 116 262 270 217 111 82 391 212 ...
 $ black       : int  724 1338 631 646 194 4320 3020 1739 3603 156 ...
 $ black_MOE   : int  179 374 187 210 98 760 442 283 621 190 ...
 $ povertyTot  : int  324 615 506 958 349 1743 652 331 2524 254 ...
 $ povertyTot_MOE : int  140 255 164 234 129 825 305 156 598 88 ...
```

```
$ povertyU18      : int  109 169 98 15 35 627 256 47 1110 15 ...  
$ povertyU18_MOE: int   105 156 60 25 47 595 136 79 318 23 ...
```

Examples will almost always use the data frame `stlLead`, which comes with the `stlData` package. To open it, simply load the `stlData` package using the `library()` function and then start referencing `stlLead` anytime you need a data frame. This allows you to easily recreate examples by minimizing dependencies within your code.



**Part I**

**First Steps**



## Chapter 2

# Approaching These Courses

Students have varying experiences learning computational techniques. For some, the math and programming that are the foundation for modern data science techniques come naturally. For others, being introduced to these concepts can be an anxiety producing experience. I am fond the phrase “your mileage will vary” for describing these differences - no two students have the exact same experience taking a computational methods course. Like the previous chapter, some of the content below is specific to the courses (Introduction to Geographic Information Science (SOC 4650/5650) and Quantitative Analysis: Applied Inferential Statistics (SOC 4930/5050)) I teach at Saint Louis University, the general outlook on approaching data science work that I describe below is hopefully applicable in a far wider arena.

## 2.1 Zen and the Art of Data Analysis

One of the biggest challenges with this course can be controlling the anxiety that comes along with learning new skills. R syntax, GIS terms, LaTeX commands, and Markdown can seem like foreign alphabets at first. Debugging R syntax can be both challenging and a large time suck, in part because you are not yet fluent with this language. Imagine trying to proofread a document written in a language that you only know in a cursory way but where you must find minute inconsistencies like misplaced commas.

For this reason, I also think it is worth reminding you that many students in the social sciences struggle with computational methods at first. It is normal to find this challenging and frustrating. I find that students who can recognize when they are beginning to go around in circles are often the most successful at managing the issues that will certainly arise during this course. Recognizing the signs that you are starting to spin your wheels and taking either ten minutes, an hour or two, or a day away from computational coursework is often a much better approach than trying to power through problems.

Data analysis therefore requires a certain mindfulness. I mentioned in the Preface that much of what this book covers are “habits of mind and habits of method”. These mental habits extend past being able to recognize that frustration is setting in. They also include the mental habits needed for Getting and Staying Organized and strategies for Getting Help as you navigate the inevitable errors that come with learning new analytically skills.

## 2.2 Getting and Staying Organized

Doing data science work, and having the space to step away for a day as the last section suggests, requires discipline and organization. Similarly, computational coursework can be demanding not just because it is complex but because the courses often have a number of moving pieces that you need to keep track of. Being

mindful of this challenge from the beginning, and taking steps to plan for it, is an important part of this course.

### 2.2.1 Keeping Track of Where You Are

Students who have some system for tracking their work and creating to-do lists are often the most successful in this course, not because they have a fundamentally better grasp on the content but because they simply are more organized. If you have never thought particularly hard about how you manage tasks, now is an excellent time to start doing so. You do not need fancy computer software to accomplish this, though there are an array of possibilities if you do want to use software to keep yourself organized. A legal pad or a notebook can be just as effective as a \$50 to-do list manager. The point is, do *something*!

I am fond of recommending the **Getting Things Done** methodology to students as part of thinking more holistically about staying organized. The website Lifehacker posted an excellent introduction to GTD that is a great way to get a sense of how it works and find additional resources for implementing it.

The GTD website has a great list of software for those of you looking for a to-do list application. One that isn't listed that I use for collaborating with my student research team is Trello, a freemium website that allows you to create simple to-do lists. It isn't sophisticated enough for implementing GTD, but it is more than sufficient for managing to-do lists related to this course.

### 2.2.2 Managing Course Materials

For both courses covered by this book, materials will be disseminated in physical and digital form. The topic of keeping digital materials organized is the subject of the next chapter Protecting Your Work. The physical handouts and materials that we'll give out will all come three hole punched, and you are expected to purchase a three-ring binder for this course. I give out enough handouts that, without a binder, it becomes difficult to keep track of where handouts are located. I have watched students spend more time looking for the handout than it takes to find the answer to their question once they have the handout itself.

There are two recommended ways to keep your binder tidy. One is to organize by lecture. There are sixteen lectures worth of handouts in both courses (not including the course preview and the final research conference session), so two sets of dividers (which often come in packs of eight) should be sufficient. The other way to organize handouts is by topic. We hand out a number of different types of documents that could be organized thematically like so:

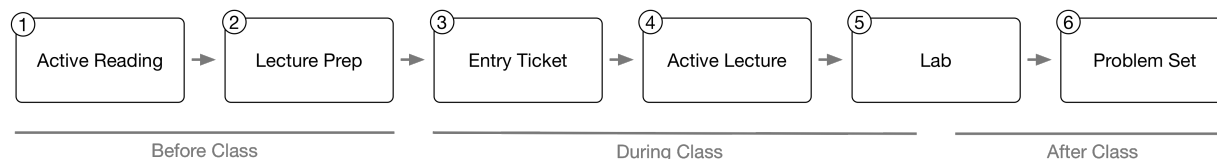
1. Syllabus and reading list
2. Workflow diagrams
3. R function sheets
4. ArcGIS process sheets (SOC 4650/5650 only)
5. LaTeX command sheets (SOC 4930/5050 only)
6. Sample outputs
7. Exercise handouts
8. Lab handouts
9. Problem set handouts

Since there are approximately eight different types of handouts in each class, a single package of dividers should be sufficient for this approach to organizing them.

I don't think that one way of organization, thematically or temporally, is inherently better or worse than the other. In large part it comes down to how *you* like to organize information. After all, learning how to process and organize information in a way that works for you is the most important takeaway here.



## 2.3 Course Flow



Both Quantitative Analysis and Introduction to GIS follow the same weekly flow or progression. The structure of the assignments and course materials presupposes that you adhere to this progression.

### 2.3.1 Reading with Purpose

The book and article **reading assignments** for this course are different from most of the other reading you will do in your graduate program because they are often very technical. Students who are most successful in this course read twice. Read the first time to expose yourself to the material, then take a break from the reading. During this first read, I don't recommend trying to complete the example problems or programming examples. Focus on the *big picture* - what are the concepts and ideas that these readings introduce?

During the second read, try to focus in in the *details* - what are the technical details behind the big picture concepts? I recommend doing this second read with your computer open. Follow along with the examples and execute as much of them as you can. By using this second read through as a way to test the waters and experiment with the week's content, you can come into the lecture better prepared to take full advantage of the class period. Students who follow this approach are able make important connections and focus on the essential details during lectures because it is their third time being exposed to the course material. They are also in a much stronger position to ask questions.

### 2.3.2 Lecture Preps

Once you've completed the readings for the week, tackle the **lecture prep**. These short assignments are designed to help prepare you for the upcoming lecture by reviewing some of the concepts from the reading. For many of the preps, I will post replication videos on YouTube that show how to solve each problem and provide some explanation for that process. Use this as an early indication of how well you understood the readings and what questions you might have. Bringing these questions to class is a great way to build your knowledge of the course material.

### 2.3.3 Active Lectures and Labs

During **lectures**, I introduce many of the same topics that your readings cover. This again is intentional - it gives you yet another exposure to concepts and techniques that are central to geospatial science. One mistake students sometimes make is focusing on the details of *how* to do a particular task rather than focusing on *when* a task should be done. If you know when a task is needed but cannot remember how to do it in R, you can look this information up. Conversely, detailed notes on executing R commands may not be helpful if you are unsure when to use a particular skill. There is no penalty in this course for not knowing how to execute a command from memory; this is what reference materials are for. The most successful students will therefore focus on *when* a particular skill is warranted first before focusing on *how* to execute that skill

Getting experience with executing tasks is the purpose of the **lab exercises**. These are narrowly defined and focused on building your confidence with the specific skills introduced each week. Time for beginning these exercises is given at the end of each class meeting, and replication files will be posted on GitHub for each lab. I suggest that students do not look at replication files until they are truly stumped. Spend some

time wrestling with problems and code, and experiment with the trouble shooting process, when the stakes are low and you have a safety-net to catch you. This will prepare you to respond to issues on the problem sets in a more resilient way.

### 2.3.4 Problem Sets

The problem sets are the key evaluation of your progress in the course. These will cover skills both for the week they are assigned as well as previous weeks to ensure that you are connecting various aspects of the course and are able to transition skills from one week to the next. Replication files will be provided after all problem sets are submitted and grades are returned. Review these replication files regardless of how well you do on a given problem set - there is much to learn from reviewing your progress and seeing a different approach to these assignments.

### 2.3.5 An Apple a Day

As I noted above, some degree of frustration with these courses is to be expected - functions in R will not work as intended, for example, and the ambiguity of error messages can be excruciating. I suggested above that you build into your approach to the course time to walk away from assignments if you hit a wall. Being able to walk away from an assignment for a day requires excellent time management. If you are waiting until the night before or the day of an assignment's due day to begin it, you give yourself little room for errors.

I recommend approaching this course in bite size chunks - a little each day. The most successful students do not do all of their reading, homework, and studying in a single sitting. I find that this approach not only creates unnecessary anxiety around assignments, it also dramatically limits the amount of course material you can absorb. Keep in mind that I expect the *median* student to spend approximately six hours on work for this class each week (twice the amount of in-class time). A sample approach to the class might look something like this:

- Monday: class
- Tuesday: finish lab
- Wednesday: Start problem set
- Thursday: Finish problem set
- Friday: First reading
- Saturday: Second reading
- Sunday: Lecture prep for next class

The single biggest failure point in both courses for students is thinking that the few hours before class are sufficient for doing *all* of the work required for a particular week. It is the rare student who can do this successfully (not only in terms of meeting the required deadlines but also in terms of actually learning the material). Err on the side of giving yourself too much time to complete work rather than just enough!

## 2.4 Staying Current

Course content is maintained both on the respective course websites (Quantitative Analysis and GIS) and GitHub organizations (Quantitative Analysis and GIS). Course materials are available throughout the semester and after the course is done. This creates two challenges. First, for all students, attention must be paid to when updates are pushed to GitHub and the course website. This is particularly true for content updated *after* a lecture. Second, for students who like to work ahead, care must be taken to ensure that you are working off of materials for the current semester and not the last semester the course was taught. There are two ways in which I help communicate the status of course content - badges and Slack.

### 2.4.1 Badges

I use badges on both the GitHub `README.md` files in individual lecture repositories and on individual lecture webpages to signal which semester the content is current for and what stage of development the content is in. These badges are used not just because they are effective but as part of a more general socialization into data science software development, where badges are used to signal development status on R package GitHub repositories. For example, the `ggplot2` `README.md` file looks like this:

## ggplot2

build error  coverage 74% 



### Overview

`ggplot2` is a system for declaratively creating graphics, based on [The Grammar of Graphics](#). You provide the data, tell `ggplot2` how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.






There are four badges in total, all of which sit under the `ggplot2` package name. We can see from the first two badges that continuous integration services are used, and that `ggplot2` is only passing its tests on one of them. The third badge indicates that the built-in tests for `ggplot2` test 74% of the package's code. We can also see from the fourth badge that the current version of the software on CRAN is version 2.2.1 (as of December, 2017). Details like these are not critical when you are first learning to use R, but they grow more important as your data science skills progress.

We use the following badges for our course repositories and on the course websites:


The badge that indicates the semester that the content is current for is orange and looks like this:

semester 


Badges that indicate the development status of the content are either red, orange, or green. Here are the badges and their meaning:

-  - only the Lecture Prep has been updated from previous semester
-  - some content updates from previous semester are in draft form
-  - lecture content (slides, exercises, handouts, and the lab) have been updated but problem set has not been updated from the previous semester
-  - all content has been updated from the previous semester
-  - additional updates have been made to this semester's content


The badge that indicates the version number is blue:

version 

The badge that indicates when the last commit was made will look like the badge below, though the color may vary. Bright green badges are for changes that have been made within the last several days.

last commit 

Finally, the badge that indicates the size of the repository is blue and looks like this:

repo size 

Here is an example of how the badges look on a course repository. As of January 2018, the `README.md` associated with the `Core-Documents` repository for Quantitative Analysis looked like this:

## Core-Documents

semester fall 2017 release updated version v17.4.2 last commit today repo size 2.47 MB



### Repository Contents

The README.md badges, from left to right, represent the following:

1. **semester**: the contents are current for the Fall 2017 semester
2. **release**: the contents have been updated since they were first released to students
3. **version**: the version number of the repository (see Versioning below) is v17.4.2
4. **last commit**: the last time the repository was edited was the day the screenshot was taken
5. **repo size**: the repository size is 2.46 megabytes (MB, see this article for details on what computer disk drive size measurements mean)

Here is an example of how the badges look on a course website. As of January 2018, the Course Preview page for GIS looked like this:

### Course Preview

#

semester spring 2018 release full last update 2017-12-29

From these three badges, we can see that the webpage is current for the Spring 2018 semester and that it is the full release - no additional updates to the site are planned. Finally, we can see the date on which the page was last updated - December 29, 2017.

### 2.4.2 Versioning

The versioning system used on course repositories is meant to emulate the release system used for data science software. Software development increasingly takes place on a website called GitHub.com. GitHub includes tools for packaging a repository at particular points in time. The release system allows developers to identify how an R package, for instance, has changed over time. This is important for reproducibility. For instance, you can see this example of the release history for `ggplot2`:

v2.2.0  
0cf7c56

## ggplot2 2.2.0

hadley released this on Nov 14, 2016 · [222 commits](#) to master since this release

### Assets

- [Source code \(zip\)](#)
- [Source code \(tar.gz\)](#)

### Major new features

#### Subtitle and caption

Thanks to [@hrbrmstr](#) plots now have subtitles and captions, which can be set with the `subtitle` and `caption` arguments to `ggtitle()` and `labs()`. You can control their appearance with the theme settings `plot.caption` and `plot.subtitle`. The main plot title is now left-aligned to better work better with a subtitle. The caption is right-aligned ([@hrbrmstr](#)).

The package's developers use the release system to document what has changed. We will emulate this with the `Core-Documents`, `lecture`, and `finalProject` repositories. For example, here is the release documentation for SOC 4650/5650 `Core-Documents` repository:

Latest release

v18.1.1  
0566923

## Draft Release v18.1.1

chris-prener released this 7 minutes ago

### Assets

- [Source code \(zip\)](#)
- [Source code \(tar.gz\)](#)

This is the draft release of the `Core-Documents` repository. The `reading-list.pdf` file still needs some small updates to reflect readings from [Sociospatial Data Science](#).

If you want to see what has changed over the course of the semester in a course's repositories, you can use the release notes to compare different versions. These notes are also stored in the `NEWS.md` file for each repository. The use of both releases and a `NEWS.md` file is a common way to document changes on GitHub. For example, you can see look at the `NEWS.md` file for `ggplot2`, which will mirror the release notes for the package.

In software development, version numbers are used along with releases to make it easy for end users to keep track of how releases have progressed. In the picture above of the `ggplot2` package's `README.md` (see Badges), one of the badges indicated the version of the software that was available via CRAN - 2.2.1. The initial 2 is what is known as the major release number. Major releases typically represent large changes to the software. The second number, also a 2 in this case, represents the minor release number. Minor releases introduce fewer changes and may be aimed more at fixing issues. Finally, the 1 represents the patch release number. Patch (or maintenance) releases are aimed exclusively at fixing bugs and issues.

For the course, version numbers will also have three parts. From left to right, the first pair of digits represents the year of the current course offering, the second pair represents whether it the development status is (1) draft, (2) lecture, (3) full, and (4) updated, and the final number represents maintenance changes that do not

rise to a change in the development status. For example, `v18.1.1` would represent the initial maintenance release for the 2018 version of a course that is in the draft development stage.

If you see a version number like `v18.1.1.9000` in `NEWS.md`, the notes will be tracking changes that have not been incorporated into an official release yet. The use of the `9000` designation follows common practice in the R community.

### 2.4.3 Slack

The `#_news` channel in each course's Slack organization will be automatically updated with a list of commit messages for each lecture repository. The updates will look like this:



You will be able to scan through commit messages for updates that might be relevant for you. Commit messages are a core part of using Git, and are discussed in the Basic Git chapter. From these messages, we can see that changes were made to the `LICENSE` (one of the meta documents) in the `Core-Documents` repository for GIS. We can also see that a change was made to the Syllabus's content for the course.

I recommend scanning through all new messages in `#_news` each time you check in with the course on Slack. If you see changes, sync your repository (if you are keeping local copies) or download the updated document.

## Chapter 3

# “Good Enough” Research Practices

This section introduces some of the core concepts that support sociospatial data science works. The title of the chapter takes inspiration from a recent article titled *Good enough practices in scientific computing* (Wilson et al. 2017). The authors note in their introduction that scientific computing advice can sometimes be both overwhelming and focused on tools that are inaccessible to many analysts. Their goal, and the goal of this text, is to de-mystify the simplest tools that enable researchers to streamline their workflows:

Our intended audience is researchers who are working alone or with a handful of collaborators on projects lasting a few days to a few months, and who are ready to move beyond emailing themselves a spreadsheet named **results-updated-3-revised.xlsx** at the end of the workday...Many of our recommendations are for the benefit of the collaborator every researcher cares about most: their future self.

I would argue that the skills they describe are useful beyond just a few months. Indeed, most of the skills here can dramatically improve students’ dissertation experiences:

Most importantly, these practices make researchers more productive individually by enabling them to get more done in less time and with less pain. They also accelerate research as a whole by making computational work (which increasingly means all work) more reproducible. But progress will not happen by itself. Universities and funding agencies need to support training for researchers in the use of these tools. Such investment will improve confidence in the results of computational work and allow us to make more rapid progress on important research questions.

While much of what we will talk about in this text is aimed at supporting your work, there are benefits that extend beyond your dissertation or your research projects. These benefits, which include developing sustainable workflows and structuring the way you interact with your own computer, can make everyday computing practices like checking email or organizing files an easier, more structured process.

### 3.1 Reproducibility

One of the mantras of this text is an emphasis on reproducibility. The unifying feature of all of the “good enough” research practices discussed below is that they contribute to a more reproducible research product.

Reproducibility is very much in vogue right now for number of reasons. Assessments of studies in psychology<sup>1</sup>, for example, have found weaker on average effect sizes and far fewer statistically significant results than the initial studies reported. There have also been high profile instances of falsified research, including research by a graduate student at UCLA. This particular instance of fraud was identified by graduate students intent on replicating the original study.

---

<sup>1</sup>Open Science Collaboration, 2015. Estimating the reproducibility of psychological science. *Science*, 349(6251), p.aac4716.

At the same time, there is a recognition that the skills necessary for producing reproducible research are not being fostered in academic disciplines and graduate programs. Thus one of the goals of this course, and this **User’s Guide** in particular, is to help develop a working knowledge of many of these skills.

One challenge, however, is that reproducibility does not have a consistent definition. Some researchers use the term to narrowly refer to code that can execute without alteration on a person’s computer. Others use it to refer to research designs that can be replicated by other researchers. Still others discuss reproducibility as the ability to obtain a similar set of results or draw similar inferences from identical research designs.

When we talk about reproducibility in this class. We’ll be primarily concerned with **methods reproducibility**:

the ability to implement, as exactly as possible, the experimental and computational procedures, with the same data and tools, to obtain the same results.<sup>2</sup>

Methods reproducibility in statistics means that other analysts have full access to both the original data and the steps used to render those original data into a final research product, such as a set of regression models. This is increasingly seen not just a matter of good research methodology, but as a matter of research ethics as well. Being able to be transparent with research decreases the potential for cases like the fraudulent dissertation research conducted by a UCLA graduate student named Michael LaCour. It was the efforts of two Stanford graduate students who wanted to reproduce LaCour’s findings that ultimately led to the identification of problematic work.

For statistics, methods reproducibility is derived from a number of sources. The first source is the use of **computer code** for working with data. Rather than making manual changes to tabular data in a spreadsheet application like Microsoft Excel, computer code provides detailed records of each individual alterations. Code can be used to execute tasks repeatedly, meaning that errors can be easily fixed if they are discovered an hour, a day, a week, or a month later. During this semester, we’ll use R’s programming language to execute reproducible data cleaning processes.

The second source of reproducibility in statistics is therefore derived from the **documentation** that we create to accompany our research products. These documents outline where our data originated, what specific variables mean (a codebook), what steps were taken to create specific maps (a research log), and how our data files are organized (a metadictionary).

Our code can also be used as documentation if it is written using literate programming techniques. In R, these techniques produce well annotated output that “weaves” together code, output, and narrative text that describes the function of the code and the results of the output.

The third and final primary source of reproducibility in statistics is derived from our **organizational approach** to our work. Statistics projects can require many megabytes of data spread across dozens of data files, scripts, and output files. A disorganized file system can make replicating your work difficult if not impossible. Much of the research practices discussed in the remainder of this section are aimed at supporting one or more of these three major sources of reproducibility.

## 3.2 Thinking in Workflows

One way to increase the reproducibility of a project is to approach each and every task with purposeful organization and thoughtfulness. **Workflows** are the processes that we use to approach a given task. Think of checking your email. You (hopefully!) follow a series of steps when you check your email that help you organize your inbox. In his excellent primer *The workflow of data analysis using Stata* (2009), Scott Long describes a structured strategy for approaching statistical research. In Long’s model, a data analysis project consists of four steps: (a) data cleaning, (b) analysis, (c) presenting results, and (d) protecting files. This is a useful model to build upon, and one that we will discuss over the course of the semester.

<sup>2</sup>Goodman, S.N., Fanelli, D. and Ioannidis, J.P., 2016. What does research reproducibility mean?. *Science translational medicine*, 8(341), pp.341ps12-341ps12.



Even more useful, not just for statistical work but for any process, are the tasks Long lays out for each step in the data analysis workflow:

1. Planning
2. Organization
3. Documentation
4. Execution

A good example of the utility of extending this logic to other workflows is with the problem sets. The “typical” approach students take with homework assignments is to sit down, open up their software, and start with question 1. Using Long’s four task approach, a workflow-based strategy to the assignment would involve beginning by reading the assignment through in its entirety to develop a **plan** for approaching it - think about what techniques and skills are needed for each step. With a plan in place, you can proceed to **organizing** yourself for the assignment - identifying and obtaining files that you will need, creating dedicated directories for saving assignment data, and getting any necessary software documentation. After pulling together all of these materials, you are ready to move on to **documentation** - setting up your assignment code and output files, and (later in the course) your research log and meta-dictionary. Once you are set-up, you would then begin to address individual assignment questions as part of the **execution** task.

The goal here is to approach everything you do for research or work with an element of mindfulness and structure about your process. This mental model for approaching research supports the creation of **re-producible** research products because we approach our work in a routinized, predictable, organized, and efficient manner. Thinking in terms of workflows also encourages a greater awareness of the complexity of tasks, which also helps you plan more accurately for how long a particular task or project will take.

In reality, there will be multiple workflows that you find yourself navigating. You will want a structured process not just for approaching a large research project like the final project, but also a process for maintaining notes related to a specific assignment, a process for documenting code, a process for approaching assignments, and even a process for backing your data up. As you go through the text, think about how to best integrate these ideas into your work habits.

### 3.3 Data

One of the themes in *Good enough practices in scientific computing* (Wilson et al. 2017) is an emphasis on data management. One of their core messages is to “save the raw data”. Particularly in GISc work, the raw data can be expansive - dozens of shapefiles, tabular files, and associated metadata. These files often come from disparate sources - city open data sites, the U.S. Census Bureau, state data repositories, and other federal agencies. Moreover, GIS data are often updated over time to reflect on-the-ground changes. Saving the raw data in sociospatial data science work therefore means not only creating a well-organized directory containing *all* of your original data. It also means logging the source of each file, when it was downloaded, and (if applicable) a permanent web link to your data source. For that reason, we’ll give you not just the course data but a read me file and a metadictionary that lists all of the files we’ve disseminated to you.

A second message in the paper is to “create the data you wish to see in the world”. The authors encourage readers to “create the data set you wish you had received.” First and foremost, this means using open and not proprietary data formats. For spatial data, ESRI shapefiles are technically proprietary, though their standard is open. This means that other software applications, like R and QGIS can read and in some cases write shapefiles. For sharing spatial data, a better option is the GeoJSON, which is a plain text file format. Tabular data are best stored as CSV files, which is also a plain text file format that can be opened by a wide variety of applications. In contrast, common file formats like Microsoft Excel’s XLS and XLSX are proprietary file packages that cannot be read as plain text and are therefore less desirable for storing data.

## 3.4 Plain Text

Along with creating the data we wish to see, *Good enough practices in scientific computing* (Wilson et al. 2017) repeatedly emphasizes the importance of using plain text files for writing as well as for data. Using plain text file formats, like `.txt` and `.md` files, frees us from reliance on proprietary applications that we cannot be sure will exist in ten or twenty years (and are thus threats to reproducible research). They can also be opened on virtually any computer across a variety of operating systems. The authors of the article reference Markdown files, which are really just plain-text files containing special characters that can be rendered by websites and other applications. Markdown, which is discussed later in this text, provides us with the best of both worlds. We can produce outputs with rich text like styling but that is saved in a plain text file container. Writing in a markup language can seem daunting at first, but within a few weeks I have found that most people can start to feel comfortable inserting markup syntax without significant cognitive burden.

## 3.5 Version Control

One of the key emphases of both *Good enough practices in scientific computing* (Wilson et al. 2017) and “opinionated analysis development” is that data science work should be tracked using version control software. This text introduces Git in the Basic Git chapter, though there are other options like Subversion and Mercurial. Git is introduced because it has been widely adopted by the R community and because of the robust web tools associated with GitHub.com. Using version control gives us a timeline of our work along with the ability to revert back to previous versions of our files. This creates a “chain of evidence” for our research process and protects us from irrevocably altering files.

As the authors of *Good enough practices in scientific computing* (Wilson et al. 2017) note, learning version control software can be tricky. There are two chapters later in this text dedicated to Git covering both Basic Git and Advanced Git techniques. Those chapters also provide links to other resources and tips for using those tools successfully.

## Chapter 4

# Protecting Your Work

One of the biggest challenges to with doing computational work is that it requires a high degree of organization. Our approaches to file management on computers are often haphazard, with **Documents** and **Downloads** folders filled with disorganized files or a **Desktop** covered in dozens of files. Perhaps this has worked in the past, but that approach (or lack thereof) **will** fail students in Introduction to Geographic Information Science (SOC 4650/5650) and Quantitative Analysis: Applied Inferential Statistics (SOC 4930/5050). Moreover, if left un-checked, it will fail students down the road when a hard drive failure or natural disaster renders their data irrevocably inaccessible. This chapter covers what some of those threats are and ways to manage those risks when conducting computational research.

### 4.1 Threats To Our Data

Each semester that I teach Introduction to GIS and Quantitative Analysis, several things happen. The first thing that happens is that students regularly lose files. The effects of losing files can range from being a minor frustration to a major headache depending on the file in question. Losing files often results in downloading multiple copies of the same data and recreating work. Both of these are wastes of your time. Moreover, files are rarely gone. They are typically just misplaced. This is bad for reproducibility, particularly when you happen across multiple versions of the same file and have to sort out which version is the version you last worked on.

The second thing that happens is that students who use thumb drives for data storage lose them. Depending on the timing of this loss, this can again range from being a minor frustration (very early in the semester) to being downright anxiety attack producing (last few weeks of the semester). Recreating an entire semester's worth of work on the final project is both a tremendous waste of your time and a particularly unpleasant experience.

Fortunately, I have never had a student's computer hard drive die during the course of the semester. However, I assume that if I teach this course long enough a hard drive failure will indeed occur. The backup provider Backblaze has analyzed their own hard drives and found that about 5% of drives fail within the first year. After four years, a quarter (25%) of drives in their data center fail. Similarly, it is only a matter of time before a student's computer is stolen along with all of their hard work. A less likely though still very plausible scenario involves the destruction of a student's belongings (computer and thumb drive included) in a fire, car accident, or natural disaster.

Despite the likelihood that you will at some-point lose a thumb drive (if not during this semester than sometime down the road) and the near certainty that your computer's hard drive will eventually fail if a rogue wave does not get it first, few students and faculty take these risks seriously. While you cannot prevent many of these things from happening, I want to suggest to you that you can take some simple steps to sure that *when* (not if) they happen, you are well prepared to get back to work with minimal disruption.

## 4.2 Creating a Sustainable File System

The first thing that students can do to stave off problems with lost files is to actively and mindfully manage their files. In his excellent document *The Plain Person's Guide to Plain Text Social Science*, Kieran Healy describes two important revolutions in computing that are currently taking place. One of them is the advent of mobile touch-screen devices, which he notes

hide from the user both the workings of the operating system and (especially) the structure of the file system where items are stored and moved around.

For most users, I would argue that this extends to their laptop or desktop computers as well. I would venture to guess that the majority of my students are used to keeping large numbers of files on their desktops or in an (distressingly) disorganized **Documents** folder. For research, particularly quantitative research, such an approach to file management is unsustainable. It is difficult to produce *any* research, let alone work that is reproducible, without an active and mindful approach to file management.

### 4.2.1 Create a *Single* Course Directory

The most successful approach to organizing files is to identify *one and only one* area that you will store course files in. Having files scattered around your hard drive between your **Desktop** directory, **Downloads**, **Documents**, and a half dozen other places is a recipe for lost files. It can also add complexity to the task of backing these files up. I recommend naming this directory simply based on the course you are enrolled in:

- SOC4650
- SOC4930
- SOC5050
- SOC5650

These names are short, have no punctuation or spaces (which can create conflicts with software), and explicitly connects the directory to this course as opposed to other courses you may take that are also statistics or GIS courses (a good reason to avoid naming the directory **Stats** or **GIS!**).

This single course directory should reside in *one and only one* place. Once you have these folders set-up, I want you to an agreement with yourself: files for the course will *only* be saved within this structure. Do not temporarily save files to the **Desktop**, for example, intending to move them later. They will be forgotten. Like New Years resolutions, it is easy to say that you will use a file system but difficult to maintain that promise when stress sets in or you are rushed. Do *everything* you can to maintain your file system.

### 4.2.2 Storing the Course Directory

Storing this directory inside a sync'd directory (like Dropbox or Google Drive) may cause conflicts with your Git repositories. Instead, students for both of my courses should utilize a large sized thumb drive or an external hard drive for data storage (see the "Course Onboarding" section of the respective course's website for additional details). Having data stored on an external device raises the risk of physical loss, but also makes it easier to move work between different computers. This is particularly challenging for GI Science work, where spatial data sources can be many megabytes in size.

macOS users should make sure that the external drive is formatted using the ExFat file system specification so that it is readable both by their operating system and the Windows operating system in the computer lab. If you buy a new drive, you will likely have to re-format it. Directions for doing so are available from Apple.

### 4.2.3 Approach Organizing Systematically

Within your single course directory, I recommend following a mindful, purposeful approach to organization. This approach begins with having a number of dedicated subfolders within your course directory:

```
/SOC5650
  /Core-Documents
  /DataLibrary
  /DoeAssignments
  /Lectures
  /Notes
  /Readings
```

Note again how these directories are named - there are no spaces, special characters, and the names are deliberately short but specific. For a directory with two words (e.g. `DataLibrary`), I use what is known as camelCase to name the file where the second (any any subsequent) words have their first character capitalized. You could also use dash-case (e.g. `Core-Documents`) or snake\_case (e.g. `Core_Documents`) as a naming strategy. Regardless of which of these approaches you take, try to use it consistently.

For both of the courses that I teach, a basic file system will be made available for download onto your external device. See the respective course websites for more information about course data releases. Only the data release for Introduction to Geographic Information Science (SOC 4650/5650) will contain a `DataLibrary` directory.

### 4.2.4 The Core-Documents Directory

The `Core-Documents` directory is used for storing the course syllabus, reading list, and a sample schedule for 3600 Morrissey Hall for the semester. You can view it online for both Introduction to GIS and Quantitative Analysis. Once we cover using Git and GitHub in class, you should **clone** the appropriate `Core-Documents` repository into your file system. If there are updates, you will want to make sure you **pull** them down onto your local copy of the repository. You can read ahead in the Basic Git chapter for more information on how this works.

You will not be able to **push** changes that you make in this directory to GitHub, and making other changes could cause conflicts. I suggest not making changes inside this repository beyond opening files for reference purposes.

### 4.2.5 The DataLibrary Directory

This directory is only applicable to students in Introduction to GIS, whose data release will contain it. It will have copies of most data not disseminated to you as R packages. The data in this directory should be used as needed but not altered (one of the of the “Good Enough” Research Practices from the next chapter).

### 4.2.6 The DoeAssignments Directory

Like the `Core-Documents` repository, this will not be included in the course data release. Once we cover using Git and GitHub in class, you should **clone** the appropriate `Core-Documents` repository into your file system. It will also have a different name - your last name instead of ‘Doe’. Once you add it, it will contain a number of subdirectories:

```
/SOC5650
  /DoeAssignments
  /FinalProject
```

```
/Labs
  /Lab-01
  ...
  /Lab-15

/LecturePreps
  /LP-01
  ...
  /LP-15

/ProblemSets
  /PS-01
  ...
  /PS-08
```

The `FinalProject` directory will not have any subfolders in it. You will be asked to populate it with the appropriate subdirectories (see Organizing Projects below). These will be detailed in the final project instructions. The `Labs`, `LecturePreps`, and `ProblemSets` subdirectories have folders dedicated to the individual assignments you'll have to submit over the course of the semester. Like the `FinalProject` directory, you will need to populate them with the appropriate deliverables. These will be detailed in each assignment's instructions.

#### 4.2.7 The Lectures Directory

This directory should contain subfolders for each of the sixteen weeks of the course.

```
/SOC5650
  /Lectures
  /Lecture-01
  ...
  /Lecture-16
```

You will need to add these subfolders each week by **cloning** them from GitHub.com. You will not be able to push changes that you make in these directories to GitHub, and making other changes could cause conflicts. I suggest not making changes inside this repository beyond opening files for reference purposes. If there is something you need to edit, copy it to the appropriate subdirectory of `DoeAssignments` and make your changes there.

#### 4.2.8 The Notes Directory

Use this as a home for course notes if you decide to take notes digitally and do not already use notebook software of some kind that organizes notes for you.

#### 4.2.9 The Readings Directory

Use this as a home for `.pdf` copies of course readings. I suggest creating subdirectories *only* for weeks that have readings assigned from outside of the main texts, such as Lecture 01:

```
/SOC5650
  /Readings
  /Lecture-01
```

## 4.3 Organizing Projects

Following the excellent article *Good enough practices in scientific computing* (Wilson et al. 2017), I suggest data science projects should be organized in a specific, standardized way. This approach serves much the same purpose as the previous section on Creating a Sustainable File System - work that is well organized is better insured against loss.

### 4.3.1 A Sample Project

Much as I suggested above that course files reside in one and only one place, projects should be organized similarly (with one important caveat below). What follows is a sample project and then descriptions of each element. More details about each folder (excepting **maps**) can be found in the article. The project below contains all of key elements of a small geospatial research project:

```

/projectName
  /data
    rawData.csv
    rawTracts.shp
  /doc
    joinedTracts.md
    notebook.Rmd
    notebook.nb.html
    researchLog.md
  /maps
    map.mxd
  /results
    joinedTracts.shp
    map.png
  /src
    script.R
  LICENSE.md
  projectName.Rproj
  README.md

```

### 4.3.2 Top-Level Files

At least two files should be present in each project directory - a `README.md` and a `LICENSE.md`. As I noted in the previous chapter, both of these files should be plain-text files. Typically, they are formatted using Markdown syntax. The `README` should describe your project, detail any key dependencies or outside data sources that might be required, and provide other important details like how to cite your project and what other online resources might be available.

The `LICENSE` is a key element of open source research. This spells out how others may use (and not use) your work. The excellent guide *The Legal Side of Open Source* notes:

Open source is an unusual circumstance...because the author expects that others will use, modify, and share the work. But because the legal default is still exclusive copyright, you need a license that explicitly states these permissions.

There are lots of different options for open source licenses, and GitHub's choose a license site does a great job of explaining some of the many options. In the R community, many packages are made available using the MIT or GNU GPLv3 licenses. For written content, presentations, and other non-code works, the Creative Commons Attribution and Attribution-ShareAlike licenses are both common options. For data, the Open Data Commons licenses are not discussed on GitHub's site but are options for data-specific licenses. Finally,

large projects sometimes require multiple licenses. If you use a data-specific, code-specific, and/or content-specific mix of licenses, be sure to describe in the **README** which license applies to which element of the project. Licensing your projects is another form of protecting your work. If these projects are out in the open, licensing ensures that your work will be used under terms that you are comfortable with.

If the project uses R, an R project file (**.Rproj**) should also be saved in the top-level of the project. This will facilitate the top-level folder being set automatically as the working directory. The working directory is the folder that R will open data from and save data to by default.

### 4.3.3 The data Directory

All data sources, with a few exceptions, should be stored here. For instance, in the example above, a tabular data source and a raw shapefile are both used as part of the project. There are, however, a number of exceptions to this. One exception is data that has been packaged in an R package. For example, I have packaged a large set of tables containing historical census data to make them easier to work with. When these are used in projects, I do not save the tables in the **data** folder. Rather, I make reference to them in the **README** file and in other project documents.

A second exception is for large geospatial data libraries, like the data release for my Introduction to GIS. Given the size of these files, it is sometimes impractical to reproduce them multiple times across different projects. Clearly noting that specific files are stored on local libraries available elsewhere in the **README** file and in other project documents is also an acceptable alternative here.

### 4.3.4 The doc Directory

The **doc** directory should be used for all text documents associated with the project. These might include metadata associated with different project data files, an interactive R notebook, a research log file, and article manuscripts. If needed, subdirectories within **doc** should be used to keep different files organized. In the example above, **joinedTracts.md** is the metadata for the similarly named shapefile in the **data** subdirectory. The **notebook** files are the interactive R notebook and its associated **html** output. Finally, the **researchLog.md** is used for tracking higher level progress over the course of the project.

### 4.3.5 The maps Directory

**maps** is a deviation from *Good enough practices in scientific computing* (Wilson et al. 2017), which does not envision some of the complexity of working with geospatial data. It should be used for storing the **.mxd** files from ArcGIS projects or the equivalent QGIS files if that application is being used instead. Illustrator files used for post-processing work should also be stored here if used. Remember that these files should be saved using relative paths to link them to data sources to limit the possibility of conflicts down the road.

### 4.3.6 The results Directory

The **results** directory should contain any data outputs, plots, and map images. These might include intermediate and final versions of data sets as well as shapefiles and GeoJSON files created as part of the analysis process. In the example above, a shapefile combining both of the raw data sources named **joinedTracts.shp** is saved here along with the exported map image from the map file in the **maps** subdirectory. Depending on the complexity of the project, subdirectories within **results** might be necessary.



### 4.3.7 The `src` Directory

For projects that rely on functions not included in packages, for example, or that rely on other pre-made scripts, the `src` subdirectory should be used for storing script files that are called by notebooks in `doc`.

### 4.3.8 Storing Your Project

There are a number of places where work can be deposited. I teach all students in my classes how to use GitHub.com because it integrates directly into version control software, which I see as a critical component of “Good Enough” Research Practices. There are other options, however. For larger projects that may not just be on GitHub (see Basic Git) but also utilize a collaborative writing space like Google Docs, the Open Science Framework is an excellent tool. It allows for an even wider set of version control tools to be applied to projects and can be used to organize work that is not easily stored on GitHub, like large number of pdf articles, for which version control is not critical.

Once a research project has begun dissemination, repositories like Zenodo and Figshare are designed to make open source data and project materials accessible and citeable. For papers specifically, there are a number of discipline specific archives for pre-prints of papers. Many journals will allow pre-prints to remain freely available even after more formal publication. The original pre-print archive, arXiv, is focused on the STEM sciences and has inspired a number of spinoffs. Many of these are supported by the Center for Open Science, the same organization that operates the Open Science Framework tool described above. For social scientists, SocArXiv is a growing pre-print repository that is part of the Center for Open Science’s pre-print network.

### 4.3.9 Organizing Coursework

For Introduction to Geographic Information Science (SOC 4650/5650) and Quantitative Analysis: Applied Inferential Statistics (SOC 4930/5050), problem sets and the final project should both be organized following the general template above.

## 4.4 Backing Up Your Data

All of the organization in the world cannot prevent a computer hard drive from failing or a natural disaster from destroying your hardware. Similarly, even the best organized of us lose things sometimes or accidentally delete files. Backing up files is therefore a critical piece of not just doing computational work but, in this day and age, it is a requisite skill for owning computer hardware. Every device you own including your phone can and should be backed up. There are a number of different ways to think about backing up your data. The most successful backup strategies will incorporate all of these three elements.

### 4.4.1 Bootable Backups

“Bootable” backups are mirrored images of your *entire* hard drive, down to temporary files, icons, and system files. With a bootable backup, you can restore your entire computer in the event of a hard drive failure or a corruption of the operating system files. They are named as such because you can plug in the external drive that you are using for this backup and literally boot your computer up from that drive (typically a *very* slow process).

These backups are often made less frequently because they can be resource intensive and it is best not to use your operating system while creating a clone. They are typically made to an external hard drive, which is subject to similar failure rates as the hard drives inside your computer. So bootable drives need to be replaced every few years to maintain their reliability.

Both major operating systems come with applications for creating clones of your main hard drive that are bootable, and there are a number of third party applications that provide this service as well.

### 4.4.2 Incremental Backups

Incremental backups are designed to keep multiple copies of a single file (how often depends on the type of software you use and the settings you select). These can be used to restore an older copy of a file if work is lost or a newer file is corrupted.

Apple's TimeMachine is a great example of an incremental backup - when kept on, it creates hourly backups of files that have been changed, daily backups for the previous month, and weekly backups for previous months. Once the disk is full, the oldest backups are deleted. Dropbox also provides a similar service, retaining all previous versions of files (and deleted files) for thirty days.

Incremental backups are typically good options for recovering files that have been recently changed (again, depending on the software you use and the settings you select). Since they run frequently (every time a file is changed or every hour, for example), recent changes tend to get captured. They can be limited in terms of their long-term storage - it may not be possible to recover older versions of a file past a few weeks.

They are also not always good solutions for recreating your entire computer since they do not save all necessary program and operating system files, and may be cumbersome to work with if you need to recover a large quantity of files. Like bootable backups, these are typically stored on external hard drives that need to be replaced on a regular basis.

In addition to the aforementioned Apple TimeMachine, the Windows OS also comes with a built-in service for creating incremental backups. Dropbox is a good option if you have a small number of files, but you may find the need to upgrade to a paid account if you have a large amount of data.

### 4.4.3 Cloud Backups

Cloud backup services like Backblaze or Crashplan offer comprehensive backup solutions for customers. These plans typically require a monthly subscription fee to maintain access to your backups. While bootable backups protect against hard drive failure and incremental backups protect against data corruption, cloud backups protect against catastrophic events like robberies, fires, and other natural disasters. A fire or a tornado that affect your house may destroy your laptop and any external hard drives you use for backup, but your cloud backup will be unaffected.

### 4.4.4 A Workflow for Backups

Just as we need a workflow for approaching file management, it is also important to establish a routine for backups. With backups, the most successful workflows are those that require next to no effort on your part. If you primarily use a desktop, this can be as simple as leaving two external hard drives plugged into your computer since most backup software can be set to run automatically. If you have tasks that require you to manually do something (plug an external hard drive into your computer, for instance), create a reminder for yourself on a paper calendar or a digital calendar or to-do list application.



I gave a presentation on workflows for backing data up as part of the Data Science Seminar series at Saint Louis University. You can easily view the slides from that presentation on Speaker Deck, and you can download the session's materials from GitHub.

For this course in particular, it is *imperative* that you backup the data on your flash drive. A number of possibilities exist for accomplishing this:

- Keep a local copy of your flash drive's files on your computer.
- Keep a `.zip` archive of your files in a service like Dropbox or Google Drive. (Using a `.zip` archive will prevent issues with your `.git` repositories.)
- Maintain a second flash drive copies of all of your files.

Whatever solution you select, make sure you regularly update your backup. The more often you keep your backup archive updated, the less stressful and disruptive losing your drive will be. This will likely be a manual task, so follow the guidance above about creating a repeating calendar event or to-do list task reminder.



## Chapter 5

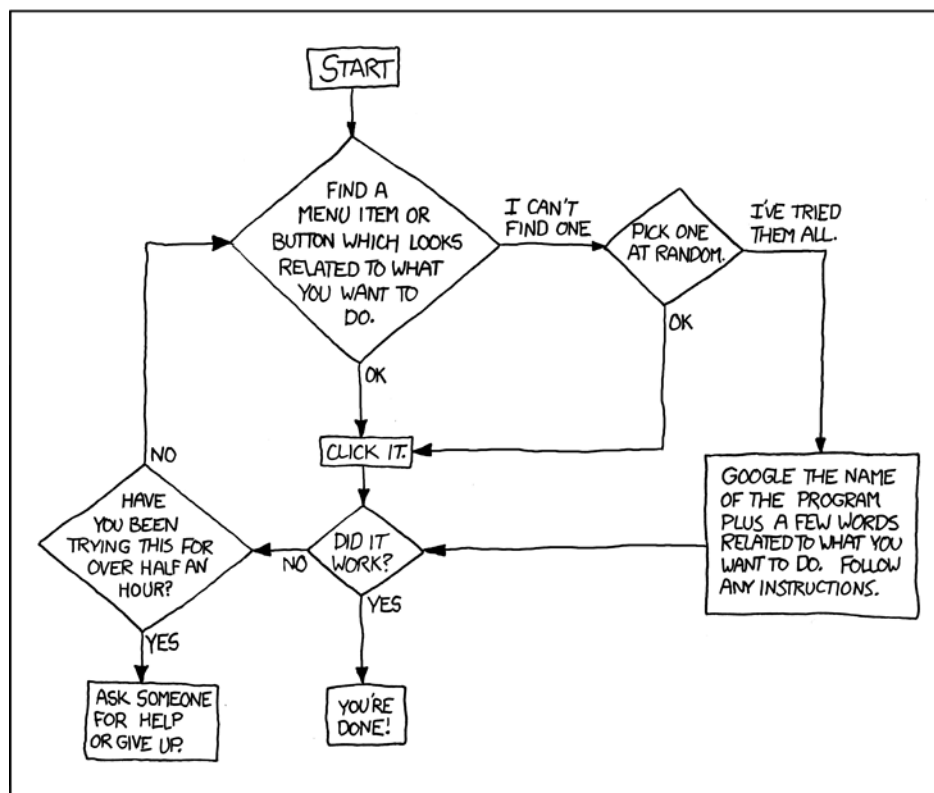
# Getting Help

One of the biggest challenges for students first learning tools like R, ArcGIS, and LaTeX is dealing with the inevitable speed-bumps and errors that come along with scientific computing. Remember, first and foremost, that these tools are not consumer software. They do not always “just work”, to borrow a turn of phrase from Steve Jobs. Learning strategies for navigating issues when the software is most definitely not working is therefore an important part of success.

One misconception that I think is important to confront is that data analysts, software engineers, and others who appear to be experts may indeed be very good at what they do, but they are equally skilled at problem solving. This xkcd comic illustrates (a bit sarcastically!) the point:

DEAR VARIOUS PARENTS, GRANDPARENTS, CO-WORKERS,  
AND OTHER "NOT COMPUTER PEOPLE."

WE DON'T MAGICALLY KNOW HOW TO DO EVERYTHING IN EVERY  
PROGRAM. WHEN WE HELP YOU, WE'RE USUALLY JUST DOING THIS:



PLEASE PRINT THIS FLOWCHART OUT AND TAPE IT NEAR YOUR SCREEN.  
CONGRATULATIONS; YOU'RE NOW THE LOCAL COMPUTER EXPERT!

Navigating challenges with data science may not be quite as easy following the flowchart above, but there are definitely strategies for working your way through errors and challenges in data science work.

## 5.1 Helping Yourself

As I noted in *Zen and the Art of Data Analysis*, a key challenge of data science work is mental. One of the traits I see frequently among students who are confronted with errors is that they stop working and wait for office hours or they immediately ask a question on Slack. Both office hours and Slack are excellent options for getting help, but there are a few strategies you should pursue first.

Before you doing anything else, however, *take a deep breath* and consider taking a break. When you being trying to fix the issue, check the time. I have spent hours trying to fix some code in R or mis-projected data in ArcGIS without success, and I have seen students do the same. Give yourself an hour time limit to try the steps below before you move on to constructing a reproducible example and asking for help from others.

### 5.1.1 Check Spelling

With R, the number one cause of errors that I see are misspellings. If objects in the global environment or variable names within a data frame are not spelled correctly (case matters!), you will get an error that an object cannot be found. The same goes for package and function names - they must be spelled 100% correctly.

### 5.1.2 Check Course Resources

If the issue is one other students have come across as well, it is likely to be discussed on Slack in the relevant channel. I may have also updated some of the course handouts or the relevant lecture's course webpage with details for how to address the concern. Check both Slack and the webpage for updates before digging deeper. Check the topic index on the relevant course website (GIS topic index) for links to different lectures where concepts or processes were covered.

### 5.1.3 Check Your Process

R functions and ArcGIS processes often have specific parameters and requirements. If you have double checked your spelling and are sure (100% sure!) that spelling is not an issue, check to make sure that you have followed all of the requirements of the workflow for the skill you are stuck on. There are two different ways to go about doing this:

- Check the course handouts and lecture slides for the relevant processes. If you cannot remember where the concepts were introduced, check the topic index on the relevant course website (GIS topic index). When I introduce processes, I often skip some (or many!) of the associated arguments and options, so the course materials are often easier to navigate than materials for other sources.
- Search official documentation sources:
  - For R, check package documentation files on CRAN. These are linked to in the package index on the relevant course website (GIS package index). You can also use CRAN's website to search for package information.
  - For ArcGIS, check ESRI's official documentation files. Make sure that you are looking at the help files for ArcGIS 10.3!
  - For Git and GitHub, check GitHub's help website.
  - For LaTeX, ShareLaTeX has an excellent set of documentation files on their knowledge base. CTAN, the LaTeX package repository, also contains documentation files for all packages that can be searched.

Use these resources to try and narrow down what might be causing your issue. Be mindful that the root cause of an error may be several steps back in your workflow. Walk back through your process to make sure your initial steps are not the cause of the error.

### 5.1.4 Cast a Wider Net

If none of these resources are sufficient, there are a few other options for seeking out guidance. When you search the following sites, it is often a good idea to search based on the specific error you are getting. Take out any aspects of the error that are specific to you, like a file path in ArcGIS, a file name, or a variable name. One set of options to search are the wealth of web resources that are available for some of the tools we use:

- For R, check out RStudio's cheatsheets and RStudio's community forums. If the package is part of the **tidyverse**, check the website as well as R for Data Science. Other packages have their own websites as well. For all of these resources, links are provided in the package index on the relevant course website (GIS package index).

- For LaTeX, there is a LaTeX wikibook with some excellent resources.

Another resource is Stack Exchange, a network of online communities on a variety of topics including:

- Stack Overflow for R, Git, and GitHub - search using tags like `[r]`, `[ggplot2]`, `[dplyr]`, `[git]`, `[github]`, etc.
- Geographic Information Systems for GIS - search using the `[arcgis-desktop]` tag
- TeX for LaTeX

Stack Exchange may have posts that address the issue or question you have. However, the match may only be partial and may require additional modification or tinkering to solve your specific concern. You may spend just as much time trying to get a Stack Exchange solution to work as you would waiting for a response on Slack, so proceed with caution if you decide to go this route.

Finally, if you have exhausted these other options, a Google search may be effective. This may help you identify GitHub issues, blogs, and other online tutorials that can help address whatever roadblock you have run into. Try searching first with double quotes around the main body of the error message's text. If nothing comes up, try searching without the double quotes. Search strings that include R package names or specific processes in ArcGIS are sometimes helpful for narrowing down a large amount of search results, especially if those results are not specific to the tool you are using. The same warning for Stack Exchange also exists for Google, however. You may find imperfect matches for your problem that take considerably more tinkering to implement.

## 5.2 How to Seek Help

If all else fails and the strategies for Helping Yourself have not yielded a solution, it is time to ask for help! Before you head to office hours or Slack, you will want to construct a **reproducible example**.

### 5.2.1 What is your question?

Start with a basic step - narrow down what your question is. Neither “I am getting an error” or “This isn’t working” are effective questions. What exactly is causing the error? What context does it appear in? To borrow from the examples below, a good question might be:

I am trying to calculate descriptive statistics for an entire data frame using the `mean()` function but am getting an error. What might be the cause of the error?

I am trying to create a thematic choropleth map in ArcGIS, but when I try and select my variable from the dropdown menu in the symbology tab under layer properties, the menu is empty. Why are all of the variables missing?

I am trying to make my text bold in LaTeX, but the text is not rendering with bold font. I am also getting an error that says “Undefined control sequence.” What is missing from my document to create bold font?

Be specific about what your issue is, and then provide a reproducible example that illustrates what you are asking about.

### 5.2.2 What is a reproducible example?

A reproducible example, or *reprex*, is a term we’ll borrow from the R ecosystem. It was coined by Roman François and has been enshrined in the **reprex** package, which I’ll describe below. The goal for a *reprex* is to strip down the process you have used to the minimal number of steps needed to replicate the error. This means using the fewest steps and data sources possible. Cut out anything that does not directly contribute to causing the error when making your *reprex*.



I have often found that the process of making a reprex actually helps isolate the cause of the problem. For instance, it may become clear that a specific step is causing an issue. Even if you are not sure how to fix the problem, having narrowed it down can be immensely helpful. If you can use built-in data in R to create the reprex, or at least example data from the course, that may also help you identify whether the issue is with the process itself or something that is idiosyncratic to the data you are using. Using built-in data from R should be the default for any reprex you create, *unless assignment data are the cause of your issue*.

### 5.2.3 Creating a reprex

Creating reproducible examples will differ slightly based on what you are trying to get help on.

#### 5.2.3.1 R

To create a reprex in R, you will need to install the **reprex** package. It is part of the tidyverse but it is not installed when the **tidyverse** package is installed, so you will need to install it separately:

```
install.packages("reprex")
```

Once you have it loaded (use the **library()** function), create a minimal example of the necessary functions that get to you to the error you are confronting. Make sure to include the **library()** functions for all packages your example depends on (except for **reprex**). Write the example in a **.R** script file (**File > New File > R Script**). For example, one might be struggling with calculating the mean of each variable in a data frame:

```
library(ggplot2)

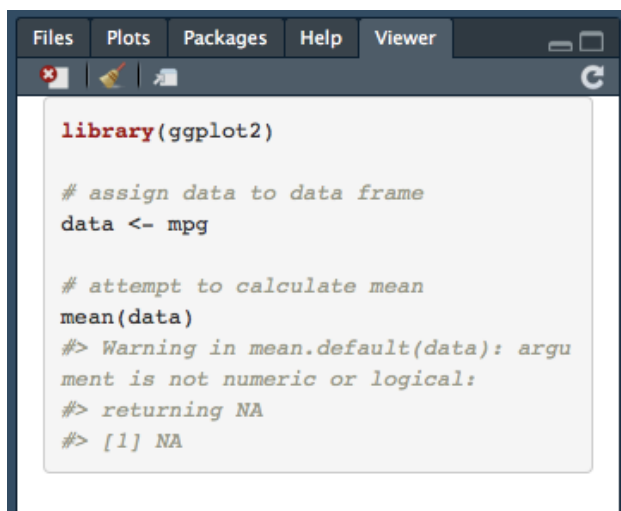
# assign data to data frame
data <- mpg

# attempt to calculate mean
mean(data)
```

The last of the three functions above will return this error in your R session:

```
> mean(data)
[1] NA
Warning message:
In mean.default(data) : argument is not numeric or logical: returning NA
```

With the three functions written in a **.R** script, highlight all seven lines of code (the three functions, both comments, and both white space lines) and copy them to your clipboard (**Edit > Copy**). Then call the **reprex()** function. Markdown formatted text that weaves both the functions and their output together will appear in the viewer tab:



```
library(ggplot2)

# assign data to data frame
data <- mpg

# attempt to calculate mean
mean(data)
#> Warning in mean.default(data): argument is not numeric or logical:
#> returning NA
#> [1] NA
```

It will also be available on your clipboard so that you can copy and paste it into Slack or another venue that accepts Markdown syntax (like GitHub). If your reprex contains image output, such as a plot or a map, it will be automatically uploaded to imgur and a link will be embedded in your Markdown syntax. For example, say we had a question about the following code:

```
library(ggplot2)

# assign data to data frame
data <- mpg

# plot highway mpg
ggplot(data = data, mapping = aes(x = hwy)) +
  geom_histogram()
```

Once we copy the above code and render it using `reprex()`, it will return the following output:

```
library(ggplot2)

# assign data to data frame
data <- mpg

# plot highway mpg
ggplot(data = data, mapping = aes(x = hwy)) +
  geom_histogram()
#> `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.


```

The `![]` (hyperlink) syntax will allow an image of your plot to appear below the code.

The `reprex` package website has some great resources, including a basic overview, some reprex do's and don'ts, and a detailed article introducing the package's functionality.

### 5.2.3.2 ArcGIS

Creating a reprex in ArcGIS is not as straightforward since there is not a dedicated tool for doing so like there is in R. Reprex creation is complicated by the fact that many steps you will be taking are manual. Nevertheless, it is possible to create a reprex. Follow these steps:

1. In a new map document, load the minimum number of shapefiles needed to illustrate the issue you are having. Note where the shapefiles are available (in the course data release?) and what their names are.
2. Note what the coordinate system of the data frame is set to.
3. Provide notes for each step that gets you to your question or error.
4. Take a screenshot of the error or the window you have a question about.

For example, a reprex of a question in ArcGIS could look like the following:

1. I have a single map shapefile open in a new map document. The shapefile is the St. Louis census tracts.
2. The coordinate system is Missouri State Plane East (Feet)
3. When I right click on the shapefile, I click on Properties and then the Symbology tab
4. Attached is a screenshot of what the symbology tab looks like for these data.

### 5.2.3.3 LaTeX

Like ArcGIS, there is no specific tool for creating a reprex using LaTeX. The LaTeX community does have, however, a long tradition of creating “minimal working examples”, which are similar to reprexs. Follow these general steps:

1. Create a new LaTeX project. Use the `\documentclass{article}` if you are using something different.
2. Only include packages in the preamble that are **directly** related to the question or issue you have.
3. Skip creating a title block
4. Provide just enough body text in your LaTeX document to illustrate the issue.

A reprex for LaTeX might look something like this:

```
\documentclass{article}

\begin{document}

\textbf{Foo bar.} Spam and eggs.

\end{document}
```

Also provide a written description of what is wrong (such as “the text is not being rendered as bold”) and, if helpful, a screenshot of your output. There is a great overview on Stack Exchange of how to write a reprex for LaTeX. If you are using ShareLaTeX, also take a look at the error log and take a screenshot of the relevant error message. If you are using another LaTeX application, provide the text of your error message instead.

### 5.2.3.4 Other Tools

For other tools we learn, such as Markdown and GitHub, you want to follow the spirit of the reprex file. Try to recreate your issue outside of the context of your assignment (this may not be possible for GitHub), and provide a detailed walk-through of the steps that you took to get where you are. For Markdown syntax, provide an example of the syntax you are using that recreates the issue or question. For GitHub, provide a screenshot of the relevant error message.

## 5.2.4 “I don’t think I can make a reprex!”

I am 99% sure that you can! In nearly every situation I have seen students in, creating a reprex is possible. Even if the error is idiosyncratic to your computer or your data, you can absolutely clarify the context within which the error appears and minimize the amount of data, code, and other information in your R code or your map document. For the less than 1% of scenarios where a reprex is not possible, the process of writing

a question, clarifying the context and steps you took to get there, and producing an example of the error will still make it easier for me to help you.

### 5.2.5 “Isn’t this a lot of work?”

Well, yes, it does require some extra effort. This effort is almost always worth it, however. In some cases, the time it takes you to produce the reprex leads you to the answer on your own, which is part of the problem solving process that this class is designed to foster. Even when this does not happen, making reprex informed inquiries is a technique that you will be able to take with you at the end of the semester. Even if you are not working in a technical setting, being able to structure clear, concise questions about a process is a valuable skill!

Finally, creating a reprex saves you time in the long run. When I get vague questions, it often takes some experimenting on my part to reproduce the error. If you send me a question with a reproducible error, you cut out that experimentation time on my end and I can get right to answering your question. Likewise, students often come to office hours without an example of their issue ready to go and hope that I can conjure in my mind the scenario they are describing. Despite my best efforts, I am usually unable to do so and ask students to reproduce the error during office hours. If you come to office hours with a reprex, we can get to answering your question right away!

## 5.3 Where to Seek Help

Once you have tried Helping Yourself and have gone through the steps outlined in How to Seek Help (i.e. made a reprex), it is time to ask your question. There are a number of different places that you can pose questions. The sections below outline these and suggest some ways to make your question most effective.

### 5.3.1 Internal Venues

Within the confines of the course, the two best places to ask questions are on Slack and during office hours. When asking a question on Slack, please consider doing so in a public channel. This helps others learn from the question that you have. The only course-related questions that are not appropriate for Slack are those that reference specific parts of a problem set or the final project. However, if you are creating reproducible examples, your questions should not be that specific even if they ultimately are about a homework assignment. If you are not sure whether the question is appropriate or not for a public channel, or just would prefer not to ask your question out in the open, feel free to send me a direct message on Slack.

When you pose a question on Slack, it should contain three or four pieces of information:

1. The question itself - what are you hoping to get help with?
2. A reproducible example - give as many details as you need to clearly illustrate the question
3. A screenshot or image output that illustrates your reprex
4. What you’ve tried - lay out the steps you took to quickly (“I’ve checked the spelling, gone through the Lecture 03 documents, looked at the `dplyr` website, and have done a quick Google search.”)

Post your reprex as a snippet in Slack by:

1. Clicking the plus sign on the left side of the message box
2. Selecting **Code or text snippet**
3. Filling out the pop-up window

If you are posting a reprex from R, remove the three backticks and the letter `r` from the first line as well as the three backticks from the bottom line. If you are including an image hyperlink created by the `reprex` package, remove that from the code snippet and paste it into Slack separately. A good question on Slack should look like this:



**Chris Prener** 6:11 AM

I am trying to figure out why I am getting an error message with my histogram that says to "pick better value with `binwidth`". What is causing that error?

I've included a reprex here that reproduces this:



**Chris Prener** 6:11 AM

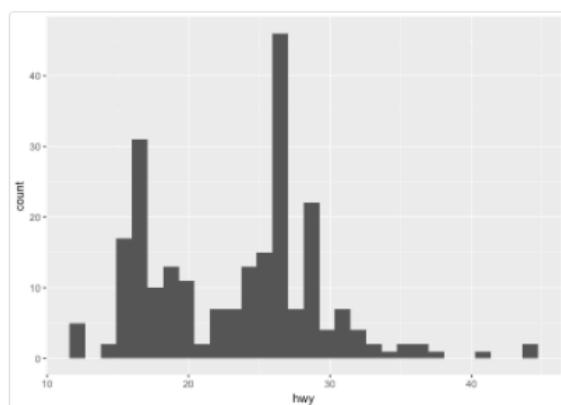
added this Plain Text snippet ▾

```
1 library(ggplot2)
2
3 # assign data to data frame
4 data <- mpg
5
6 # plot highway mpg
7 ggplot(data = data, mapping = aes(x = hwy)) +
8   geom_histogram()
9 #> `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



**Chris Prener** 6:11 AM

<https://i.imgur.com/VmCLq7w.png> (17 kB) ▾



I don't see a reference to `binwidth` in the lecture or the handouts, and cannot find a reference to it on Google.

This same general process holds for asking questions in person - come ready with a solid question, a reproducible example, and be ready to pull it up on your computer or a lab computer.

### 5.3.2 External Venues

There are two good places to ask questions external to the course as well. For questions related to RStudio, R Markdown, and the tidyverse, RStudio maintains a web forum called RStudio Community. If you are posting there, you can use `reprex()` to generate Markdown formatted reprexes.

Another place to post questions in on one of the relevant Stack Exchange communities:

- Stack Overflow for R, Git, and GitHub - post using tags like `[r]`, `[ggplot2]`, `[dplyr]`, `[git]`, `[github]`, etc.
- Geographic Information Systems for GIS - post using the `[arcgis-desktop]` tag
- TeX for LaTeX

Before you post, search thoroughly to make sure that your question has not already been answered. Stack Exchange has strong norms about how to post appropriately. You can descriptions of what makes a good post here and here. You can also find a description of what not to ask. If you are going to post on Stack Overflow, the `reprex()` function can be modified to produce well-formatted output specific to that site by adding the `venue` argument, as in `reprex(venue = "so")`.

Of the two sites, I find RStudio Community to be a friendlier place that is more relaxed than Stack Overflow and its cousins. However, Stack Exchange communities have much larger user bases to draw from.



# **Part II**

## **Data Science Toolkit**





## Chapter 6

# Opinionated Tools

The Introduction to this text describes opinionated analysis development as a key element that both Introduction to Geographic Information Science (SOC 4650/5650) and Quantitative Analysis: Applied Inferential Statistics (SOC 4930/5050) are built around. Parker's (2017) term is a reference to opinionated software, which is designed with a strong set of guiding principles that strongly encourages users to follow certain processes (Eccles 2015). When possible, we'll use opinionated applications or processes that support our larger goals of opinionated analysis development. This section introduces a number of the key tools that we use to accomplish data science work with social and spatial data.



# Chapter 7

## Markdown

Markdown is a simple markup language. Markup languages are used to give computer programs directions on how particular blocks of text should be processed. Markdown was developed in 2004 by writer and developer John Gruber. Gruber describes Markdown on his website:

Markdown is intended to be as easy-to-read and easy-to-write as is feasible.

Readability, however, is emphasized above all else. A Markdown-formatted document should be publishable as-is, as plain text, without looking like it's been marked up with tags or formatting instructions. While Markdown's syntax has been influenced by several existing text-to-HTML filters — including Setext, atx, Textile, reStructuredText, Grutatext, and EtText — the single biggest source of inspiration for Markdown's syntax is the format of plain text email.

To this end, Markdown's syntax is comprised entirely of punctuation characters, which punctuation characters have been carefully chosen so as to look like what they mean. E.g., asterisks around a word actually look like *emphasis*. Markdown lists look like, well, lists. Even blockquotes look like quoted passages of text, assuming you've ever used email...

...Markdown's syntax is intended for one purpose: to be used as a format for writing for the web.

Markdown is utilized to varying degrees by many of the data science tools we'll use in both courses, including RStudio, GitHub, and Slack. In all three applications, Markdown provides a straightforward way to format and stylize plain text files. Markdown therefore gives us the best of two worlds - text files can be organized and formatted as in WYSIWYG editors like Microsoft Word, but they remain plain text files that are accessible in a variety of computing environments and do not depend on proprietary applications.

### 7.1 Document Organization

There are two principle means for organizing Markdown documents - headings of varying size and paragraph breaks.

#### 7.1.1 Headings

Markdown contains six heading levels. Headings are identified with # symbols and a space that separates them from the heading text.

**Input:**

```
# This is the largest heading
## This is the second largest heading
### This is the third largest heading
#### This is the fourth largest heading
#### This is the second smallest heading
##### This is the smallest heading
```

Output:

# This is the largest heading

## This is the second largest heading

### This is the third largest heading

#### This is the fourth largest heading

#### This is the second smallest heading

##### This is the smallest heading

### 7.1.2 New Paragraphs

Leave a blank line between two paragraphs to create a break.

Input:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Quam pellentesque nec nam aliquam sem et tortor consequat. Auctor urna nunc id cursus metus. Eleifend mi in nulla posuere. Facilisis sed odio morbi quis. Nibh nisl condimentum id venenatis. Lectus nulla at volutpat diam ut venenatis.

Pretium lectus quam id leo in vitae. Neque vitae tempus quam pellentesque nec nam aliquam. Curabitur gravida arcu ac tortor. Arcu risus quis varius quam quisque id diam. Viverra aliquet eget sit amet tellus cras adipiscing. Tellus molestie nunc non blandit massa enim nec. Tempus imperdiet nulla malesuada pellentesque elit eget. Non blandit massa enim nec. Sed risus pretium quam vulputate dignissim suspendisse in est ante.

Output:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Quam pellentesque nec nam aliquam sem et tortor consequat. Auctor urna nunc id cursus metus. Eleifend mi in nulla posuere. Facilisis sed odio morbi quis. Nibh nisl condimentum id venenatis. Lectus nulla at volutpat diam ut venenatis.

Pretium lectus quam id leo in vitae. Neque vitae tempus quam pellentesque nec nam aliquam. Curabitur gravida arcu ac tortor. Arcu risus quis varius quam quisque id diam. Viverra aliquet eget sit amet tellus cras adipiscing. Tellus molestie nunc non blandit massa enim nec. Tempus imperdiet nulla malesuada pellentesque elit eget. Non blandit massa enim nec. Sed risus pretium quam vulputate dignissim suspendisse in est ante.

## 7.2 Working with Text

Markdown provides a number of tools for working with and stylizing text. These include options for bold and italicized text, adding code blocks, quoting text, and creating bulleted and enumerated lists.

### 7.2.1 Styling Text

Text can be styled using bold and italics. To create italicized text, wrap your text with a single asterisk \*. To create bold text, wrap your text with double asterisks \*\*.

**Input:**

```
*This is an italicized sentence.*
**This is a bolded sentence.**
```

**Output:**

*This is an italicized sentence.*

**This is a bolded sentence.**

### 7.2.2 Quoting Text

Quoting text (which I have used above to illustrate examples) is done with a greater than symbol (>).

**Input:**

```
> This is quoted text.
```

**Output:**

*This is quoted text.*

### 7.2.3 Quoting Code

There are two types of code quotes in Markdown. In-line quotes, which are included in a sentence, are wrapped in single backticks: > Use the **describe** command to list variables in R

To include code blocks, which are better for including the full syntax of particular commands and their output, use triple backticks:

```
. describe make price mpg
```

variable name	storage type	display format	value label	variable label
make	str18	%-18s		Make and Model
price	int	%8.0gc		Price
mpg	int	%8.0g		Mileage (mpg)

Note how the word ‘Stata’ is written after the first set of triple backticks. This is an indicator for GitHub that the code is written in Stata’s programming language. By including this, GitHub can apply some syntax highlighting to your files. This makes them easier to read.

### 7.2.4 Links

In Markdown, adding hyperlinks is a two step process. The text that you want to have hyperlinked is written first and is wrapped in brackets []. After this, you include the URL wrapped in parentheses (). This is an example of including in-line hyperlinks:

The course website is hosted using the service GitHub.

### 7.2.5 Embedding Images

Within the directory that contains your Markdown file, create a subdirectory called `Output`. Save all images for a particular assignment there. In your main assignment Markdown file, include a hyperlink reference:

Note how, instead of including text to be hyperlinked, we suppress this aspect of the syntax by using an exclamation point !.

### 7.2.6 Simple Lists

Bulleted lists are indicated in Markdown using the dash - or a single asterisk \*:

- mean
- median
- mode
- variance
- standard deviation

Enumerated lists are created by preceding each line with the appropriate number:

1. calculate the mean
2. calculate the variance
3. calculate the standard deviation

You can create more complex lists by preceding a line with two single spaces. You can also combine bulleted and enumerated lists when using this approach.

## 7.3 GitHub Markdown

These are specific to what is called GitHub Markdown - there are some subtle differences in the way GitHub uses Markdown formatting.

### 7.3.1 Styling Text

Text can be styled using bold, italics, and strikethroughs. To create italicized text, wrap your text with a single asterisk \* \*. To create bold text, wrap your text with double asterisks \*\* \*\*. To create strike-through text, wrap your text with two tildes ~ ~.

~~This is a sentence with strikethrough text~~

### 7.3.2 Tables

### 7.3.3 Task Lists

If you want to create task lists on GitHub, you can include brackets separated by a space before each list item [ ]. Completed tasks include an x in place of the space [x]. These task lists are interactive - when published on GitHub, you can click on the resulting checkboxes to toggle them between complete / incomplete.

1. [x] calculate the mean
2. [ ] calculate the variance
3. [ ] calculate the standard deviation

### 7.3.4 Mentioning Other GitHub Users

If you want to mention me or one of your classmates in a comment, include the @ symbol before their username:

Hey ?, thanks for the feedback. I made the changes to lines 40 and 41.

Once the document is uploaded to GitHub, the username will render as a hyperlink and the user will be alerted.





# Chapter 8

## Basic Git

Much of our interaction this semester outside of class will utilize GitHub.com (or just “GitHub”). GitHub is a web service that is a social network for programmers, developers, data scientists, researchers, and academics. It is also a tool for collaborating on projects, especially projects that involve writing code.

We’ll use GitHub as an alternative to Blackboard, the *course management system* that students are typically familiar with. Course materials will be posted there, and GitHub’s features will allow you to copy them and keep them updated as changes are made. You’ll also use GitHub to submit assignments for grading, and we’ll give you feedback and grades via GitHub as well.

### 8.1 Git Basics

#### 8.1.1 Git

GitHub is a web application that utilizes Git:

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Essentially, Git is a project-wide system for tracking changes to files. Think of it as Microsoft office’s track changes feature on steroids - every change to every file in a directory is tracked. A directory that has tracking enabled with Git is called a **repository** or “repo” in Git-lingo.

You do not need to host files online to use Git. If you have a project saved locally (say, a doctoral thesis), you could utilize Git to version control that project without ever uploading it to the Internet. However, Git integrates seamlessly with **remote** repositories. The most well-known host of remote repos is GitHub.com.

Beyond “repositories”, there are a few additional terms that are specific to Git and that are helpful to know:

- **Clone:** Make an identical copy of a repository on your local hard drive.
- **Commit:** Approve any changes you have made to a repository.
- **Sync:** For cloned repositories, files that have been changed need to be synced or **pushed** to GitHub.com after they are committed.

For our purposes, this is just about all you need to know about Git. If you want to learn more, Git’s ‘About’ page is a great place to start.

### 8.1.2 GitHub.com

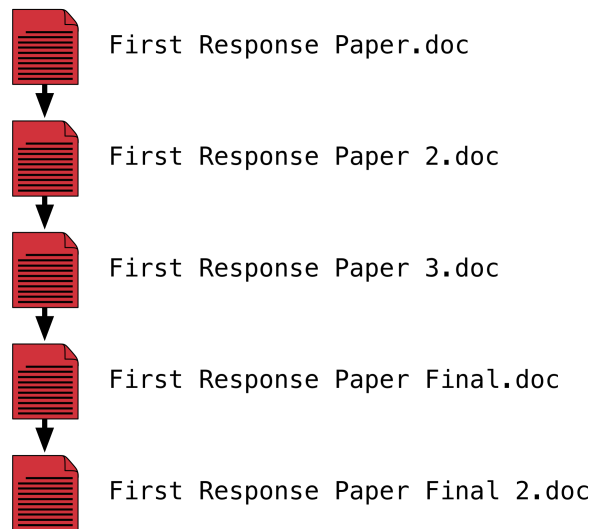
GitHub is a web service that can host projects using Git’s version tracking. It is widely used by programmers, software developers, data scientists, and academics to host and collaborate projects.

GitHub is an excellent way to backup files for a project since you can “sync” changes made to a repository up to GitHub’s servers. It is also an excellent way to collaborate on files with colleagues while also using Git’s version tracking. Repositories can be either public (like all of the repos for our seminar) or private, which means that only people who have been given access to can view the contents of the repo. Private repos require an upgraded account, which retails for \$7/month.

Students can get access to GitHub’s paid services for free, however, by signing up for a free student account. This will give you access to private repositories for as long as you are a student. GitHub also provides this free upgrade service to educators, which is how our classes have private repositories.

## 8.2 The Workflow of Git and GitHub

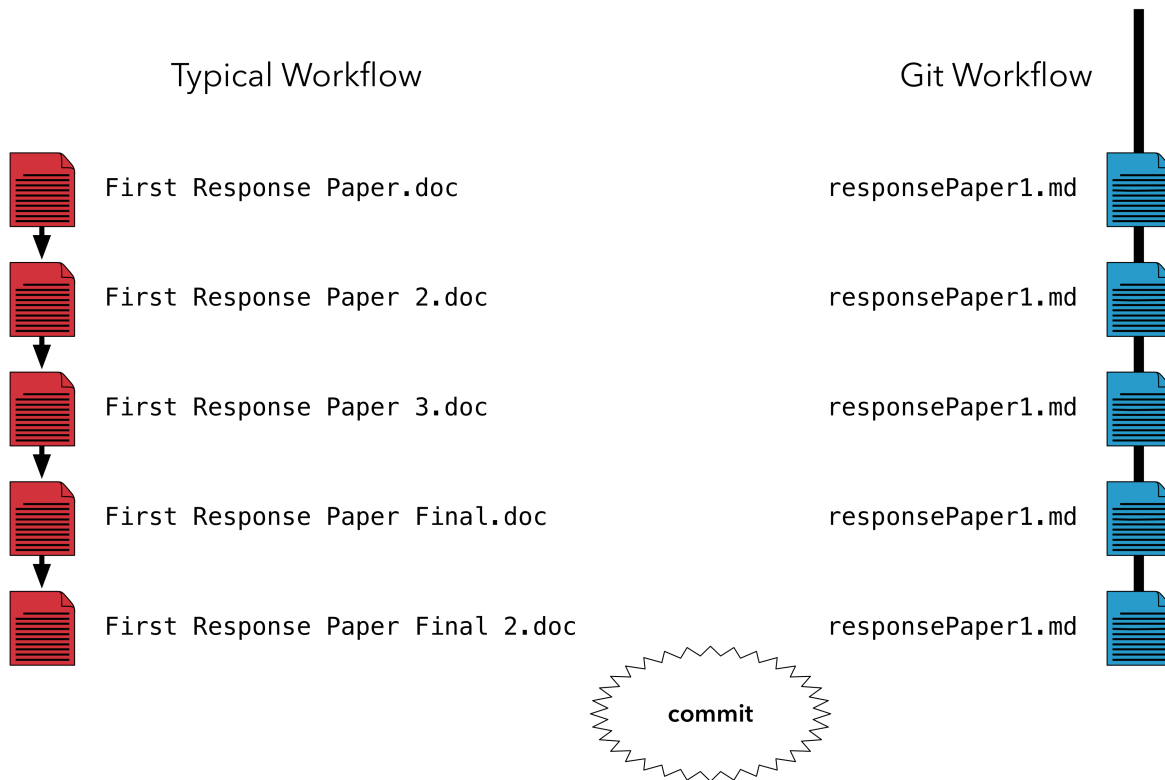
The typical approach to versioning for many students is manual. For a hypothetical class response paper, it might look something like this:



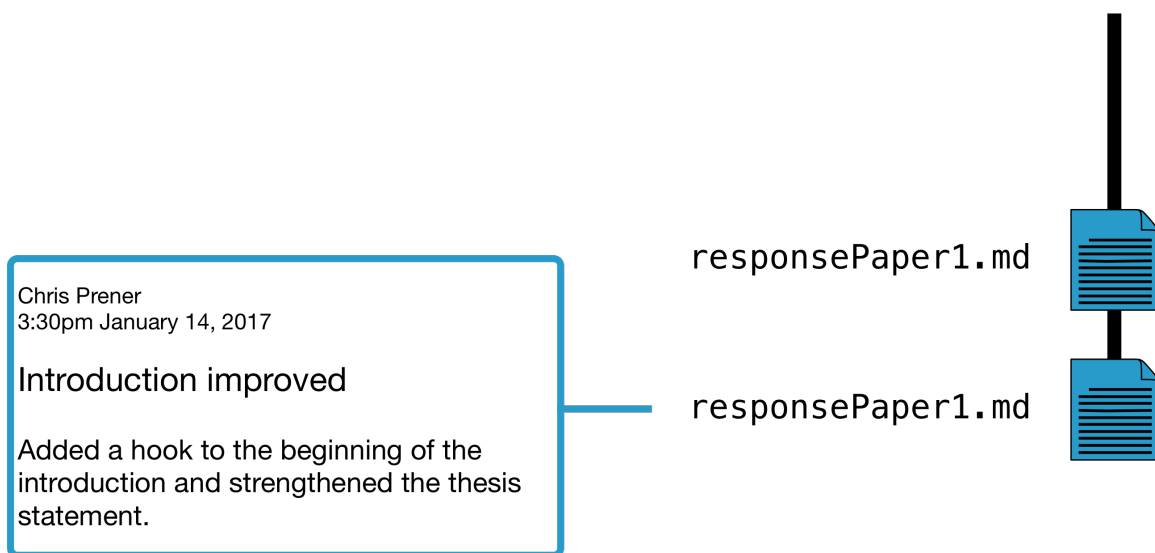
The author made an initial copy of the paper, and then used a haphazard and inconsistent approach for naming subsequent copies of the paper. We can presume that changes were made in a linear fashion, though it is easy to make changes to, say, `First Response Paper 2.doc` after `First Response Paper 3.doc` has been created and edited.

### 8.2.1 Commits

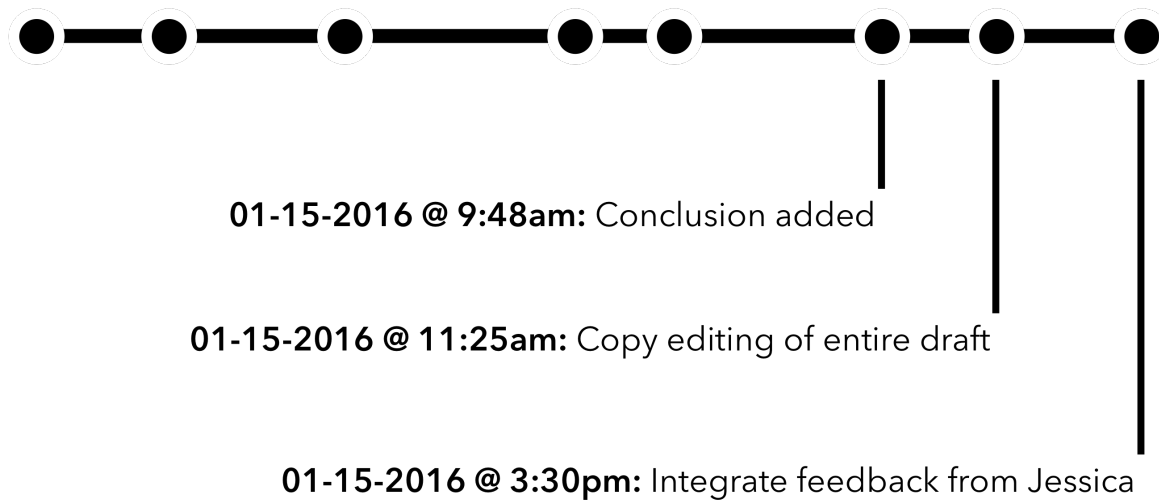
Instead of saving copies of their hypothetical paper, a student using GitHub could write the paper in a single document, **committing** their changes as they progress to take “snapshots” of their progress. These snapshots contain information on changes the student has made, tracked line-by-line. So, at each point in which a new document would have been created in the typical workflow, a student using GitHub would simply **commit** their changes:



Git provides a number of useful features beyond simply tracking changes. Each commit is accompanied a **message**. These messages must have a short summary that appears on GitHub and can also have a longer description that can be used to describe in detail what changes are being applied with a specific commit:



Messages, combined with the changes that are tracked, allow users to trace the development of a single document or an entire project overtime:



This means that, if necessary, the project can also be rolled back to an earlier period. Finally, users can **sync** their commits with GitHub.com, hosting their changes and their data in a way that protects them against certain types of computer failures and also allowing them to easily share their work with others.

## 8.2.2 More About Repositories

Users of GitHub.com adhere to a couple of norms with their repositories that are worth knowing about. Repositories cannot have spaces in their names (much like objects in R), so the naming conventions that we will discuss in relation to R this semester all apply to GitHub as well!

Public GitHub repositories also contain (typically) at least three core files:

1. A **license** file - since the data is out there for public consumption, it is important to think about how that data is licensed. The norm among GitHub users has been to use open source licenses, which let others edit and adapt your work. There are a range of licenses that are commonly used on GitHub.
2. A **README** file - this describes the purpose and content of the project.
3. A **.gitignore** file - this stops certain types of files from being swept up by GitHub when a user syncs their files with a server.

When you clone your repositories, you will be prompted to save them on your computer. There are a number of ways in which this process can introduce sources for trouble down the road. The principle way that I have seen students run into problems with GitHub is by storing repositories on cloud storage services like Dropbox or Google Drive. In order to avoid any issues, I advise against storing GitHub repositories in an area of your computer that syncs with a cloud service.

## 8.3 GitHub Issues

GitHub has a powerful tool for interaction called Issues. These can be accessed by opening a repository and then clicking on the “Issues” tab. Issues can be “opened” by anyone with access to the repository. They allow for a conversation to occur in the form of messages posted within the Issue itself. Files can be attached to Issues, and the messages can contain Markdown formatting. Once the conversation is complete, issues can be marked as “closed”, which moves them into a secondary view on the website so that they are archived.

We’ll use issues for both assignment feedback and grading. Please keep up with issues as they appear, and feel free to follow-up with specific questions about your grade or the assignment feedback in the Issue

conversation. Once you are satisfied, please mark the issue as closed.

## 8.4 GitHub Desktop Application

GitHub Desktop is a tool that allows you to easily clone repositories hosted on GitHub, commit changes to them, and then sync those changes up to the website. You can also create new repositories, however this is not task you will have to do this semester. GitHub Desktop is not a fully functional desktop version of GitHub. For our purposes, it is important to note that the Desktop application will not let you easily identify when repositories have been updated by other users, view Wikis associated with repositories, or view Issues.

## 8.5 Learning More

GitHub has a resources page with links to websites that are great for helping you learn more about how Git and GitHub work! The next chapter also has some additional GitHub and Git information.

One particularly great tutorial walks you through the command line process for creating and using a git repository. Even if you do not want to use Git via the command line, the tutorial does an *excellent* job of describing the logic and sequence behind the Git workflow.



## Chapter 9

# Advanced Git

GitHub has a number of tools available for managing multiple contributors' work simultaneously. These tools will be useful for collaborating with your teammates on the final project.

### 9.1 Even More Git-lingo

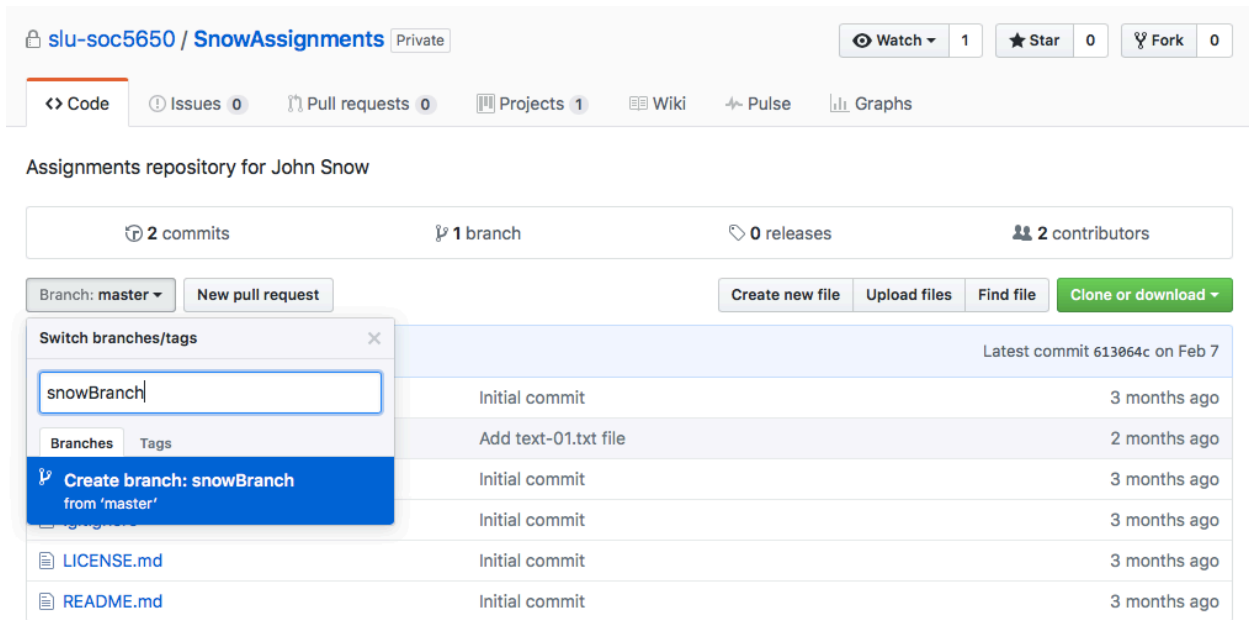
Beyond the terms introduced in the previous chapter, there are a few additional terms that are specific to collaborating that are helpful to know:

- **Branch:** An identical copy of the repository that is distinct from the `master` copy.
- **Checkout:** The act of switching to a new branch.
- **Pull Request:** Request that the changes made on a branch be integrated into the `master` copy.

### 9.2 Creating Branches

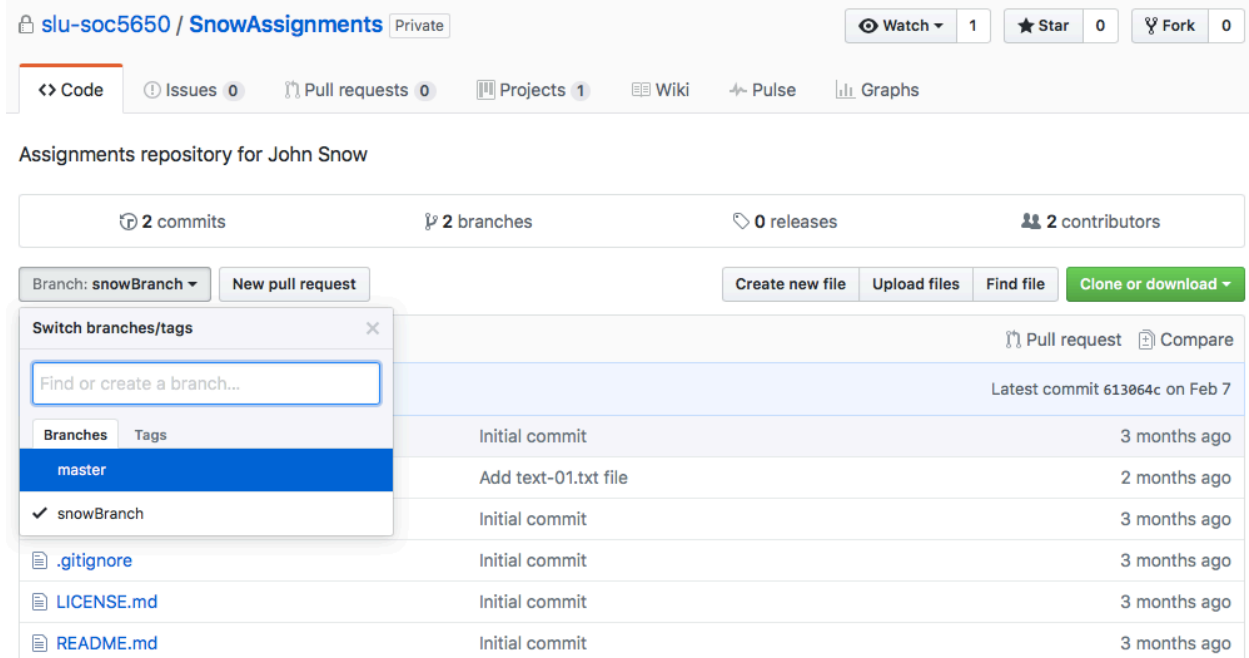
The logic behind branches is that they allow you to make modifications to the content of the repository without impacting the content that is on the `master` branch. Think of the `master` branch as your production data - this is what you know is good and in working order. If you need to experiment, say by adding new code or data sources, you can do this on a branch without worrying about breaking something in the production part of the repository.

To create a new branch, navigate to the repository in question on GitHub.com. Just above the list of the repository's contents on the lefthand side of the window is a button that will read "Branch: master". If you pull the associated menu down, you have the ability to create a new branch (by typing a name into the text field) or switch between branches:



The image above shows this menu pulled down with the fictitious user John Snow creating a new branch on his assignments repository named **snowBranch**. Once the branch is created, you will automatically be directed to the new branch's contents. These should look identical to the **master** branch at this point because the branch creation process generates a mirror image of **master**.

You can use the same pulldown menu to switch back to the **master** branch if needed:



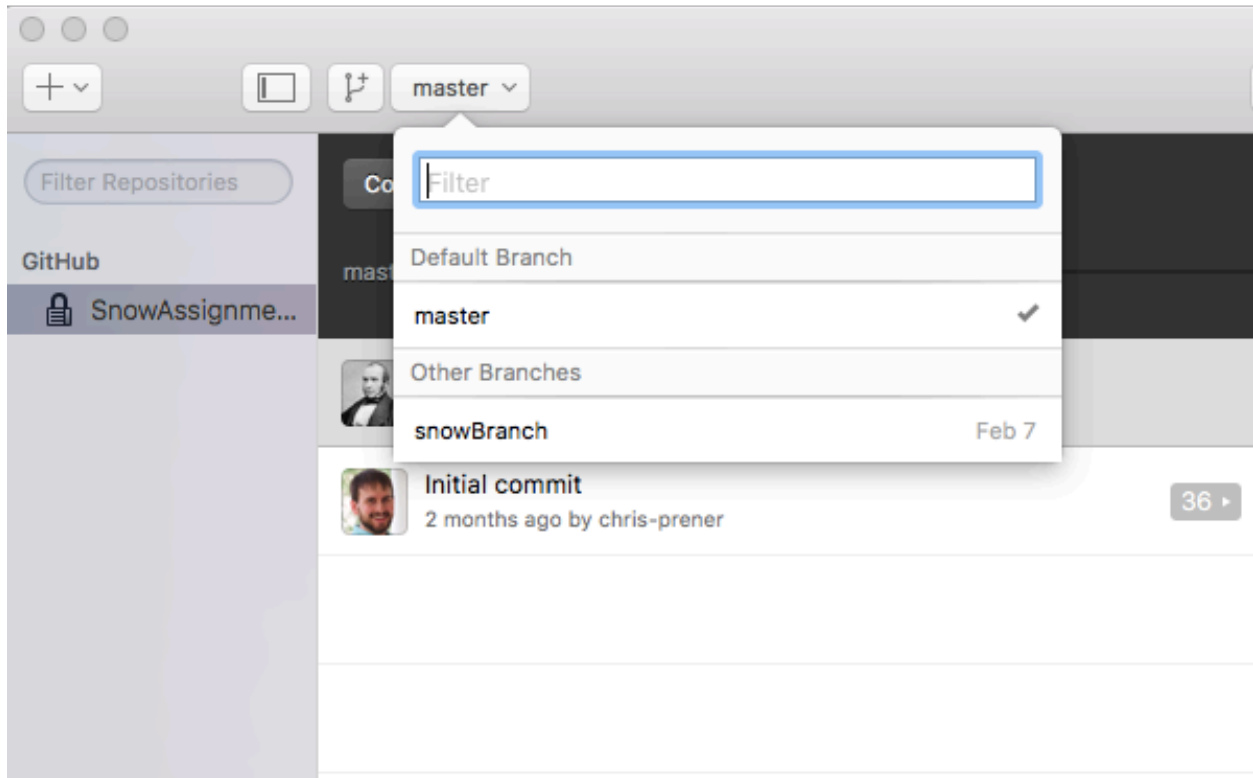
### 9.2.1 Working with Branches

Once your branch is created, you need to open up GitHub desktop and, if you haven't already, clone the repository you created the branch in. If you have already cloned the repository, you just need to **Sync** it so



that the branch information is downloaded to your local copy of the repository.

Once you have updated or cloned your repository, it is time to **checkout** the branch. This means to switch from the current branch (by default, the **master** branch, to your new branch). In GitHub Desktop, there is a button on the top toolbar that will say the current branch name (by default, **master**). You can click on this button to open a pulldown menu that will list all available branches:



Click on a branch to check it out. Once checked out, the files available to your operating system will change to reflect the contents of that branch!

It is important that you ensure that you are always intentionally working on the correct branch. Any time you return to your computer to continue working on a branch, open up GitHub Desktop and ensure that the proper branch remains checked out.

Branches are used just like regular repos in Git. You can commit changes to them and sync them with GitHub.com. I recommend committing changes to branches regularly and syncing them with the remote version of the repository to ensure that your work is not lost.

### 9.2.2 Opening Pull Requests

When you have made changes to your branch and want to integrate them into the **master**, production part of your repository, it is time to open a **pull request**. Below, you will see that a directory named **ExtraWork** has been added to the **SnowAssignments** repository's **snowBranch**:

The screenshot shows the GitHub repository page for 'slu-soc5650 / SnowAssignments'. The repository is private and has 1 Watch, 0 Stars, and 0 Forks. The main navigation bar includes links for Code, Issues (0), Pull requests (0), Projects (1), Wiki, Pulse, and Graphs. Below the navigation bar, the repository is described as 'Assignments repository for John Snow'. It shows 3 commits, 2 branches, 0 releases, and 2 contributors. A 'Branch: snowBranch' dropdown is visible, along with a 'New pull request' button. A green 'Clone or download' button is also present. The repository content shows a list of files and folders: 'ExtraWork' (Add extra work file, 23 seconds ago), 'FinalProject' (Initial commit, 3 months ago), 'Labs' (Add text-01.txt file, 2 months ago), 'ProblemSets' (Initial commit, 3 months ago), '.gitignore' (Initial commit, 3 months ago), 'LICENSE.md' (Initial commit, 3 months ago), and 'README.md' (Initial commit, 3 months ago). A message at the top states 'This branch is 1 commit ahead of master.' and a 'Pull request' button is visible.

We have verified that the file is complete and ready for integration into the **master** branch. To do this, we need to click on the **New pull request** button just to the right of the **Branch** button above your repository contents. Make sure that you are opening the pull request on your branch and not on the **master**!

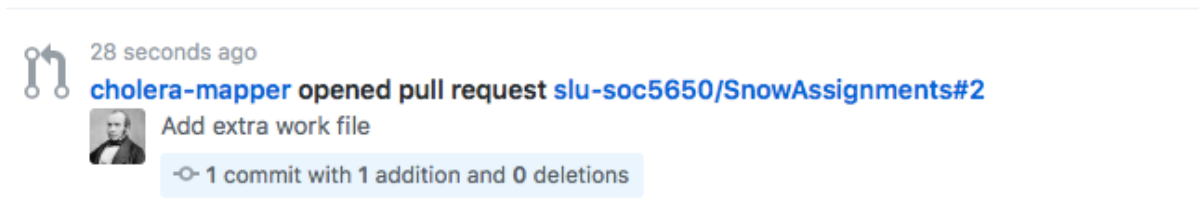
The screenshot shows the 'Open a pull request' page in GitHub. The repository is 'slu-soc5650 / SnowAssignments'. The 'base' branch is 'master' and the 'compare' branch is 'snowBranch'. A green checkmark indicates 'Able to merge. These branches can be automatically merged.' The main content area has a title 'Add extra work file' and a description 'The extra work file contains important details about the assignment.' There are tabs for 'Write' and 'Preview'. A green 'Create pull request' button is at the bottom right. On the right side, there are sections for 'Reviewers' (No reviews—request one), 'Assignees' (No one—assign yourself), 'Labels' (None yet), 'Projects' (None yet), and 'Milestone' (No milestone).

The **pull request** page looks very similar to GitHub's **Issue** feature - there is a place for a title (which will

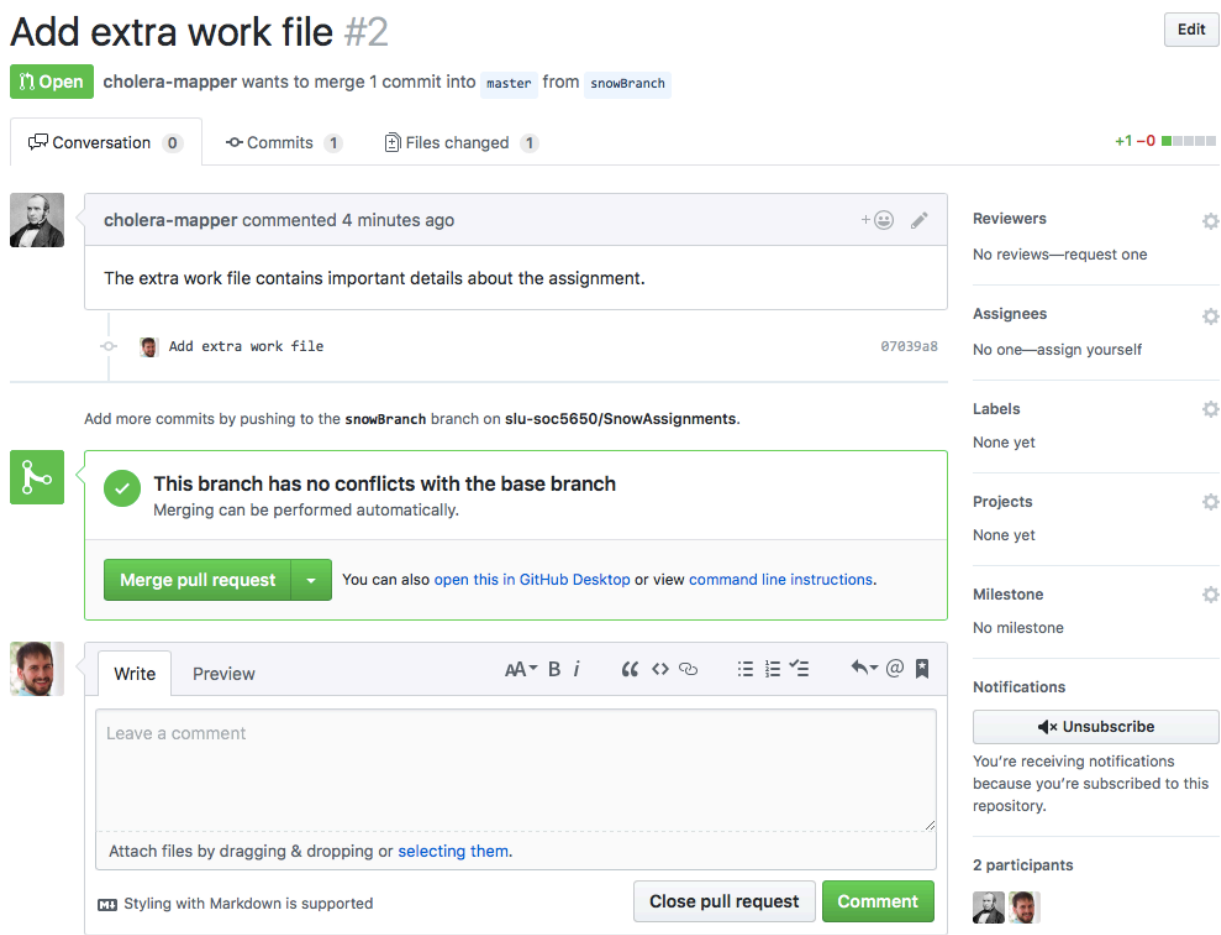
be autofilled for you) and for describing the contents and significance of your changes. You can optionally assign individual reviewers (such as your final project team members). Once you are done, click the green **Create pull request** button.

### 9.2.3 Evaluating Pull Requests

When a team member opens a **pull request**, you should see a notification in your news feed on the main landing page when you log into GitHub.com:



Clicking on that message (or, alternatively, selecting the **Pull Requests** tab in any repository) will take you to a page where you can evaluate changes that were made on your colleague's branch:

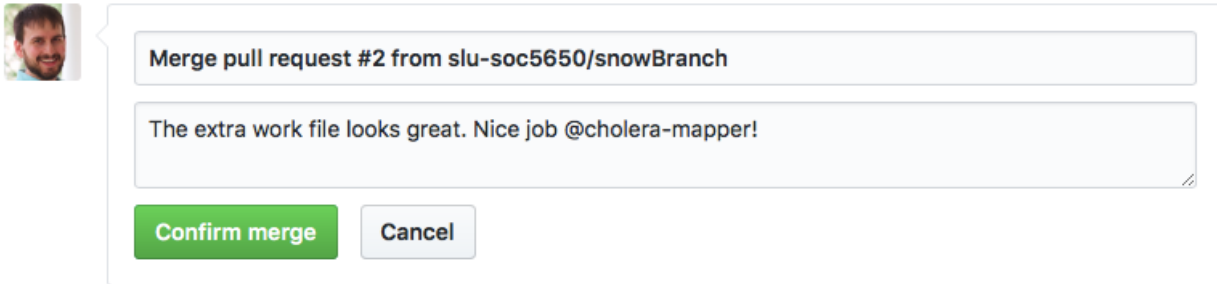


Here you can: \* view files that have been changed or added, \* respond to the pull request with a comment if you have follow-up questions, \* view the results of GitHub's conflict check, \* and choose an outcome (either merging - i.e. accepting - the pull request or closing - i.e. rejecting - the pull request)

This is an integral step. It is part of the peer review process that drives open-source software and is conducted in the same spirit of peer review that academics use to evaluate publications. The goal here is provide constructive feedback about proposed additions or changes. If you choose to close (i.e. reject) a pull request, give a detailed explanation as to why you are doing so.

If you see that there are conflicts with the base (i.e. **master**) branch, please let Chris know immediately to problem solve before any pull requests are accepted.

If you want to accept the changes, select the green **Merge pull request** button. Chosing this button will open up a space for you to add a commit message that corresponds with accepting the pull request:

A screenshot of a GitHub merge pull request dialog box. On the left is a small profile picture of a man with a beard. The dialog box has a title bar that says "Merge pull request #2 from slu-soc5650/snowBranch". Below the title bar is a text input field containing the message "The extra work file looks great. Nice job @cholera-mapper!". At the bottom of the dialog box are two buttons: a green "Confirm merge" button and a grey "Cancel" button.

After filling out your commit message and selecting **Confirm merge**, you have one final opportunity to either roll back the changes (by chosing **Revert**) or make them permanent by chosing **Delete branch**.

## 9.3 Git Terminal

T

# Bibliography